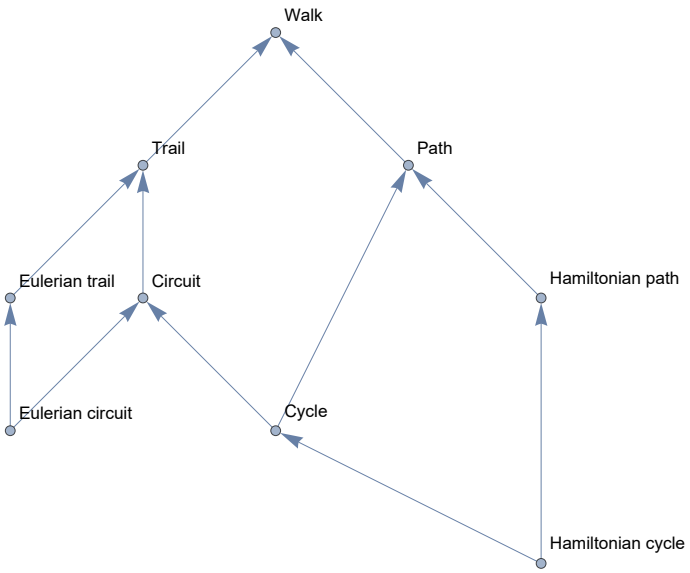
Solutions to A First Course in Graph Theory using Mathematica

Colophon

Benefits of using Mathematica: typesetting, helping with mechanics of solution, empirical testing of hypothetical solutions. Visualization and interaction help in understanding.

Taxonomy

```
Graph[
 {
  "Walk", "Trail", "Circuit", "Path", "Cycle",
  "Eulerian trail", "Eulerian circuit",
  "Hamiltonian path", "Hamiltonian cycle"},
 {
  "Trail" ↔ "Walk",
  "Circuit" ↔ "Trail",
  "Path" ↔ "Walk",
  "Cycle" ↔ "Circuit",
  "Cycle" ↔ "Path",
  "Eulerian trail" ↔ "Trail",
  "Eulerian circuit" ↔ "Circuit",
  "Eulerian circuit" ↔ "Eulerian trail",
  "Hamiltonian path" ↔ "Path",
  "Hamiltonian cycle" ↔ "Cycle",
  "Hamiltonian cycle" ↔ "Hamiltonian path"},
 VertexLabels → "Name",
 VertexCoordinates →
  {{0, 0}, {-1, -1}, {-1, -2}, {+1, -1}, {0, -3}, {-2, -2}, {-2, -3}, {+2, -2}, {+2, -4}}]
```



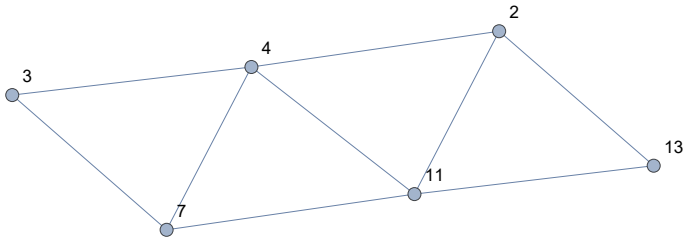
1 Introduction

1.1 Graphs and Graph Models

1.3

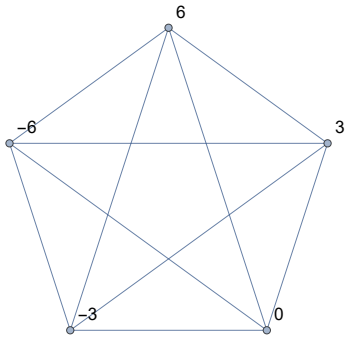
```
f[S_] :=
 UndirectedEdge @@@
  DeleteDuplicates[ (* deleting pairs of the same elements *)
   Sort /@
    Select[
     Select[ (* list of pairs of elements from S whose elements are not the same *)
      Flatten[
       Table[{i, j}, {i, S}, {j, S}],
       1],
      Apply[Unequal]],
     (* those whose sum is in S or absolute difference is in S *)
     MemberQ[S, Total[#]] ||
       MemberQ[S, Abs[#[[1]] - #[[2]]]] &]]
```

```
Graph[f[{2, 3, 4, 7, 11, 13}], VertexLabels → "Name"]
```



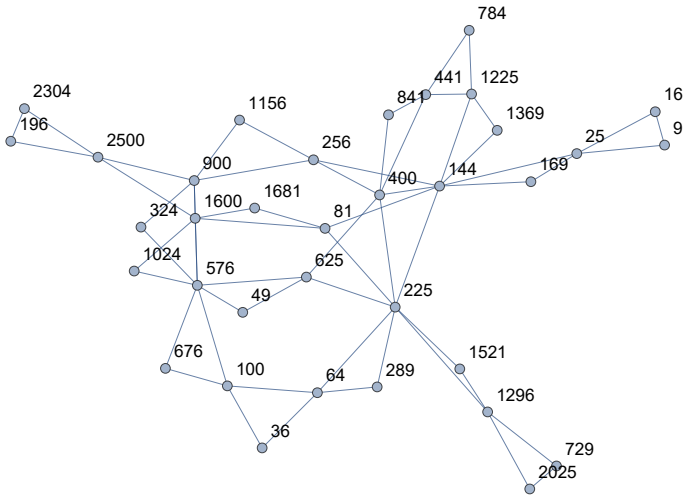
1.4

```
Graph[f[{-6, -3, 0, 3, 6}], VertexLabels → "Name", ImageSize → Small]
```



1.5

```
Graph[f[Table[i * i, {i, 50}]], VertexLabels → "Name"]
```

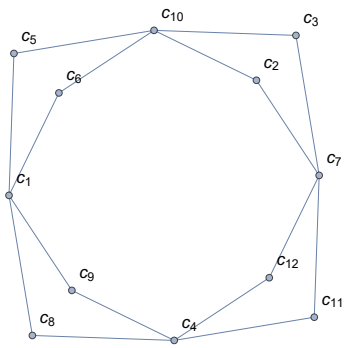


1.7a

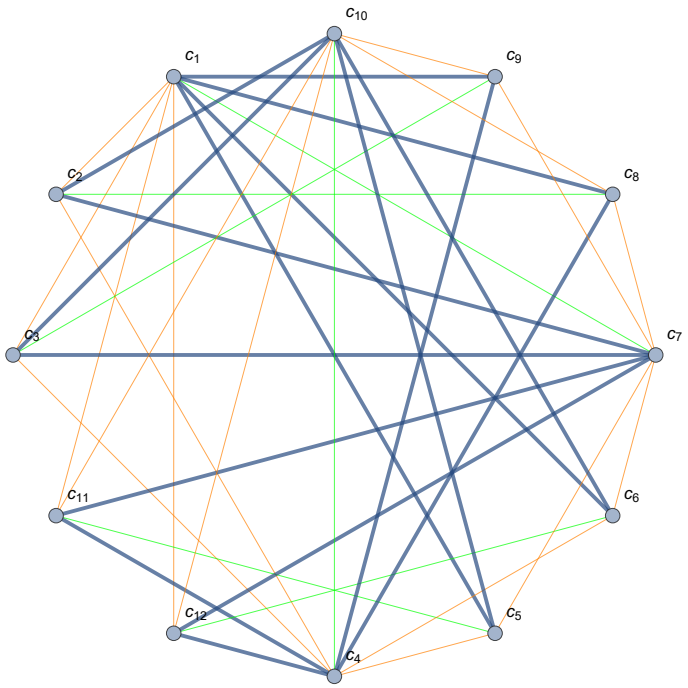
```
shift :=
 (* horizontal moves of black coin *)
 AdjacencyMatrix[Graph[Range[12], {1 ↔ 3, 2 ↔ 4, 7 ↔ 8, 9 ↔ 10}]] +
  (* horizontal moves of white coin *)
  AdjacencyMatrix[Graph[Range[12], {1 ↔ 2, 3 ↔ 4, 7 ↔ 9, 8 ↔ 10}]] +
  (* vertical moves of black coin *)
  AdjacencyMatrix[Graph[Range[12], {1 ↔ 12, 4 ↔ 6, 5 ↔ 7, 10 ↔ 11}]] +
  (* vertical moves of white coin *)
  AdjacencyMatrix[Graph[Range[12], {1 ↔ 11, 4 ↔ 5, 6 ↔ 7, 10 ↔ 12}]]
```

```
swap :=
 (* swap of coins *)
 AdjacencyMatrix[Graph[Range[12], {1 ↔ 7, 2 ↔ 8, 3 ↔ 9, 4 ↔ 10, 5 ↔ 11, 6 ↔ 12}]]
```

```
AdjacencyGraph[shift.swap, VertexLabels → Table[i → c_i, {i, 12}], ImageSize → Small]
```



```
Graph[
 Catenate[{
   (* first moves in orange edges *)
   Property[#, EdgeStyle → Orange] & /@ EdgeList[AdjacencyGraph[shift]],
   (* second moves in green edges *)
   Property[#, EdgeStyle → Green] & /@ EdgeList[AdjacencyGraph[swap]],
   (* compoound move with thick edges *)
   Property[#, EdgeStyle → Thick] & /@ EdgeList[AdjacencyGraph[shift.swap]]}],
 VertexLabels → Table[i → c_i, {i, 12}], GraphLayout -> "CircularEmbedding"]
```



1.7a

```
(* obtain the text of "As You Like It" *)
text = URLFetch["http://www.gutenberg.org/cache/epub/1121/pg1121.txt"];
```

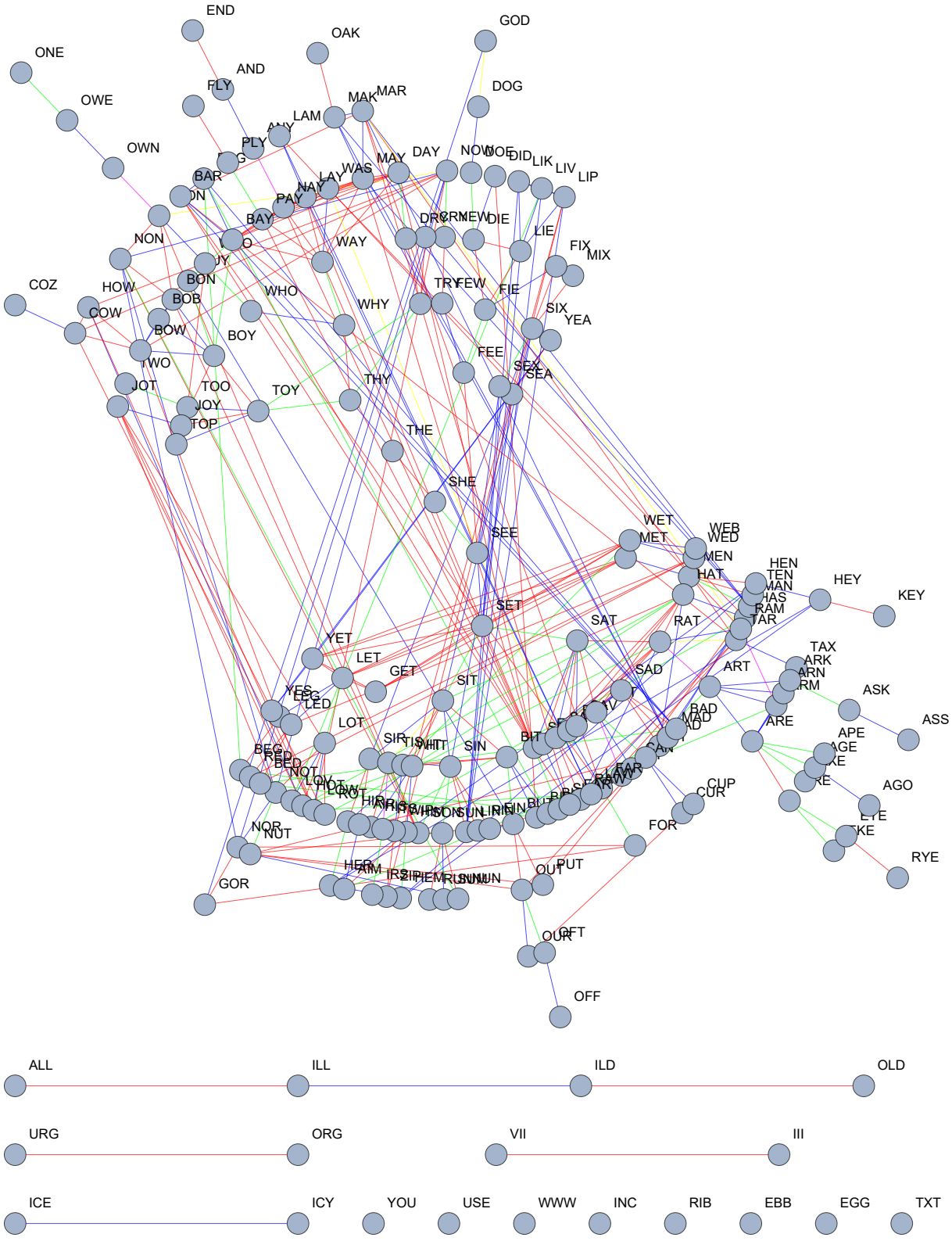
```
(* tokenize to a list of words *)
stream = StringToStream[text];
```

```
words = ReadList[
   stream,
   Word,
   WordSeparators → FromCharacterCode[
     List[#] & /@
      Join[ (* non-alphabetic ASCII characters *)
       Range[32, 64], Range[91, 96]]]];
```

```
Close[stream];
```

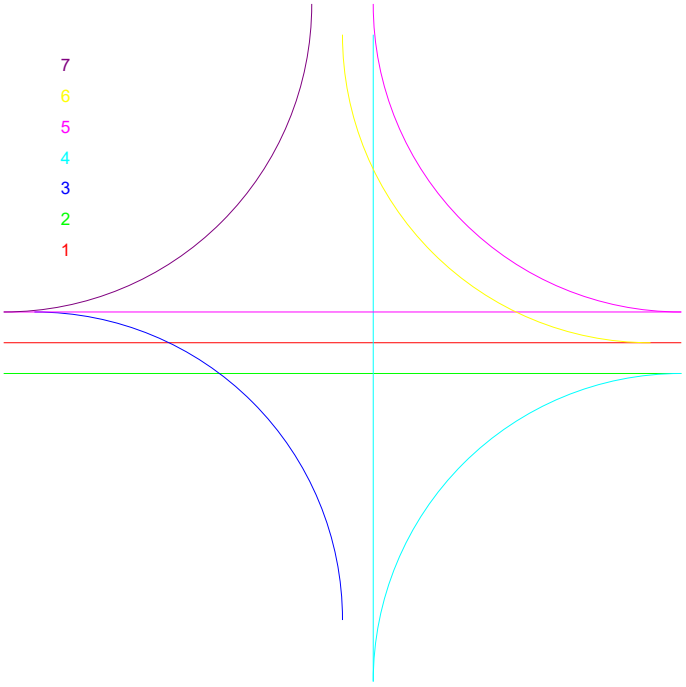
```
(* extract three-letter words, convert to upper case, and remove duplicates *)
words3 = DeleteDuplicates[ToUpperCase /@ Select[words, StringLength[#] == 3 &]]
```

```
Graph[words3,
 Flatten[ (* remove nesting introduced by Table *)
  Join[ (* join character substitution relations with permutation relations *)
   Table[ (* substitutions over each of the characters in a word *)
    Property[
       Apply[UndirectedEdge, words3[[#]]], (* convert each number pair into an edge *)
       EdgeStyle → {Red, Green, Blue}[[letter]] (* apply edge color according to letter *)
      ] & /@
     Select[
      (* list of diagonal pairs of word indexes *)
      Flatten[Table[{i, j}, {i, Length[words3]}, {j, i + 1 , Length[words3]}], 1],
      (* select word pairs that are equal with one letter removed *)
      Apply[Equal, StringDrop[#, {letter}] & /@ words3[[#]] &],
     {letter, 1, 3}],
   Table[ (* permutations between each pair of characters in a word *)
    Property[
       Apply[UndirectedEdge, words3[[#]]],
       EdgeStyle → {Magenta, Cyan, Yellow}[[letter]]
      ] & /@
     Select[
      (* list of diagonal pairs of word indexes *)
      Flatten[Table[{i, j}, {i, Length[words3]}, {j, i + 1 , Length[words3]}], 1],
      (* select words that are equal when one permutes a pair of characters *)
      Equal[
        StringJoin[
         Permute[
          (* characters of the first word *)
          Characters[words3[[First[#]]]],
          (* pick one of three permutations *)
          {Cycles[{{1, 2}}], Cycles[{{2, 3}}], Cycles[{{3, 1}}]}[[letter]]]],
        (* second word *)
        words3[[#[[2]]]]
       ] &],
    {letter, 1, 3}]],
  1],
 (* graph options *)
 VertexLabels → "Name", ImageSize -> Full, GraphLayout → "RadialDrawing"]
```

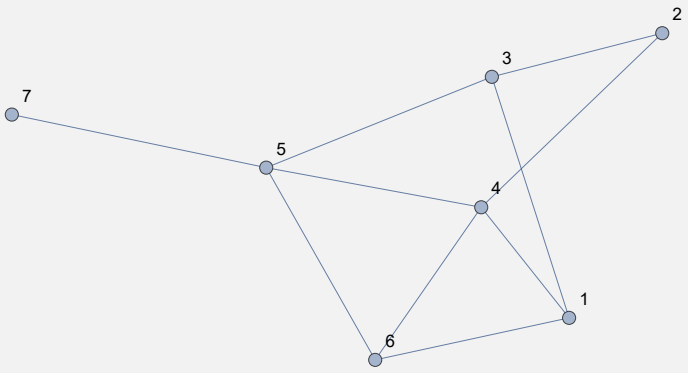

END
ONE
OAK
GOD
AND
FLY
OWE
MAK
MAR
DOG
LAM
OWN
PLY
DAY
NOW
DID
LIK
LIV
LIP
BAR
WAS
MAY
LAY
NAY
PAY
BAY
DIE
LIE
FIX
MIX
NON
WAY
COZ
HOW
BON
BOB
WHO
WHY
TRY
FEW
FIE
SIX
YEA
COW
BOW
BOY
FEE
SEX
SEA
TWO
JOT
TOO
TOY
THY
JOY
TOP
THE
SHE
WET
MET
WEB
WED
SEE
MEN
HAT
HEN
TEN
HEY
KEY
SET
SAT
RAT
TAX
ARK
ART
SAD
ASK
YET
LET
GET
SIT
LED
LOT
BAD
ARE
ASS
APE
AGE
AGO
CUP
CUR
FOR
NOR
NUT
OUT
PUT
GOR
OFT
OUR
OFF
RYE
ALL
ILL
ILD
OLD
URG
ORG
VII
III
ICE
ICY
YOU
USE
WWW
INC
RIB
EBB
EGG
TXT

1.10

```
(* define traffic flows *)
traffic = {
   (* 1 *) {Line[{{-10, 0}, {12, 0}}]},
   (* 2 *) {Line[{{-10, -1}, {12, -1}}]},
   (* 3 *) {Circle[{-9, -9}, 10, {0, π/2}]},
   (* 4 *) {Line[{{2, -11}, {2, +10}}], Circle[{+12, -11}, 10, {Pi/2, Pi}]},
   (* 5 *) {Line[{{+12, 1}, {-10, 1}}], Circle[{12, 11}, 10, {Pi, 3 Pi/2}]},
   (* 6 *) {Circle[{11, 10}, 10, {Pi, 3 Pi/2}]},
   (* 7 *) {Circle[{-10, +11}, 10, {3 Pi/2, 2 Pi}]}};
Graphics[
 MapIndexed[
  Prepend[Prepend[
     #1,
     Text[First[#2], {-8, 2 + First[#2]}]],
    {Red, Green, Blue, Cyan, Magenta, Yellow, Purple}[[First[#2]]]
   ] &,
  traffic]]
```



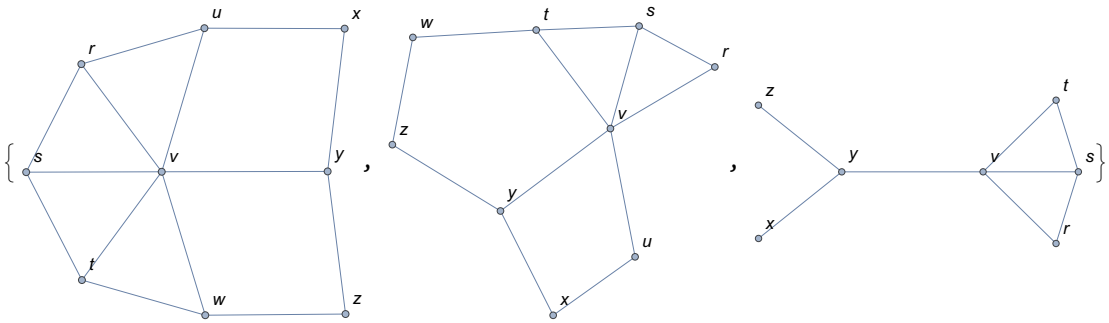
```
Graph[
 Select[
  (* pairs of lanes *)
  Flatten[Table[{i, j}, {i, Length[traffic]}, {j, i + 1, Length[traffic]}], 1],
  (* if any combination of Graphic primitives intersects *)
  Apply[
    Or,
    Flatten[Outer[ (* Cartesian product of Graphic primitives *)
      (* intersection has solution? *)
      Length[
          Solve[{x, y} ∈ #1 && {x, y} ∈ #2, {x, y}]
        ] > 0 &,
      traffic[[#[[1]]]], traffic[[#[[2]]]]
     ], 1]] &
 ],
 VertexLabels → "Name"]
```



1.2 Connected Graphs

1.11

```
G = Graph[
    {r ↔ s, s ↔ t, u ↔ v, v ↔ w, x ↔ y, y ↔ z, r ↔ u, u ↔ x, s ↔ v, v ↔ y, t ↔ w, w ↔ z, r ↔ v, v ↔ t}];
{Graph[G, VertexLabels → "Name"],
 Graph[VertexList[G], EdgeList[G, Except[r ↔ u | v ↔ w]], VertexLabels → "Name"],
 Graph[VertexList[G, Except[u | w]], EdgeList[G, Except[u ↔ _ | w ↔ _]], VertexLabels → "Name"]}
```

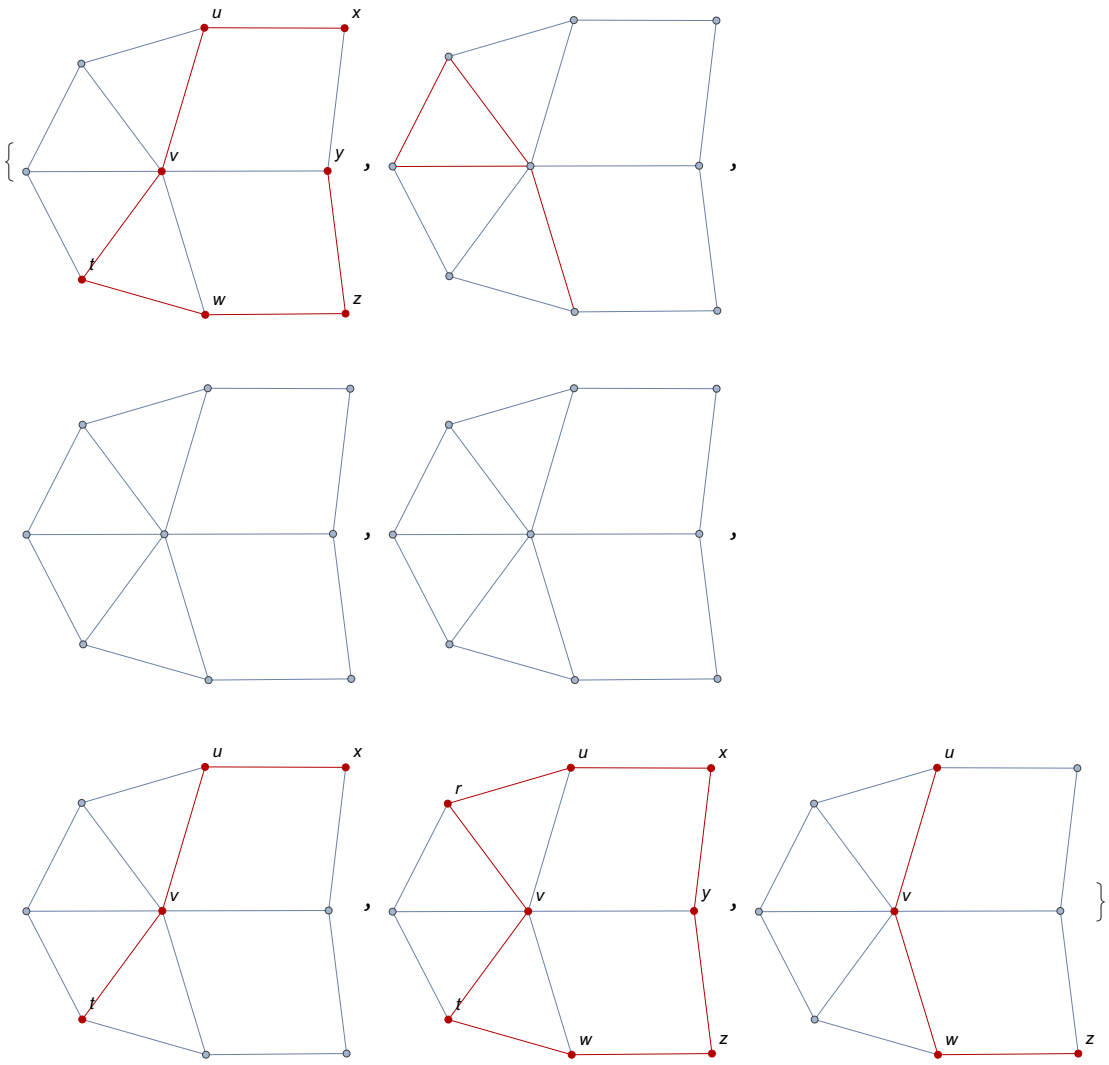


1.12

```
HighlightGraph[G, #, VertexLabels → "Name"] & /@ {
  (* a. *)
  PathGraph[First[FindPath[G, x, y, {6}]]],
  (* b. *)
  RandomChoice[
   (* adding the final edge to the ending vertex *)
   Append[#, v ⟷ w] & /@
    Select[
     (* find all cycles containing the starting vertex *)
     FindCycle[{G, v}, Infinity, All],
     (* that don't contain the ending vertex *)
     Apply[
       And, (* all of the edges *)
       (* don't match *)
       MatchQ[#, Except[_ ⟷ w]] & /@ #] &]],
  (* c. *)
  PathGraph[First[FindPath[G, r, z, {2}]]],
  (* d. *)
  PathGraph[First[FindPath[G, x, z, {3}]]],
  (* e. *)
  PathGraph[FindShortestPath[G, x, t]],
  (* g. *)
  PathGraph[First[FindCycle[G, {8}]]],
  (* h. *)
  PathGraph[
   Apply[FindShortestPath, Prepend[
     VertexList[G][[
       (* find a position in the matrix that has maximum distance of the graph *)
       Block[{GDM = GraphDistanceMatrix[G]},
        RandomChoice[Position[GDM, Max[GDM]]]]]],
     G]]]
 }
```

First::nofirst : {} has zero length and no first element. ≫

First::nofirst : {} has zero length and no first element. ≫



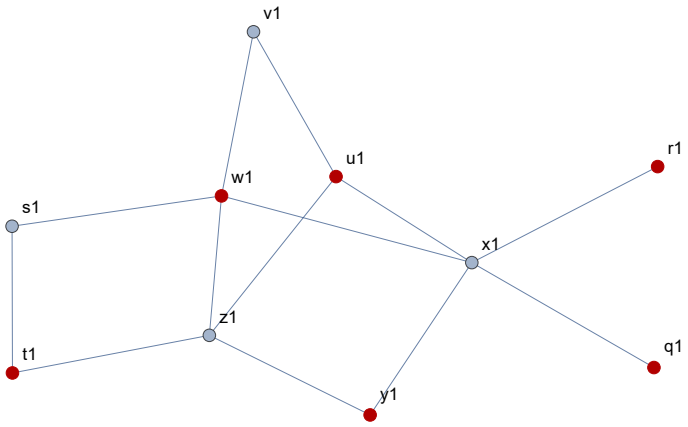
1.3 Common Classes of Graphs

```
(* Rename a sequential number-indexed graph
 with the given letter subscripted by the vertex number  *)
RenameSubscriptGraph[g_, K_] :=
 VertexReplace[g,
  Table[i → Subscript[K, i], {i, VertexCount[g]}]]
```

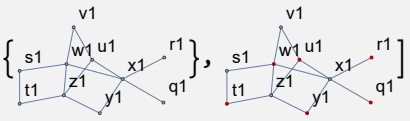
```
(* Construct the join of two graphs *)
GraphJoin[g_, h_, options__: {}] := Graph[
  Catenate[
   Append[Append[
     Outer[UndirectedEdge, VertexList[g], VertexList[h]],
     EdgeList[g]], EdgeList[h]
   ]],
  options]
```

1.24

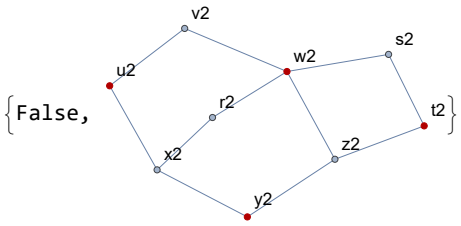
```
Module[
 {G1 = Graph[
    {r1 ⟷ x1, q1 ⟷ x1, x1 ⟷ u1, u1 ⟷ v1, v1 ⟷ w1, w1 ⟷ z1, z1 ⟷ y1,
     y1 ⟷ x1, u1 ⟷ z1, x1 ⟷ w1, w1 ⟷ s1, s1 ⟷ t1, t1 ⟷ z1}, VertexLabels → "Name"]},
 HighlightGraph[G1, FindIndependentVertexSet[G1]]]
```




```
Module[{ , }]
```



```
Block[
 {G2 = Graph[{u2 ⟷ v2, v2 ⟷ w2, w2 ⟷ z2, z2 ⟷ y2, y2 ⟷ x2, x2 ⟷ u2,
     x2 ⟷ r2, r2 ⟷ w2, w2 ⟷ s2, s2 ⟷ t2, t2 ⟷ z2}, VertexLabels → "Name"]},
 {
  BipartiteGraphQ[G2],
  HighlightGraph[G2, FindIndependentVertexSet[G2]]}]
```



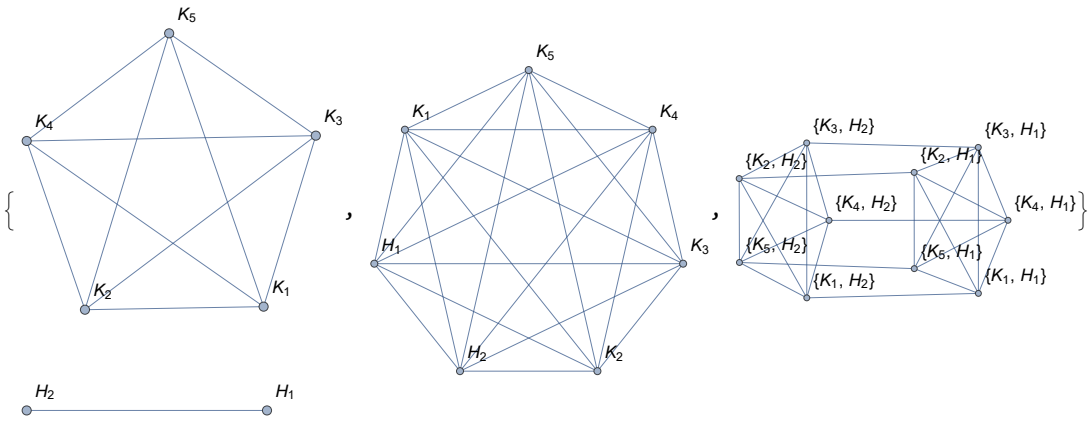
1.27

```
(* Contruct the Cartesian product of two graphs *)
GraphProduct[g_, h_, options__] := Graph[
  (* explicitly represent the product vertices, since the edge list may be empty *)
  Flatten[Outer[List, VertexList[g], VertexList[h]], 1],
  (* edges in product graph *)
  Flatten[Catenate[{
     (* all edges in G over vertices of H *)
     Function[v, {#, v} & /@ # & /@ EdgeList[g]] /@ VertexList[h],
     (* all edges in H over vertices of G *)
     Function[v, {v, #} & /@ # & /@ EdgeList[h]] /@ VertexList[g]}]],
  options]
```

```
(* display union, join, and product graphs *)
GraphShowCombinations[g_, h_] :=
 Apply[#, {g, h, VertexLabels → "Name"}] & /@ {GraphUnion, GraphJoin, GraphProduct}
```

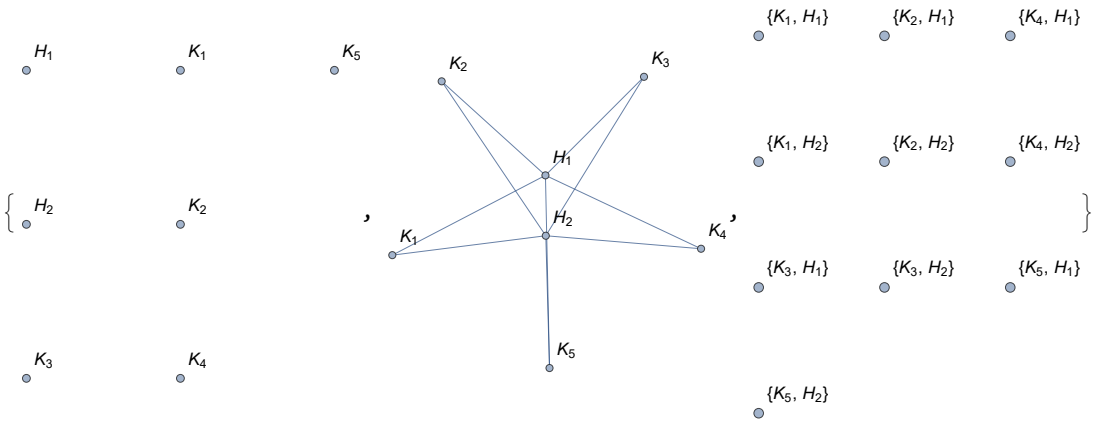
1.27a

```
GraphShowCombinations[
 RenameSubscriptGraph[CompleteGraph[5], K],
 RenameSubscriptGraph[CompleteGraph[2], H]]
```



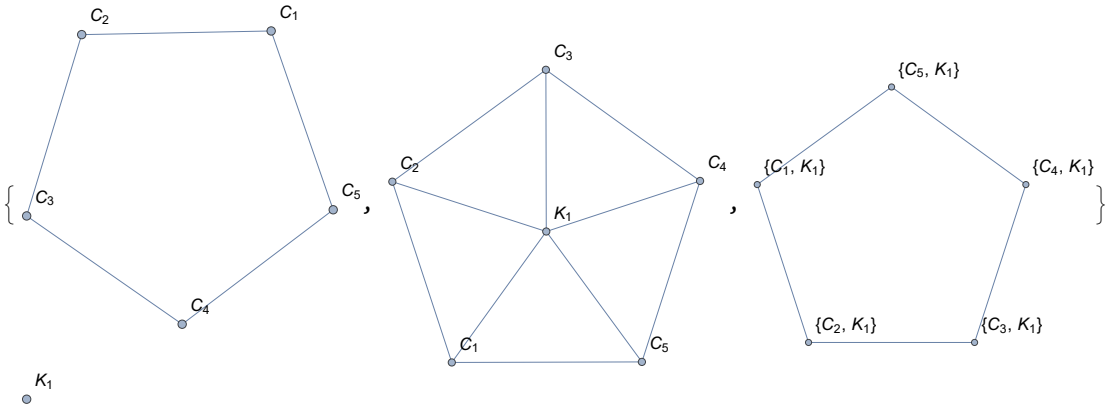
1.27b

```
GraphShowCombinations[
 Graph[Table[Subscript[K, i], {i, 5}], {}], (* empty graph with 5 vertices *)
 Graph[Table[Subscript[H, i], {i, 2}], {}]] (* empty graph with 2 vertices *)
```



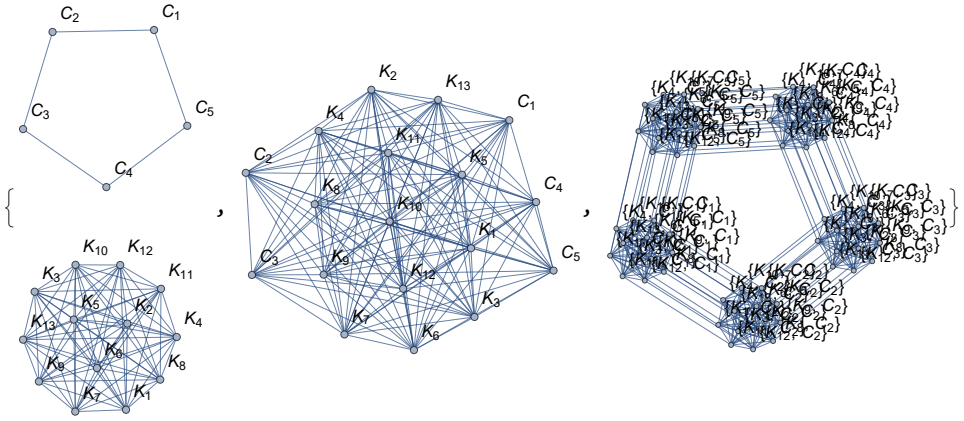
1.27c

```
GraphShowCombinations[
 RenameSubscriptGraph[CycleGraph[5], C],
 RenameSubscriptGraph[CompleteGraph[1], K]]
```

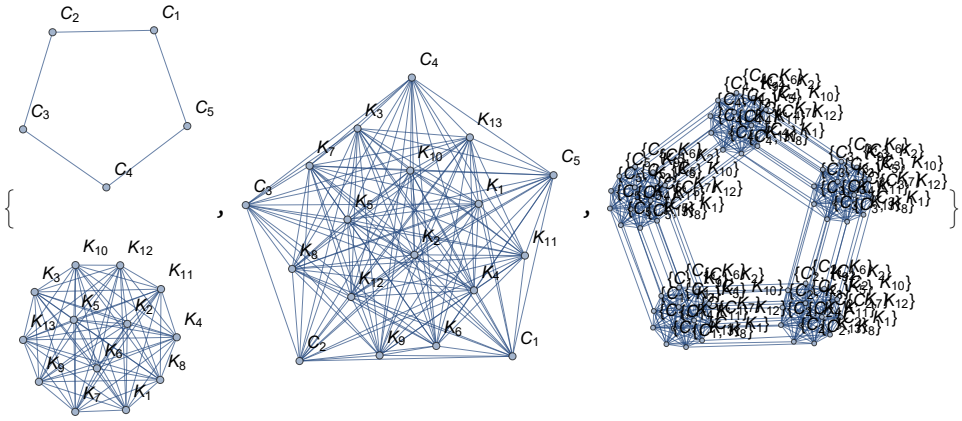


1.27 extra

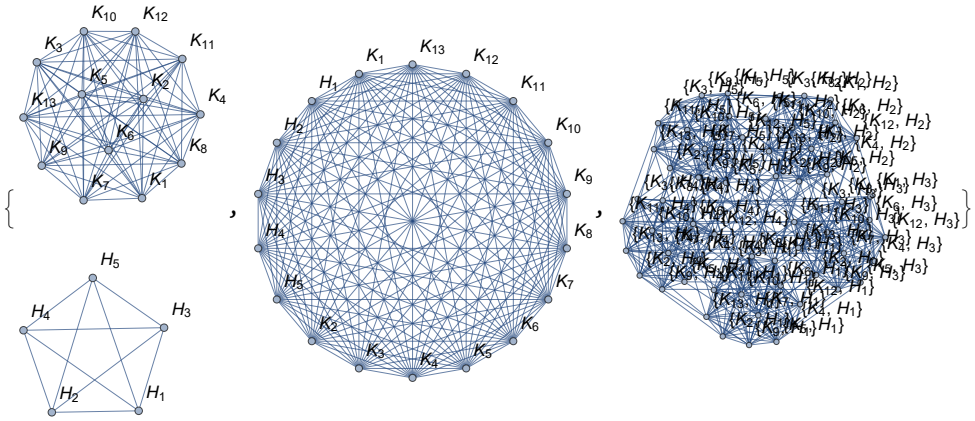
```
GraphShowCombinations[
 RenameSubscriptGraph[CompleteGraph[13], K],
 RenameSubscriptGraph[CycleGraph[5], C]]
```



```
GraphShowCombinations[
 RenameSubscriptGraph[CycleGraph[5], C],
 RenameSubscriptGraph[CompleteGraph[13], K]]
```



```
GraphShowCombinations[
 RenameSubscriptGraph[CompleteGraph[13], K],
 RenameSubscriptGraph[CompleteGraph[5], H]]
```



2 Degrees

2.1 The Degree of a Vertex

2.1a

The putative graph would have an odd number of odd vertices, which is impossible.

2.1b

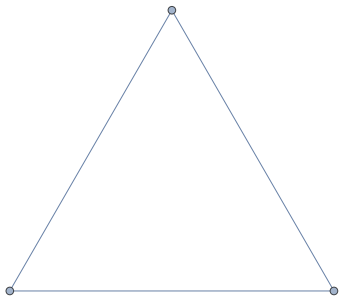
The putative graph would have a vertex with the same degree as the graph order, which is impossible for a non-pseudograph.

2.1c

The vertex of degree 1 would have to be connected to a vertex of degree 3. The remaining subgraph would have three vertices with degrees {2, 3, 3} that includes vertices with the same degree as the order.

2.2a

```
CycleGraph[3, ImageSize → Small]
```

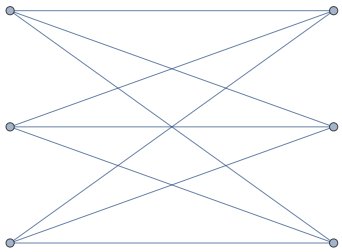


2.2b

```
GraphFromNumericalEdgeList[n_, edges_] :=
 Module[
  {nf},
  nf := FromCharacterCode[{96 + #}] &;
  Graph[
   nf /@ Range[n],
   Map[nf, edges, {2}],
   VertexLabels → "Name"]]
```

Since vertices have degree 3, the order of the graph must be at least 4. For the graph not to be connected, its order must be at least 5. Since a graph must have an even number of odd vertices, its order must be even. Therefore the first viable order can only be 6. See $K_{3,3}$:

```
g22b = CompleteGraph[{3, 3}, ImageSize → Small]
```



```
ContainsOnly[VertexDegree[g22b], {3}]
```

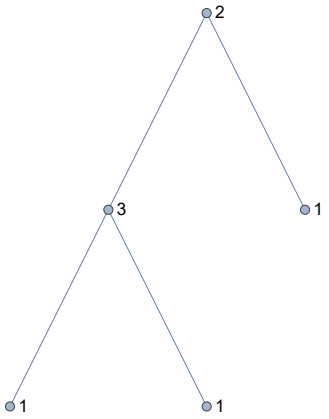
```
True
```

```
CompleteGraphQ[g22b]
```

```
False
```

2.2c

```
g22c = Graph[{a ⟷ b, b ⟷ c, a ⟷ d, a ⟷ e}, VertexLabels → "VertexDegree", ImageSize → Small]
```




```
AllTrue[ (* for all edges *)
 Map[ (* convert vertices to their respective degree *)
  VertexDegree[g22c, #] &,
  List @@@ EdgeList[g22c], (* convert list of UndirectedEdge to list of list *)
  {2}],
 (* degrees are not equal *)
 Apply[Unequal]]
```

```
True
```

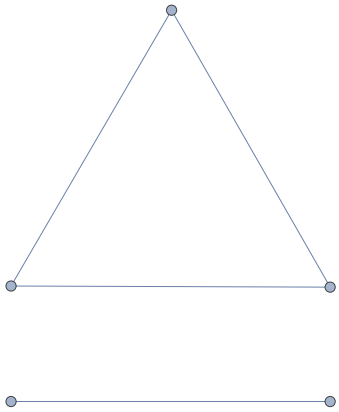
```
VertexCount[g22c] ≥ 5
```

```
True
```

2.2d

Trivially, a complete graph has no nonadjacent vertices. A nontrivial option:

```
g22d = Graph[{1 ⟷ 2, 3 ⟷ 4, 4 ⟷ 5, 3 ⟷ 5}, ImageSize → Small]
```



```
AllTrue[ (* for all edges *)
 Map[ (* convert vertices to their respective degree *)
  VertexDegree[g22d, #] &,
  List @@@ (* convert list of UndirectedEdge to list of list *)
   (* list of nonadjacent vertices *)
   Complement[EdgeList[CompleteGraph[5]], EdgeList[g22d]],
  {2}],
 (* degrees are not equal *)
 Apply[Unequal]]
```

```
True
```

2.3

```
Solve[n×4 + (12 - n) 6 == 2 × 31, n]
```

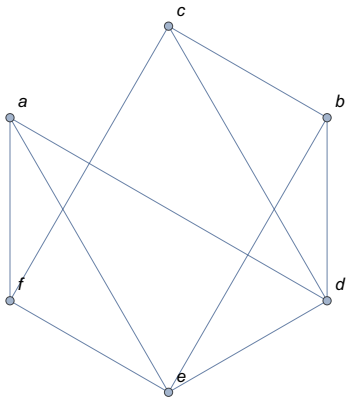
```
{{n → 5}}
```

2.4

```
Solve[n×3 + (6 - n) 4 == 2×10, n]
```

```
{{n → 4}}
```

```
g24 = Graph[{a ↔ f, a ↔ e, a ↔ d, b ↔ e, b ↔ d, b ↔ c, c ↔ f, c ↔ d, f ↔ e, d ↔ e},
  GraphLayout → "CircularEmbedding", VertexLabels → "Name", ImageSize → Small]
```




```
VertexCount[g24]
```

6

```
EdgeCount[g24]
```

10

2.5

```
Solve[n×3 + 2×4 + (25 − 2 − 11 − n) ×5 + 11×6 == 2×62, n]
```

{{n → 5}}

2.6

Such a graph has the following number of edges, which can only be integral if $3\,n^2$ is even, which is to say n^2 is even, which implies that n is even.

```
Simplify[1/2 (n (n + 1) + n n + n (n − 1))]
```

$$\frac{3\,n^2}{2}$$

2.7a

Since the graph is bipartite, each edge is incident in each partite set.

2.7b

```
Solve[12×3 == (22 - 12 - n) ×4 + n×2, n]
```

{{n → 2}}

2.8

```
(* all pairwise nonadjacent vertices have degrees that sum to ≥ 4*)
AllTrue[
 (* sum of degrees of pairwise nonadjacent vertices *)
 Total[
    (* list of degrees of pairwise nonadjacent vertices *)
    VertexDegree[CycleGraph[6], #] & /@
     #] & /@
  (* find cliques of 3 pairwise nonadjacent vertices of 6-cycle *)
  FindClique[
   Graph[
    (* find list of nonadjacent edges of 6-cycle *)
    Complement[
     EdgeList[CompleteGraph[6]],
     EdgeList[CycleGraph[6]]]],
   {3},
   All],
 GreaterThan[4]]
```

True

2.9

Each component can be regarded as a graph, which cannot have an odd number of odd vertices.

2.10a

The sum of the degrees of the first and last vertices is 2.

```
PathGraph[Range[4], VertexLabels → "VertexDegree"]
```

2.10b

Suppose G has a component K of order k. Then $G' = G - K$ is a graph of order $n' = n - k$ in which $\deg u + \deg v \geq n - 2 = n' + k - 2$, and if $k \geq 1$ then $\deg u + \deg v \geq n' - 1$ so that G' is connected. Therefore G can have at most two components.

2.10c

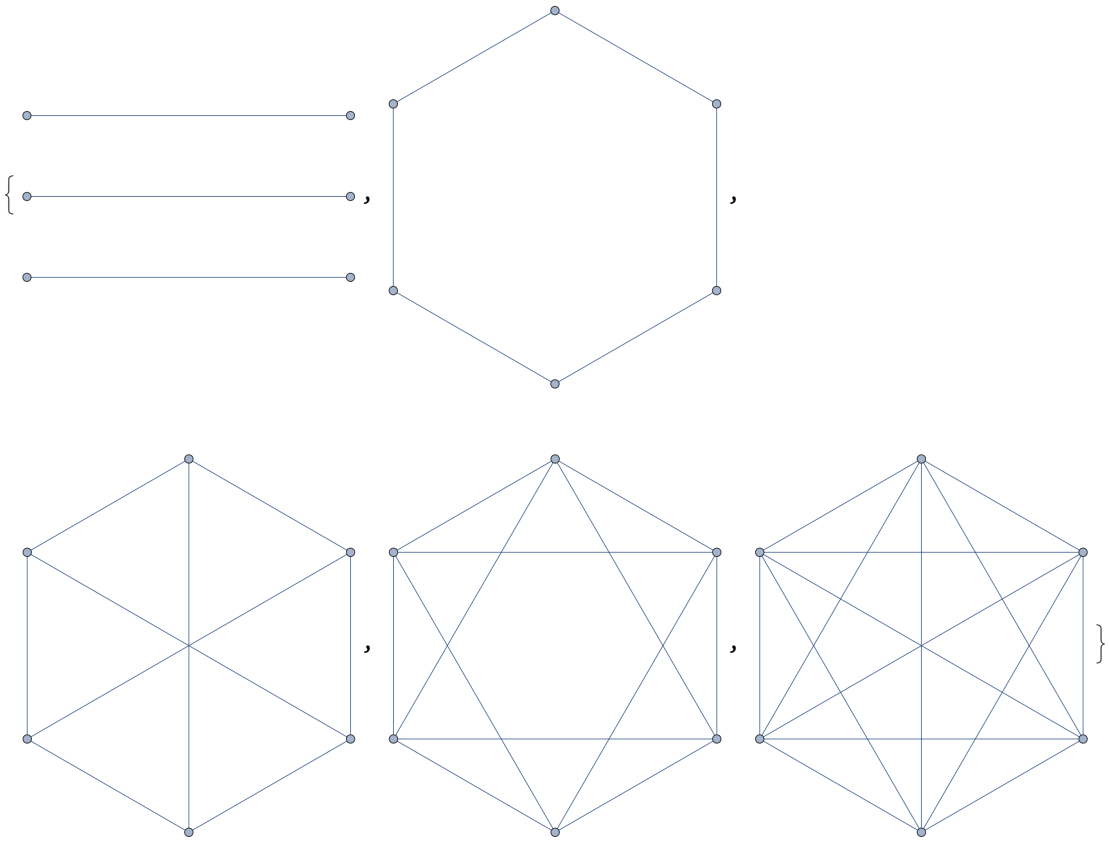
2.11

No: consider the disconnected graph with two components K_2; $\delta K_2 = 1 \geq \frac{4-2}{2}$.

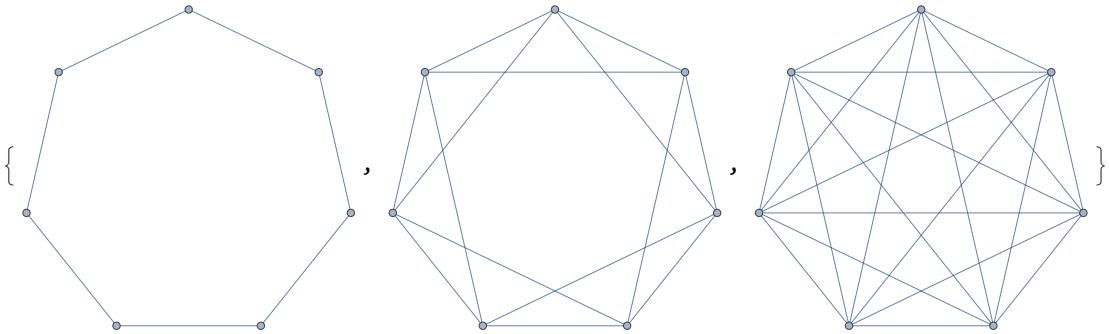
2.2 Regular Graphs

2.19

```
Table[If[i == 1, Graph[{1 ⟷ 4, 2 ⟷ 5, 3 ⟷ 6}], HararyGraph[i, 6]], {i, 1, 5}]
```



```
Table[HararyGraph[i, 7], {i, {2, 4, 6}}]
```

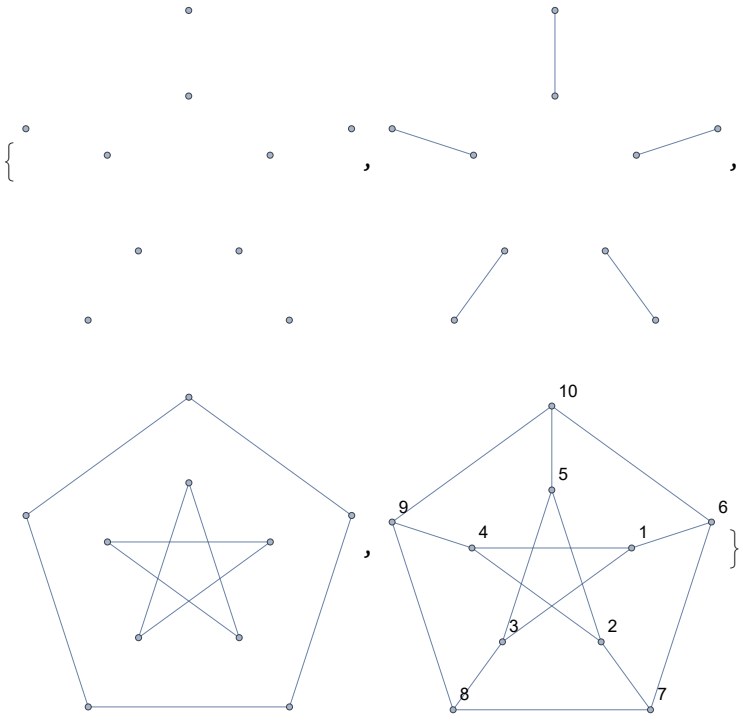


2.20

By definition; otherwise it would be regular.

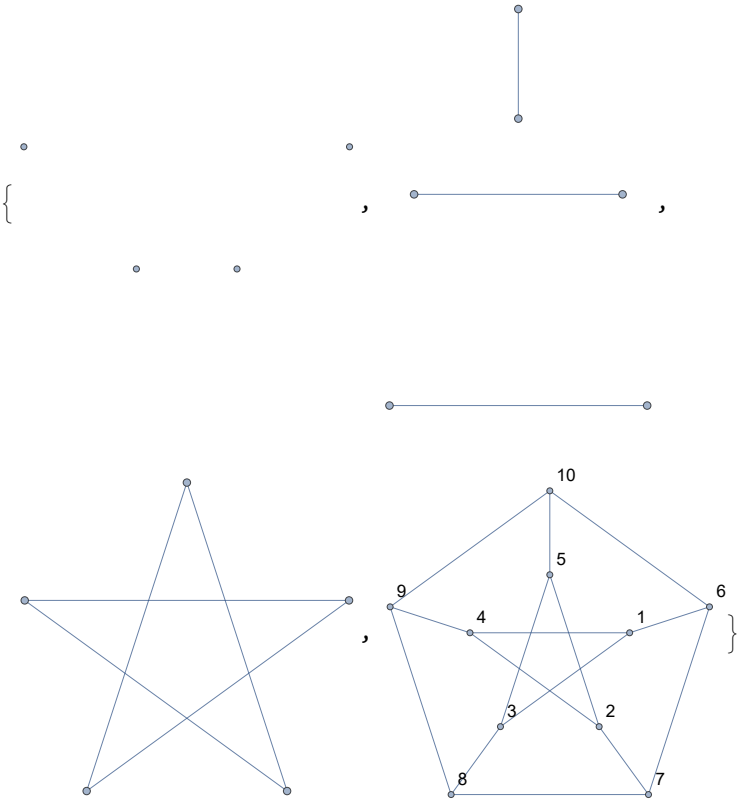
2.21a

```
Table[
 (* EdgeDelete of empty list hangs the kernel *)
 If[Length[el] == 0, PetersenGraph[VertexLabels → "Name"], EdgeDelete[PetersenGraph[], el]],
 {el, {
   EdgeList[PetersenGraph[]],
   {1 ↔ 4, 4 ↔ 2, 2 ↔ 5, 5 ↔ 3, 3 ↔ 1, 6 ↔ 7, 7 ↔ 8, 8 ↔ 9, 9 ↔ 10, 10 ↔ 6},
   {1 ↔ 6, 2 ↔ 7, 3 ↔ 8, 4 ↔ 9, 5 ↔ 10},
   {}}}]
```



2.21b

```
Table[
 If[Length[vl] == 0, PetersenGraph[VertexLabels → "Name"], VertexDelete[PetersenGraph[], vl]],
 {vl, {
   {1, 4, 5, 7, 8, 10},
   {2, 3, 6, 9},
   {6, 7, 8, 9, 10},
   {}
  }}]
```

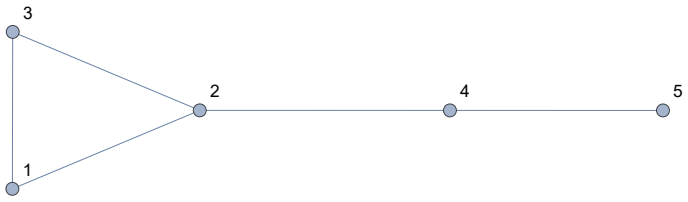


2.21c

A more challenging problem would be to ask for a *maximal* induced subgraph; i.e., one with a minimal number of vertices removed.

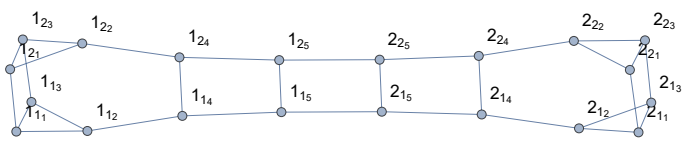
2.22a

```
g222 = Graph[{1 ⟷ 2, 2 ⟷ 3, 3 ⟷ 1, 2 ⟷ 4, 4 ⟷ 5}, VertexLabels → "Name"]
```




```
ConstructByTheorem27[g_] :=
 Module[
  {mvd = Max[VertexDegree[g]]},
  If[
   (* is regular graph? *)
   AllTrue[VertexDegree[g], EqualTo[mvd]],
   (* then done *)
   g,
   (* otherwise, recurse on *)
   ConstructByTheorem27[
    (* a new graph containing *)
    EdgeAdd[
     GraphUnion[
      (* two copies of the original graph *)
      VertexReplace[g, # → 1# & /@ VertexList[g]],
      VertexReplace[g, # → 2# & /@ VertexList[g]],
      VertexLabels → "Name"
     ],
     (* and edges added between vertices of the copies *)
     1# ⟷ 2# & /@
      Select[
       VertexList[g],
       (* that are not of maximal degree *)
       VertexDegree[g, #] < mvd &]]]]]
```

```
g222a = ConstructByTheorem27[g222]
```

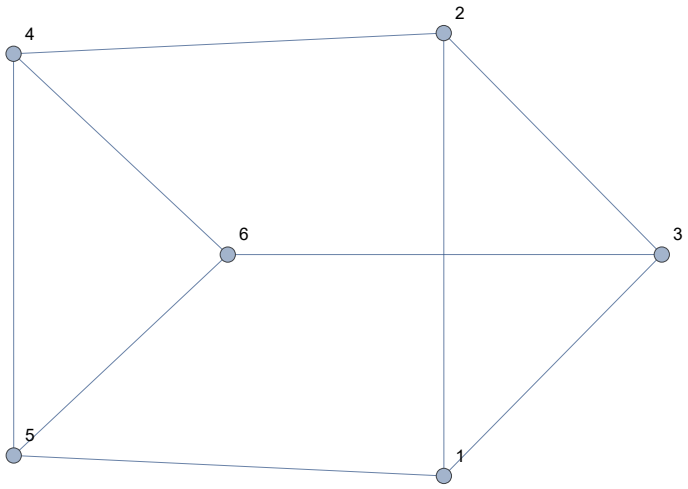



```
VertexCount[g222a]
```

```
20
```

2.22b

```
g222b = EdgeAdd[g22, {4 ⟷ 6, 5 ⟷ 6, 1 ⟷ 5, 3 ⟷ 6}]
```




```
AllTrue[VertexDegree[g222b], EqualTo[3]]
```

```
True
```

```
VertexCount[g222b]
```

```
6
```

2.3 Degree Sequences

2.31

The degree sequence describes the complement graph, and therefore is graphical iff the ‘uncomplement’ sequence is graphical.

2.32

```
IsSequenceGraphical[seq_List] :=
 Module[{s0 = Select[seq, # > 0 &]},
  Length[s0] == 0 || (
    Length[Rest[s0]] ≥ First[s0] &&
     Module[
      {s = Sort[
         MapIndexed[
          If[First[#2] ≤ First[s0], #1 - 1, #1] &,
          Rest[s0]],
         Greater]},
       If[
        OddQ[Length[Select[s, OddQ]]],
        False,
        IsSequenceGraphical[s]]])]
```

```
IsSequenceGraphical[{5, 3, 3, 3, 3, 2, 2, 1}]
```

True

```
IsSequenceGraphical[{6, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1}]
```

True

```
IsSequenceGraphical[{6, 5, 5, 4, 3, 2, 1}]
```

False

```
IsSequenceGraphical[{7, 5, 4, 4, 4, 3, 2, 1}]
```

True

```
IsSequenceGraphical[{7, 6, 5, 4, 4, 3, 2, 1}]
```

True

2.33

```
NoneTrue[Table[IsSequenceGraphical[{x, 1, 2, 3, 5, 5}], {x, 0, 5}], TrueQ]
```

True

2.34

```
Select[Range[0, 7], IsSequenceGraphical[{#, 7, 6, 5, 4, 3, 2, 1}] &]
```

```
{4}
```

2.35

```
Select[Range[0, 7], IsSequenceGraphical[{#, 7, 7, 5, 5, 4, 3, 2}] &]
```

```
{3, 5}
```

2.36

Ugly: don't know how to return the first value for which a predicate holds, or how to return the value of the iterator from a loop.

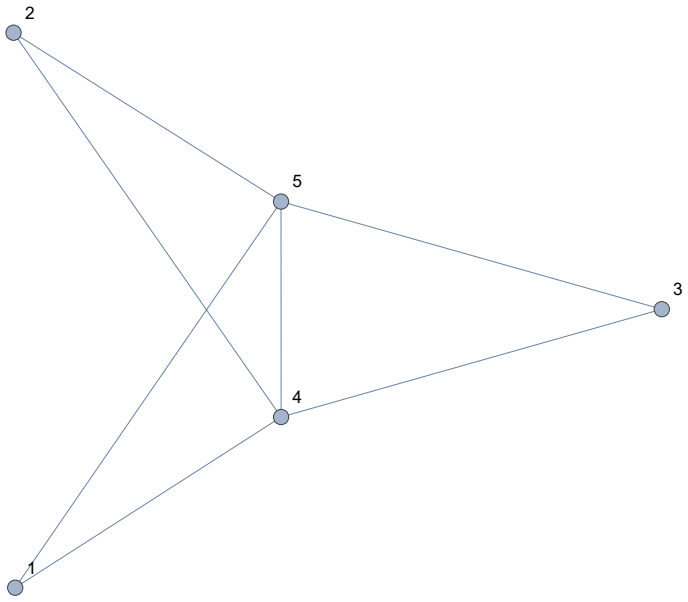
```
For[
k = 1,
Not[IsSequenceGraphical[Flatten[ConstantArray[{2, 6, 7}, k]]]],
k++,
Print[k]]
```

```
1

2

3
```

2.4 Graphs and Matrices

2.37

```
g237 = Graph[
Range[5] (* let the vertices appear in deterministic order *),
{1 ↔ 4, 1 ↔ 5, 2 ↔ 4, 2 ↔ 5, 3 ↔ 4, 3 ↔ 5, 4 ↔ 5},
VertexLabels → "Name"]
```




```
{{2, 2, 2, 1, 1}, {2, 2, 2, 1, 1}, {2, 2, 2, 1, 1}, {1, 1, 1, 4, 3}, {1, 1, 1, 3, 4}} ==
MatrixPower[AdjacencyMatrix[g237] , 2]
```

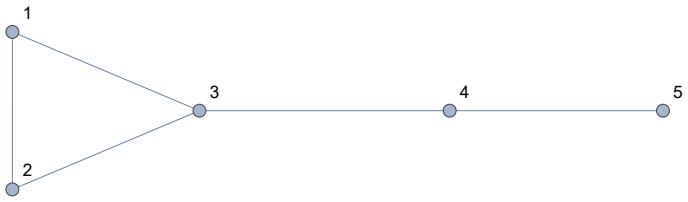
```
True
```

```
{{2, 2, 2, 3 + 4, 3 + 4}, {2, 2, 2, 3 + 4, 3 + 4}, {2, 2, 2, 3 + 4, 3 + 4}, {3 + 4, 3 + 4, 3 + 4, 3 + 3, 3 + 4},
  {3 + 4, 3 + 4, 3 + 4, 3 + 4, 3 + 3}} == MatrixPower[AdjacencyMatrix[g237], 3]
```

```
True
```

2.38

```
g238 = Graph[Range[5], {1 ⟷ 2, 2 ⟷ 3, 3 ⟷ 1, 3 ⟷ 4, 4 ⟷ 5}, VertexLabels → "Name"]
```




```
{{2, 1, 1, 1, 0}, {1, 2, 1, 1, 0}, {1, 1, 3, 0, 1}, {1, 1,
  0, 2, 0}, {0, 0, 1, 0, 1}} == MatrixPower[AdjacencyMatrix[g238], 2]
```

```
True
```

```
{{2, 2 + 1, 3 + 1, 1, 1}, {2 + 1, 2, 3 + 1, 1, 1}, {3 + 1, 3 + 1, 2, 2 + 2, 0},
  {1, 1, 3 + 1, 0, 1 + 1}, {1, 1, 0, 2, 0}} == MatrixPower[AdjacencyMatrix[g238], 3]
```

```
True
```

2.39

```
g239 = Graph[Range[4], {1 ⟷ 2, 2 ⟷ 3, 3 ⟷ 4}, VertexLabels → "Name"]
```


```
{{2, 0, 2 + 1, 0}, {0, 2 + 2 + 1, 0, 1 + 2}, {1 + 1 + 1, 0, 2 +
  2 + 1, 0}, {0, 2 + 1, 0, 2}} == MatrixPower[AdjacencyMatrix[g239], 4]
```

```
True
```

2.40

```
BipartiteAdjacencyMatrix2[r_] :=
Flatten[{
ConstantArray[Flatten[{ConstantArray[r, r], ConstantArray[0, r]}], r],
ConstantArray[Flatten[{ConstantArray[0, r], ConstantArray[r, r]}], r]},
1]
```

```
BAM2Correct[r_] := BipartiteAdjacencyMatrix2[
  r] == MatrixPower[AdjacencyMatrix[CompleteGraph[{r, r}]], 2]
```

```
AllTrue[Range[10], BAM2Correct]
```

```
True
```

```
BipartiteAdjacencyMatrix3[r_] :=
Flatten[{
ConstantArray[Flatten[{ConstantArray[0, r], ConstantArray[r^2, r]}], r],
ConstantArray[Flatten[{ConstantArray[r^2, r], ConstantArray[0, r]}], r]},
1]
```

```
BAM3Correct[r_] := BipartiteAdjacencyMatrix3[r] ==
MatrixPower[AdjacencyMatrix[CompleteGraph[{r, r}]], 3]
```

```
AllTrue[Range[10], BAM3Correct]
```

True

```
BipartiteAdjacencyMatrix4[r_] :=
Flatten[{
ConstantArray[Flatten[{ConstantArray[r^3, r], ConstantArray[0, r]}], r],
ConstantArray[Flatten[{ConstantArray[0, r], ConstantArray[r^3, r]}], r]},
1]
```

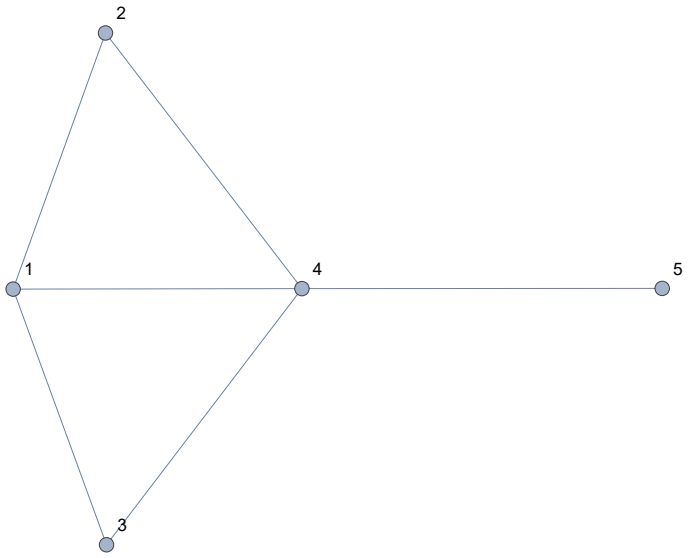
```
BAM4Correct[r_] := BipartiteAdjacencyMatrix4[r] ==
MatrixPower[AdjacencyMatrix[CompleteGraph[{r, r}]], 4]
```

```
AllTrue[Range[10], BAM4Correct]
```

True

2.41

```
g240 = Graph[Range[5], {1 ↔ 2, 1 ↔ 3, 1 ↔ 4, 2 ↔ 4, 3 ↔ 4, 4 ↔ 5}, VertexLabels → "Name"]
```



```
IncidenceMatrix[g240] .Transpose[IncidenceMatrix[g240]] // MatrixForm
```

$$\begin{pmatrix} 3 & 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 & 0 \\ 1 & 0 & 2 & 1 & 0 \\ 1 & 1 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The matrix represents the number of incident edges in common between vertices i and j.

2.5 Irregular Graphs

```
RegularGraphQ[G_] := Module[
  {vd = VertexDegree[G]},
  AllTrue[
   Rest[vd],
   EqualTo[First[vd]]]]
```

```
(* Calculate the K-degrees of the vertices of G *)
KDegrees[G_, K_] :=
 Lookup[#[[1]] → #[[2]] & /@ (* convert tally list to rules *)
   Tally[ (* tally number of occurrences of each vertex *)
    Flatten[
     VertexList /@
      Select[ (* selecting only those graphs with the same size and order of K *)
       Flatten[
        Module[
            {sg = Subgraph[G, #]}, (* construct a subgraph out of the vertex subset *)
            (* construct new graphs having the same vertex set over edge lists *)
            Graph[VertexList[sg], #] & /@
             (* construct edge lists from the subgraph,
             but having the same number of edges as K *)
             Subsets[EdgeList[sg], {EdgeCount[K]}]
           ] & /@
         (* all the subsets of vertices with the same number of vertices as K *)
         Subsets[VertexList[G], {VertexCount[K]}]],
       (* that are actually isomorphic to K *)
       IsomorphicGraphQ[#, K] &]]],
  VertexList[G], (* for each of the graph's vertices *)
  0] (* degree is 0 by default if the vertex never appeared *)
```

```
KSubgraphHighlight[G_, K_] := Table[
  HighlightGraph[G, kg],
  {kg,
   Select[
    Flatten[Module[
        {sg = Subgraph[G, #]},
        Graph[VertexList[sg], #] & /@
         Subsets[EdgeList[sg], {EdgeCount[K]}]
       ] & /@ Subsets[VertexList[G], {VertexCount[K]}]],
    IsomorphicGraphQ[#, K] &]}]
```

2.42

The following graph is K_4-regular of degree 1, but P_2-irregular (all vertices have degree 3, but a_1 and b_1 have degree 4).

```
g242H1 = GraphUnion[
  RenameSubscriptGraph[CompleteGraph[4], a],
  RenameSubscriptGraph[CompleteGraph[4], b],
  Graph[{a1 ⟷ b1}],
  VertexLabels → "Name"]
```

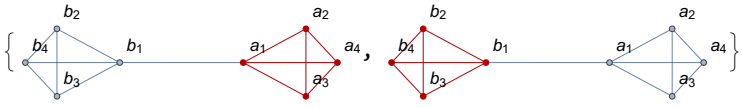

```
RegularGraphQ[g242H1]
```

```
False
```

```
KDegrees[g242H1, CompleteGraph[4]]
```

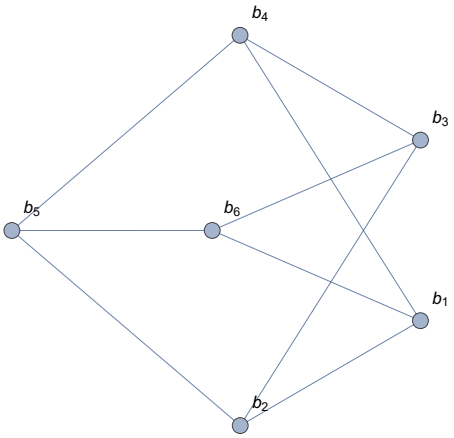
```
{1, 1, 1, 1, 1, 1, 1, 1}
```

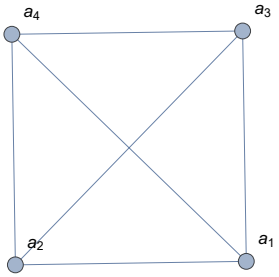
```
KSubgraphHighlight[g242H1, CompleteGraph[4]]
```



The following graph is K_4-irregular (the vertices in the Harary graph have degree 0), but P_2-regular of degree 3.

```
g242H2 = GraphUnion[
  RenameSubscriptGraph[CompleteGraph[4], a],
  RenameSubscriptGraph[HararyGraph[3, 6], b],
  VertexLabels → "Name"]
```






```
RegularGraphQ[g242H2]
```

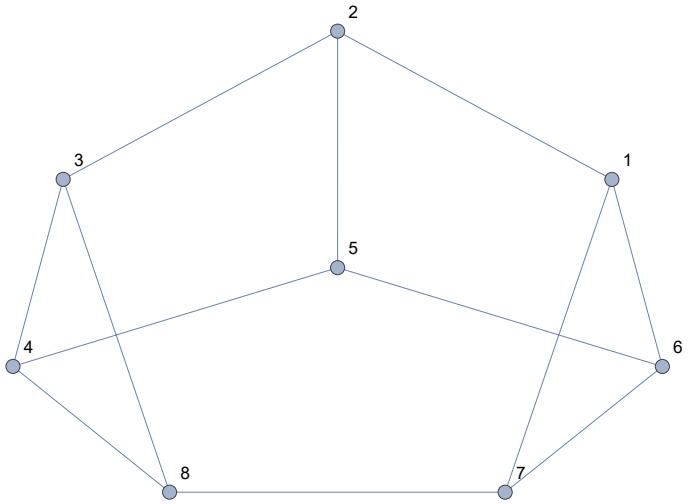
```
True
```

```
KDegrees[g242H2, CompleteGraph[4]]
```

```
{1, 1, 1, 1, 0, 0, 0, 0, 0, 0}
```

2.43

```
g243 = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 5 ↔ 6,
    6 ↔ 1, 2 ↔ 5, 1 ↔ 7, 6 ↔ 7, 3 ↔ 8, 4 ↔ 8, 8 ↔ 7}, VertexLabels → "Name"]
```

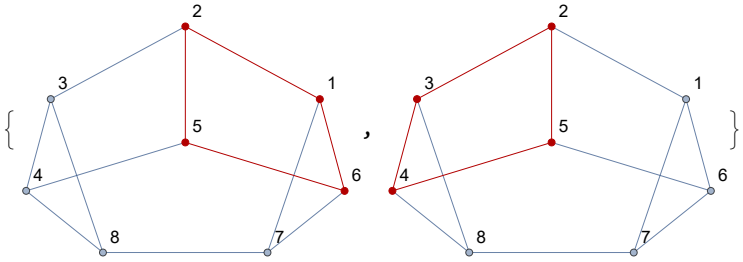


Vertices 2 and 5 have C_4-degree 2; vertices 7 and 8 have C_4-degree 0.

```
KDegrees[g243, CycleGraph[4]]
```

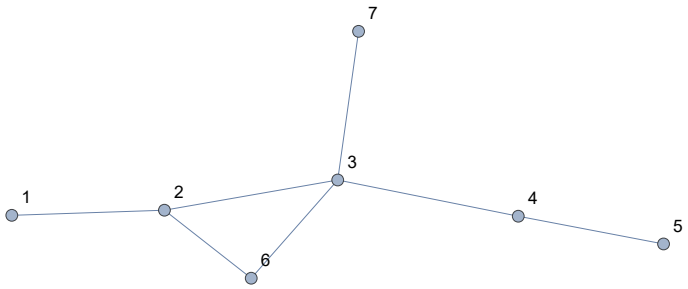
```
{1, 2, 1, 1, 2, 1, 0, 0}
```

```
KSubgraphHighlight[g243, CycleGraph[4]]
```



2.44

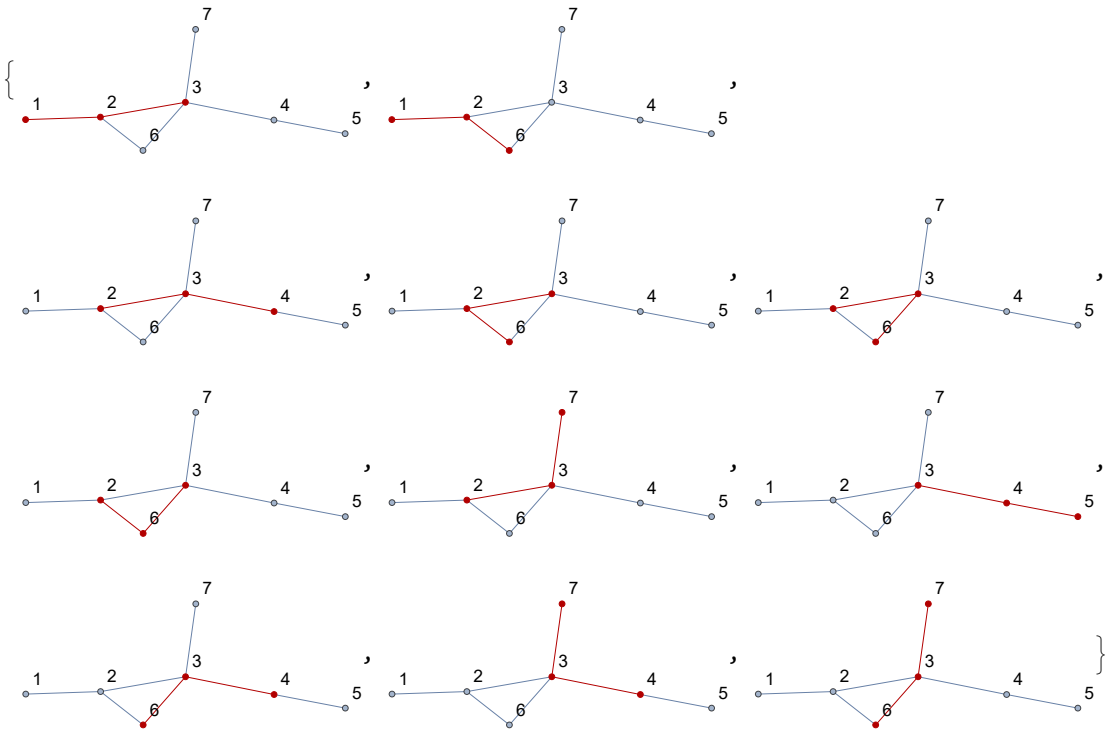
```
g244 = Graph[{1 ⟷ 2, 2 ⟷ 3, 3 ⟷ 4, 4 ⟷ 5, 2 ⟷ 6, 6 ⟷ 3, 3 ⟷ 7}, VertexLabels → "Name"]
```




```
KDegrees[g244, PathGraph[Range[3]]]
```

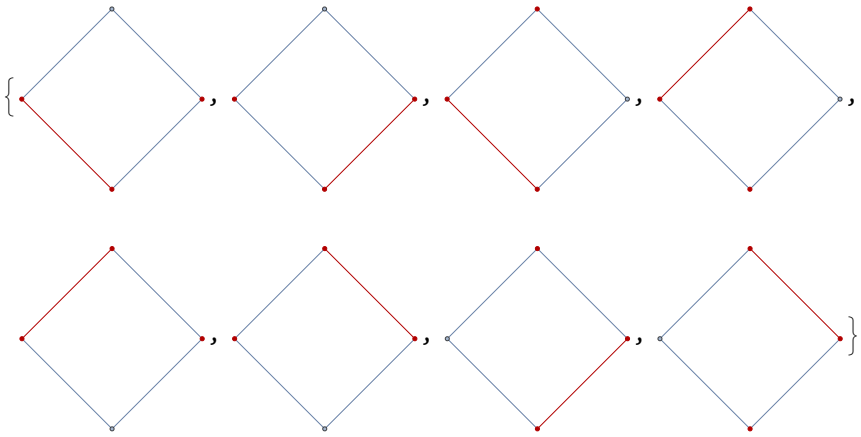
```
{2, 7, 10, 4, 1, 6, 3}
```

```
KSubgraphHighlight[g244, PathGraph[Range[3]]]
```



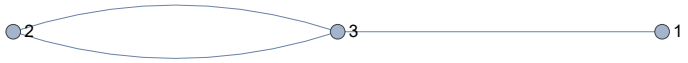
2.45

```
KSubgraphHighlight[CycleGraph[4, ImageSize → Tiny], Graph[Range[3], {2 ↔ 3}]]
```



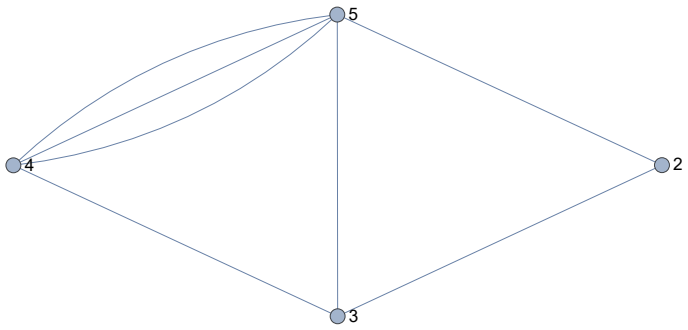
2.46

```
Graph[{1 ↔ 2, 1 ↔ 2, 2 ↔ 3}, VertexLabels → "VertexDegree"]
```

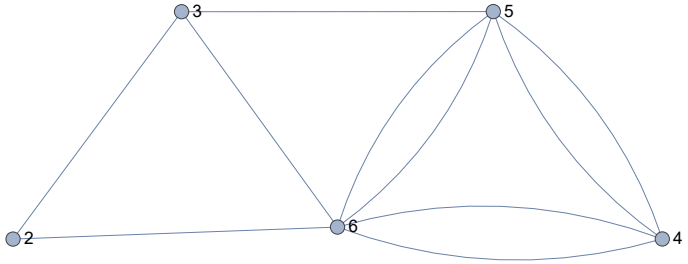



```
Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 1 ↔ 2, 1 ↔ 2}, VertexLabels → "VertexDegree"]
```

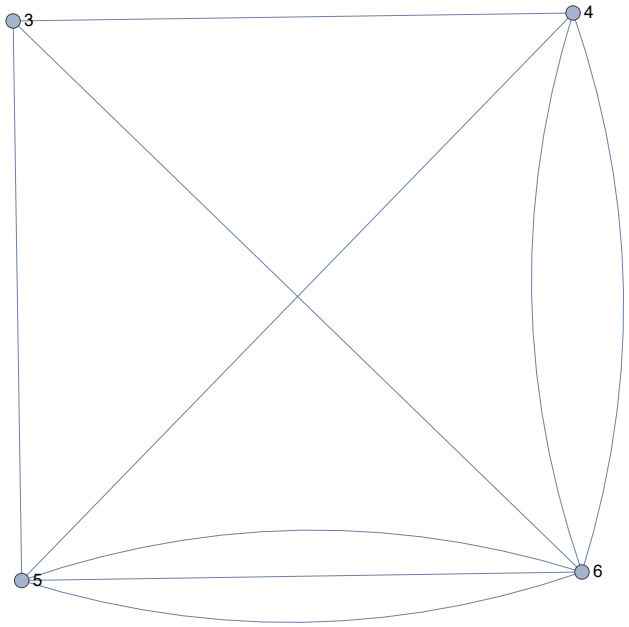

```
Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 1, 1 ↔ 2, 1 ↔ 2, 1 ↔ 3}, VertexLabels → "VertexDegree"]
```



```
Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 5 ↔ 1, 1 ↔ 2, 1 ↔ 3, 1 ↔ 4, 1 ↔ 3, 2 ↔ 3},
 VertexLabels → "VertexDegree"]
```

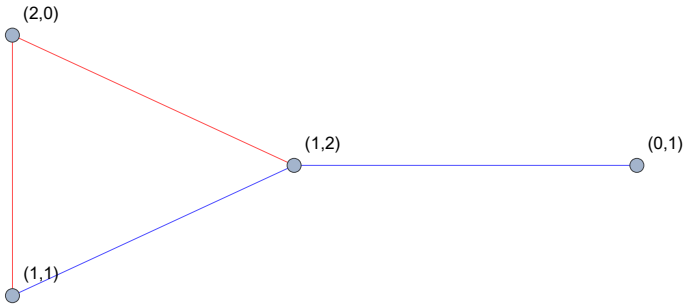



```
Graph[{1 ↔ 2, 1 ↔ 3, 1 ↔ 4, 2 ↔ 3, 2 ↔ 4, 3 ↔ 4, 1 ↔ 2, 1 ↔ 2, 1 ↔ 3}, VertexLabels → "VertexDegree"]
```



▫ 2.49

```
Graph[{
  Property[1, VertexLabels → "(1,1)"],
  Property[2, VertexLabels → "(2,0)"],
  Property[3, VertexLabels → "(1,2)"],
  Property[4, VertexLabels → "(0,1)"]}, {
  Property[1 ⟷ 2, EdgeStyle → Red],
  Property[1 ⟷ 3, EdgeStyle -> Blue],
  Property[2 ⟷ 3, EdgeStyle → Red],
  Property[3 ⟷ 4, EdgeStyle -> Blue]}]
```



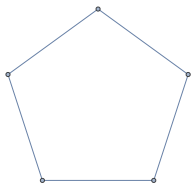
3 Isomorphic Graphs

3.1 The Definition of Isomorphism

```
ComplementGraph[G_] :=
 Graph[
  Complement[
   Sort /@ EdgeList[CompleteGraph[VertexCount[G]]],
   Sort /@ EdgeList[G]]]
```

▫ 3.1

```
g31a = CycleGraph[5, ImageSize → Tiny]
```



```
VertexCount[g31a]
EdgeCount[g31a]
```

5

5

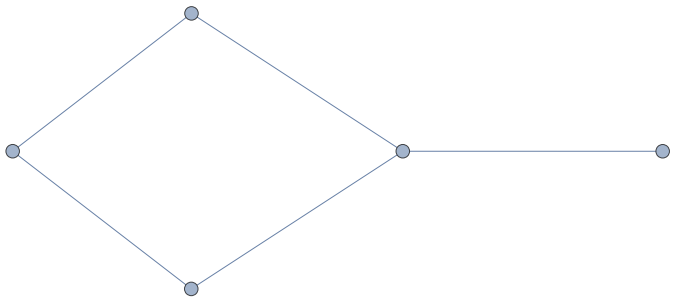
```
g31b = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 1 ↔ 3}]
```

```
VertexCount[g31b]
EdgeCount[g31b]
```

5

5

```
g31c = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 1 ↔ 4}]
```



```
VertexCount[g31c]
EdgeCount[g31c]
```

5

5

```
IsomorphicGraphQ[g31a, g31b]
IsomorphicGraphQ[g31b, g31c]
IsomorphicGraphQ[g31c, g31a]
```

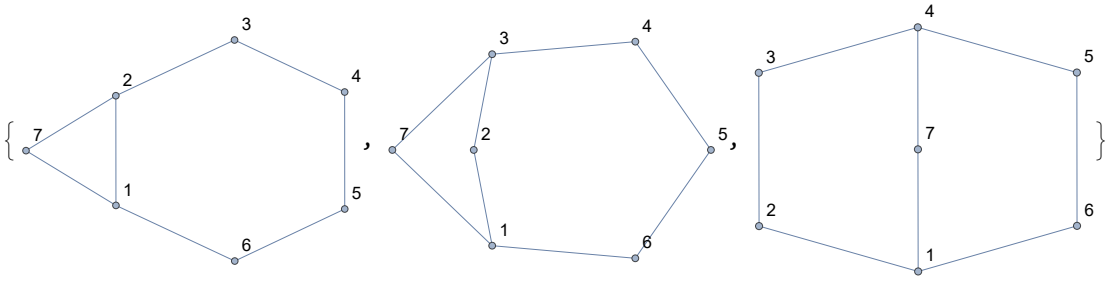
False

False

False

3.2

```
g32 = Table[Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 5 ↔ 6, 6 ↔ 1, 1 ↔ 7, 7 ↔ i},
    VertexLabels → "Name"], {i, {2, 3, 4}}]
```




```
Composition[Sort, VertexDegree] /@ g32
```

```
{{2, 2, 2, 2, 2, 3, 3}, {2, 2, 2, 2, 2, 3, 3}, {2, 2, 2, 2, 2, 3, 3}}
```

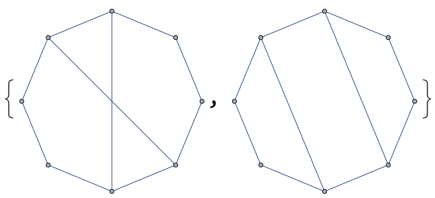
```
IsomorphicGraphQ[g32[[1]], g32[[2]]]
IsomorphicGraphQ[g32[[2]], g32[[3]]]
IsomorphicGraphQ[g32[[3]], g32[[1]]]
```

```
False

False

False
```

3.3

The graphs in (a) are not isomorphic: G_1 has 5-cycles, whereas G_2 does not; and G_2 has 4-cycles, whereas G_1 does not:

```
g33 = Table[
  GraphUnion[
   CycleGraph[8], edges,
   GraphLayout -> "CircularEmbedding", ImageSize → Tiny],
  {edges, {{1 ↔ 5, 4 ↔ 8}, {1 ↔ 4, 5 ↔ 8}}}
 ]
```



```
IsomorphicGraphQ[g33[[1]], g33[[2]]]
```

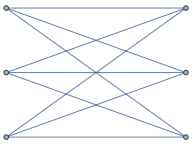
```
False
```

The graphs in (b) are isomorphic: they are both a 5-path whose ends are joined through two different vertices.

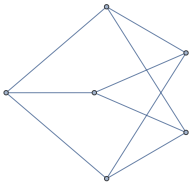
3.4

Note first that all four graphs are of order 6 and size 9 and are regular of degree 3; so we seek to distinguish them by structural properties. G_1 and G_2 are isomorphic as they are both bipartite:

```
g34G1 = CompleteGraph[{3, 3}, ImageSize → Tiny]
```



```
g34G2 = GraphUnion[CycleGraph[6, ImageSize → Tiny], {1 ⟷ 4, 2 ⟷ 5, 3 ⟷ 6}]
```



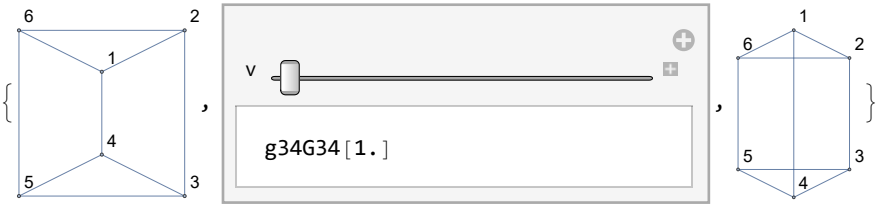
```
IsomorphicGraphQ[g34G1, g34G2]
```

```
True
```

Next, G_3 and G_4 are also isomorphic as one can be 'morphed' into the other by graphically moving the center vertices along the vertical axis.

```
g34G34[v_] := GraphUnion[
  CycleGraph[6],
  {2 ⟷ 6, 3 ⟷ 5, 1 ⟷ 4},
  VertexLabels → "Name",
  VertexCoordinates → {{0, v}, {2, 2}, {2, -2}, {0, -v}, {-2, -2}, {-2, +2}},
  ImageSize → Tiny
 ]
```

```
{g34G34[1], Manipulate[g34G34[v], {v, 1, 3}], g34G34[3]}
```

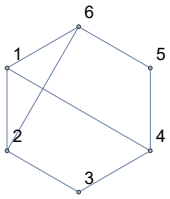


G_1/G_2 and G_3/G_4 are not mutually isomorphic: the latter contain 3-cycles, whereas the former (being bipartite) do not.

3.5

That conclusion is not valid; the similarity in vertex naming doesn't imply structural similarity. For example:

```
GraphUnion[
 CycleGraph[6],
 {2 ↔ 6, 1 ↔ 4},
 VertexLabels → "Name",
 GraphLayout -> "CircularEmbedding", ImageSize → Tiny
]
```

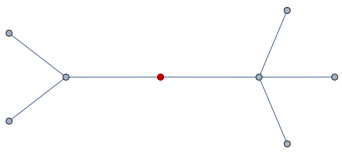


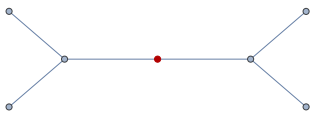
Vertex 1 is of degree 3 and is not adjacent to a vertex of degree 2; while vertex 4 is also of degree 3 but *is* adjacent to a vertex of degree 2.

3.6

Again, no: consider trivially a disconnected graph with two components G_1 and G_2 as described in the problem. That graph satisfies the requirements of both G_1 and G_2, and is obviously isomorphic with itself:

```
HighlightGraph[
 GraphUnion[
  Graph[{1ₐ ↔ 2, 1ᵦ ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5ₐ, 4 ↔ 5ᵦ, 4 ↔ 5꜀}],
  Graph[{6ₐ ↔ 7, 6ᵦ ↔ 7, 7 ↔ 8, 8 ↔ 9, 9 ↔ 10ₐ, 9 ↔ 10ᵦ}]
 ],
 {3, 8},
 ImageSize → Small
]
```





3.7

The proposed solution appeals to 'graphical' properties related to how the graph is drawn; but it is not certain that those are structurally invariant. In fact, it is clear that the two graphs are in fact isomorphic: they both consist of two 'terminal' vertices

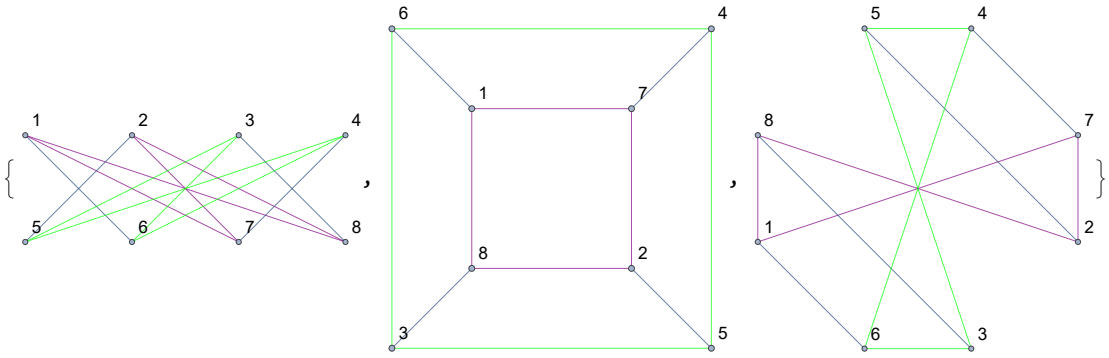
connected to each other by two separate 2-paths and a single 3-path.

3.8

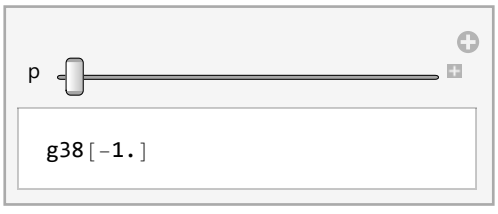
The graphs are each pairs of 4-cycles; in G_1 and G_2, the vertices in the pairs can be matched so that the cycles can be traversed in parallel: 1-6, 8-3, 2-5, 7-4. However, in G_3, the vertices in the pair of cycles cannot be matched up in this way– the cycles are oriented differently.

```
g38[p_] := HighlightGraph[
  Graph[
   Range[8],
   {1 ↔ 6, 1 ↔ 7, 1 ↔ 8, 2 ↔ 5, 2 ↔ 7, 2 ↔ 8, 3 ↔ 5, 3 ↔ 6, 3 ↔ 8, 4 ↔ 5, 4 ↔ 6, 4 ↔ 7},
   VertexCoordinates →
    Interpolation[{
        {-1, {{-3, +1}, {-1, +1}, {+1, +1}, {+3, +1}, {-3, -1}, {-1, -1}, {+1, -1}, {+3, -1}}},
        {0, {{-1, +1}, {+1, -1}, {-2, -2}, {+2, +2}, {+2, -2}, {-2, +2}, {+1, +1}, {-1, -1}}},
        {+1, {{-3, -1}, {+3, -1}, {+1, -3}, {+1, +3}, {-1, +3}, {-1, -3}, {+3, +1}, {-3, +1}}}},
      InterpolationOrder → 1][p],
   VertexLabels → "Name"
  ], {
   Style[{5 ↔ 4, 4 ↔ 6, 6 ↔ 3, 3 ↔ 5}, Green],
   Style[{1 ↔ 7, 7 ↔ 2, 2 ↔ 8, 8 ↔ 1}, Purple]
  }
 ]
```

```
{g38[-1], g38[0], g38[+1]}
```




```
Manipulate[g38[p], {p, -1, +1}]
```



3.9

They are not isomorphic: for example, the four vertices of degree 4 form a path in G_1 and a cycle in G_2.

3.10

No: the complement of a disconnected graph is always connected and is thus not isomorphic. Two vertices in *different components* are connected in the complement; two vertices in the *same component* are both connected to every other vertex in another component in the complement and are therefore also connected in the complement.

3.11

(From the solutions) Let $a = |U| = |V|$. Since there are a vertices v with $\deg_G v \geq \frac{1}{2} n$, in the complement there are also a vertices with $\deg_{\overline{G}} v \leq (n-1) - \frac{1}{2} n = \frac{1}{2} n - 1$. Since G is isomorphic with $\overline{G}$, in the complement there are also a vertices with $\deg_{\overline{G}} v \leq \frac{1}{2} n$. Therefore there are no vertices v in the complement, or in G, with $\deg v = \frac{1}{2} n$.

3.12

Since G and H themselves are isomorphic with their respective complements, it suffices to show that the whole of $\overline{G \cup H}$ is isomorphic with $G \cup H$; that is, $\forall\, g \in G, h \in H$ there is an edge between $\phi\,\overline{g}, \phi\,\overline{h}$ iff there is one between $\overline{g} \in \overline{G}, \overline{h} \in \overline{H}$.
Any vertex $\overline{h} \in \overline{H}$ with $\deg \overline{h} < \frac{1}{2} n$ has since n is even $\deg \overline{h} \leq \frac{1}{2} n - 1$ and $\deg h \geq (n-1) - \left(\frac{1}{2} n - 1\right) = \frac{1}{2} n$ not less than $\frac{1}{2} n$ in H and was therefore not connected to any vertex in G, and thus connected to every vertex in $\overline{G}$. Since it is isomorphic to H, the degree of $\phi\,\overline{H} \in H$ is also less than $\frac{1}{2} n$ and is thus connected to every vertex in G. Similarly for vertices $\overline{h} \in \overline{H}$ of degree not less than $\frac{1}{2} n$.

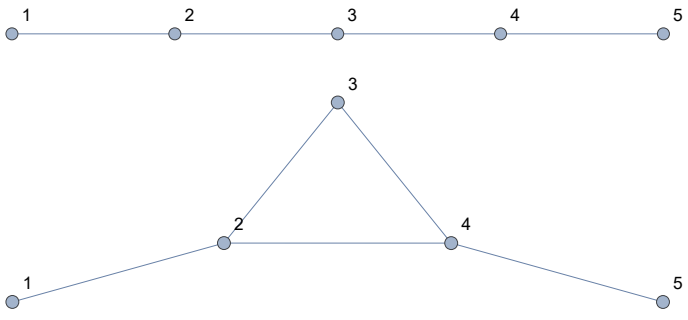
3.13

First, note that for any two vertices the distance between them is 1 iff there is an edge between them. Then for every $u, v \in V(G)$, iff there is an edge between u and v, then $d_G(u, v) = 1 \Rightarrow d_H(\phi u, \phi v) = 1$ so there is an edge between ϕu and ϕv. This shows that ϕ is an isomorphism.

3.14

No: for example, the following two graphs:

```
g314a = PathGraph[Range[5], VertexLabels → "Name"]
g314b = GraphUnion[PathGraph[Range[5]], Graph[{2 ⟷ 4}], VertexLabels → "Name"]
```



```
Table[GraphDistance[#, i, i + 1], {i, VertexCount[#] - 1}] & /@ {g314a, g314b}
```

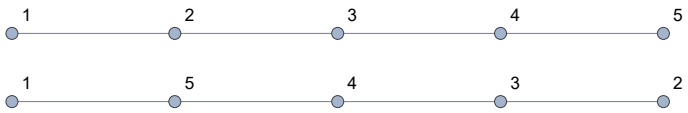
```
{{1, 1, 1, 1}, {1, 1, 1, 1}}
```

Under the identity isomorphism the distance between consecutively-numbered vertices is the same (i.e., always one) although the graphs are clearly not isomorphic.

3.15

No: while ϕ itself may not be an isomorphism, there can exist ψ that are. For example, for the following graphs *G* and *H*:

```
PathGraph[Range[5], VertexLabels → "Name"]
PathGraph[{1, 5, 4, 3, 2}, VertexLabels → "Name"]
```



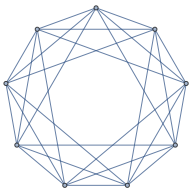
under the identity $\phi = i$, $d_G(1, 2) = 1$ and $d_H(\phi 1, \phi 2) = d_H(1, 2) = 4$ while under a more carefully chosen map ψ the graphs can be shown to be isomorphic.

3.2 Isomorphism as a Relation

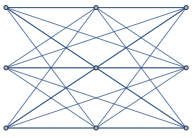
3.16

Not sure how to find *all* of them, but here are two:

```
g316a = HararyGraph[6, 9, ImageSize → Tiny]
```



```
g316b = CompleteGraph[{3, 3, 3}, ImageSize → Tiny]
```



```
VertexDegree /@ {g316a, g316b}
```

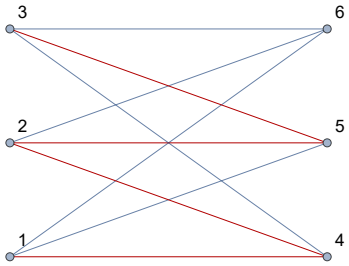
```
{{6, 6, 6, 6, 6, 6, 6, 6, 6}, {6, 6, 6, 6, 6, 6, 6, 6, 6}}
```

```
IsomorphicGraphQ[g316a, g316b]
```

```
False
```

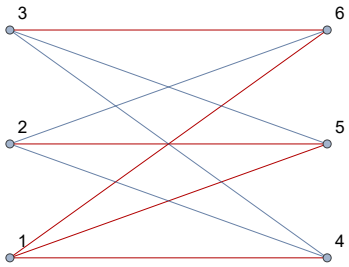
3.17

```
HighlightGraph[CompleteGraph[{3, 3}, VertexLabels → "Name"],
 {1 ↔ 4, 4 ↔ 2, 2 ↔ 5, 5 ↔ 3}, ImageSize → Small]
```



No for (b), since a bipartite graph contains no odd cycles.

```
HighlightGraph[CompleteGraph[{3, 3}, VertexLabels → "Name"],
 {1 ↔ 4, 1 ↔ 5, 1 ↔ 6, 5 ↔ 2, 6 ↔ 3}, ImageSize → Small]
```



3.18

No: by transitivity, if one component is isomorphic to the other two, then the other two are also.

3.19

More clearly, the question is: show that the number of graphs that are isomorphic to an odd number of graphs is itself even (and not: show that there exists an isomorphism class with an even number of graphs that are isomorphic to an odd number of graphs).

Intuitively, this is trivial: there must be an even number of 'end instances' in a set of binary relationships, so if each graph participates an odd number of times as an 'end instance' then there must be an even number of graphs. More precisely, using the approach in the solution, express the graph isomorphisms as a graph itself. Since there is an even number of odd vertices, there is an even number of graphs isomorphic to an odd number of graphs.

3.3 Graphs and Groups

3.20

The permutation that swaps v_3 and v_4 and leaves all other vertices fixed preserves degrees but is not an automorphism.

3.21

Any permutation on the three vertices is an automorphism, so Aut $K_3 = S_3$.

```
SameQ[
 GroupMultiplicationTable[GraphAutomorphismGroup[CompleteGraph[3]]],
 GroupMultiplicationTable[SymmetricGroup[3]]]
```

```
True
```

3.22

Any permutation of the three leaf vertices is an automorphism, and these are the only ones; so again, Aut $K_{1,3} = S_3$.

```
SameQ[
 GroupMultiplicationTable[GraphAutomorphismGroup[CompleteGraph[{3, 1}]]],
 GroupMultiplicationTable[SymmetricGroup[3]]]
```

```
True
```

3.23

The only permutation is a reflection around the middle of the path; so Aut $K_{n\geq 2} = S_2$.

```
SameQ[
 GroupMultiplicationTable[GraphAutomorphismGroup[PathGraph[Range[9]]]],
 GroupMultiplicationTable[SymmetricGroup[2]]]
```

```
True
```

3.24

These are the permutations of an n-sided regular polygon; so Aut $C_4 = D_4$.

```
SameQ[
 GroupMultiplicationTable[GraphAutomorphismGroup[CycleGraph[4]]],
 GroupMultiplicationTable[DihedralGroup[4]]]
```

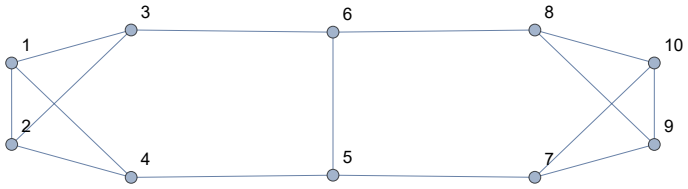
```
True
```

3.25

The orbits of H_1 are {1}, {2}, {3}, {4, 5}, {6}, {7}, {8, 9}, and Aut $H_1 = S_2 \times S_2$. The orbits of H_2 are {1}, {2}, {3, 4}, {5}, {6, 9}, {7, 8}, {10}, and Aut $H_2 = S_2 \times S_2$.

3.26

```
g326 = Graph[{u ↔ u', u ↔ v', u' ↔ v, u ↔ v, u' ↔ v', v ↔ w, v' ↔ w', w ↔ w',
    w ↔ x, w' ↔ x', x ↔ y', x' ↔ y, x ↔ y, x' ↔ y', y ↔ y'}, VertexLabels → "Index"]
```



The automorphism group has four generating permutations: the one that swaps *u* and *u*', the one that swaps *y* and *y*', the reflection of all primed and unprimed vertices, and the reflection along the *w* – *w*' axis. This gives the following orbits:

```
VertexList[g326][[#]] & /@ GroupOrbits[GraphAutomorphismGroup[g326]]
```

```
{{u, u′, y′, y}, {v′, v, x, x′}, {w, w′}}
```

The group can be characterized by analysis. Note first that the permutation of just *w* and *w*' can be constructed from a product of the reflection of primed and unprimed vertices with the permutations of *u* with *u*' and of *y* with *y*'. Therefore we can construct a different generating set of the permutations of *u* with *u*', of *y* with *y*', of *w* with *w*', and of the reflection along the *w* – *w*' axis. The permutation of *w* with *w*' is unaffected by any of the other three, so the automorphism group as a whole is isomorphic to the direct product of C_2 with the subgroup generated by the three remaining permutations:

```
g326s = PermutationGroup[{Cycles[{{9, 10}}], Cycles[{{1, 2}}], Cycles[{{1, 10}, {2, 9}}]}]
```

```
SameQ[
 PermutationProduct[GroupGenerators[g326s][[2]]⊙GroupGenerators[g326s][[1]]],
 PermutationProduct[GroupGenerators[g326s][[1]]⊙GroupGenerators[g326s][[2]]]]
```

```
True
```

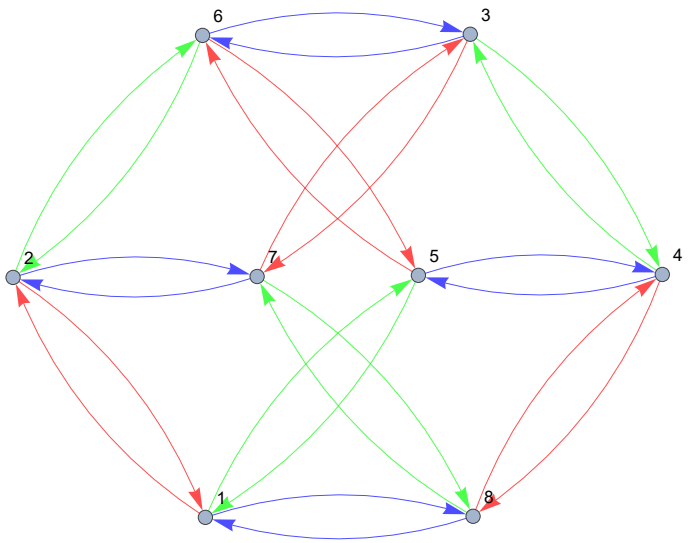
```
GroupElements[g326s]
```

```
{Cycles[{}], Cycles[{{9, 10}}], Cycles[{{1, 2}}],
 Cycles[{{1, 2}, {9, 10}}], Cycles[{{1, 9}, {2, 10}}],
 Cycles[{{1, 9, 2, 10}}], Cycles[{{1, 10, 2, 9}}], Cycles[{{1, 10}, {2, 9}}]}
```

```
GroupGenerators[g326s]
```

```
{Cycles[{{9, 10}}], Cycles[{{1, 2}}], Cycles[{{1, 10}, {2, 9}}]}
```

```
CayleyGraph[g326s, VertexLabels → "Index"]
```




```
FiniteGroupData[{"DirectProduct", {{"DihedralGroup", 4}, {"SymmetricGroup", 2}}},
 "PermutationGroupRepresentation"]
```

```
PermutationGroup[{Cycles[{{5, 6}}], Cycles[{{1, 2, 3, 4}}], Cycles[{{1, 4}, {2, 3}}]}]
```

3.27

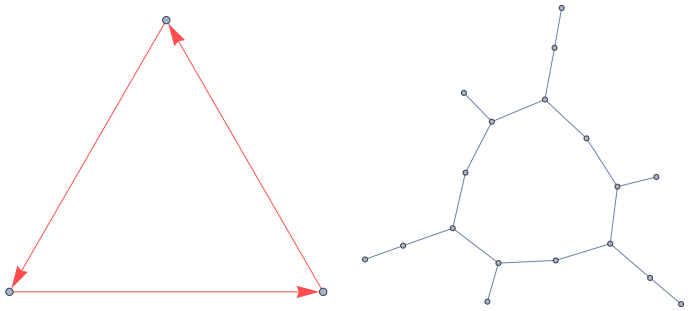
$\phi \in \text{Aut}\, G$ iff $\phi : V_G \to V_G : u\, v \in E_G \Longleftrightarrow \phi u\, \phi v \in E_G$, which implies that $u\, v \notin E_G \Longleftrightarrow \phi u\, \phi v \notin E_G$ or $u\, v \in E_{\overline{G}} \Longleftrightarrow \phi u\, \phi v \in E_{\overline{G}}$ and $\phi \in \text{Aut}\, \overline{G}$.

3.28

```
AutomorphismGraph[g_] := Graph[
  Flatten[
   Block[
       (* left and right edge ordinal *)
       {lv = #[[1]], rv = #[[2]]},
       Block[
        (* ordinal of generator that the edge represents *)
        {gp = First[FirstPosition[
            GroupGenerators[g],
            (* permutation that the edge represents *)
            PermutationProduct[
             InversePermutation[GroupElements[g][[lv]]],
             GroupElements[g][[rv]]]]]},
        (* edge complex replacing the simple Cayley graph edge *)
        Flatten[{
          lv ⟼ L_{lv,rv,0},
          L_{lv,rv,0} ⟼ R_{lv,rv,0},
          R_{lv,rv,0} ⟼ rv,
          Table[L_{lv,rv,i-1} ⟼ L_{lv,rv,i}, {i, 2*gp - 1}],
          Table[R_{lv,rv,i-1} ⟼ R_{lv,rv,i}, {i, 2*gp}]
         }]]] & /@
    (* over all the edges of the group's Cayley graph *)
    EdgeList[CayleyGraph[g]]]]
```

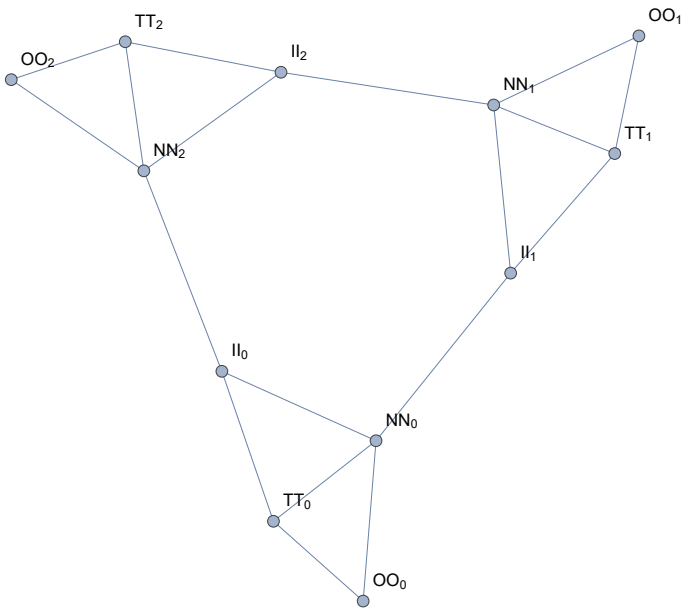
```
CayleyAutomorphismRow[g_] :=
 GraphicsRow[
  #[g] & /@
   {CayleyGraph, AutomorphismGraph}]
```

```
CayleyAutomorphismRow[PermutationGroup[{Cycles[{{1, 2, 3}}]}]]
```



A hand-crafted graph of order 12 with the same automorphism group:

```
g328 = Graph[
  Flatten[
   Block[
       (* left and right edge ordinal *)
       {lv = #, rv = Mod[# + 1, 3], nv = Mod[# + 2, 3]},
       (* edge complex replacing the simple Cayley graph edge *)
       Flatten[{
         II_lv ⟷ NN_lv, NN_lv ⟷ II_rv,
         II_lv ⟷ TT_lv, NN_lv ⟷ TT_lv,
         TT_lv ⟷ OO_lv, OO_lv ⟷ NN_lv
        }] & /@
     Range[0, 2]], VertexLabels -> "Name"]
```



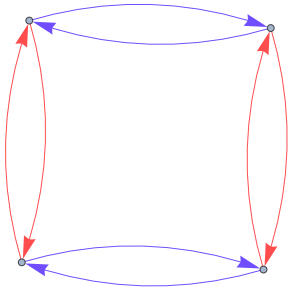

```
GraphAutomorphismGroup[g328]
VertexCount[g328]
```

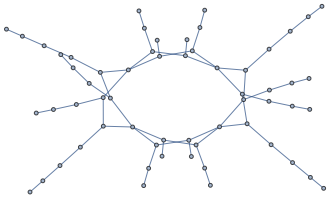
```
PermutationGroup[{Cycles[{{1, 3, 7}, {2, 6, 10}, {4, 8, 11}, {5, 9, 12}}]}]
```

```
12
```

3.29

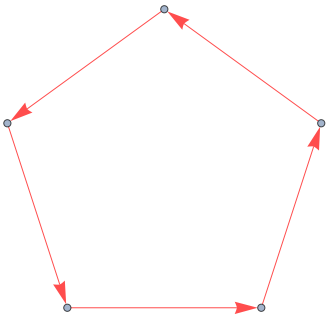
```
CayleyAutomorphismRow[
 PermutationGroup[{
   Cycles[{{1, 2}}],
   Cycles[{{3, 4}}]}]]
```

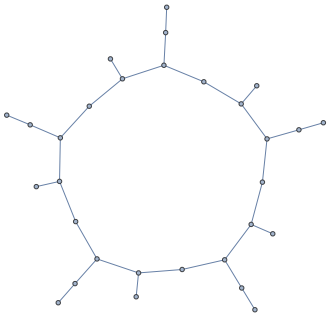




3.30

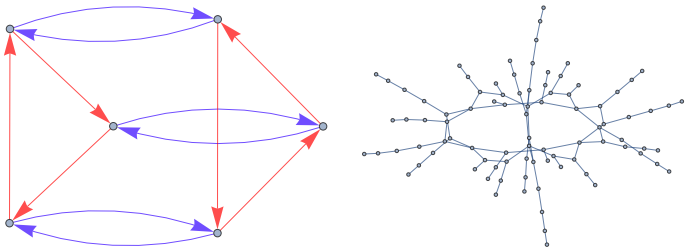
```
CayleyAutomorphismRow[
 PermutationGroup[{
   Cycles[{{1, 2, 3, 4, 5}}]}]]
```





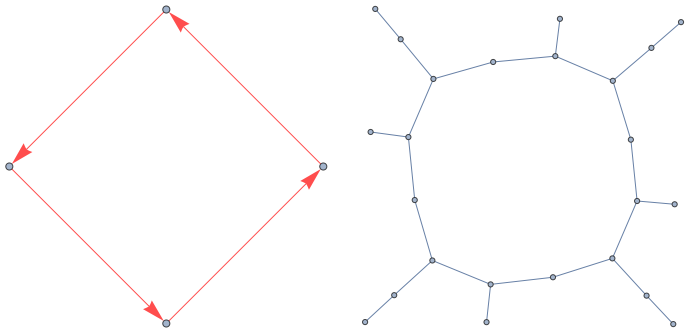
3.31

```
CayleyAutomorphismRow[
 PermutationGroup[{
   Cycles[{{1, 2, 3}}],
   Cycles[{{1, 2}}]}]]
```

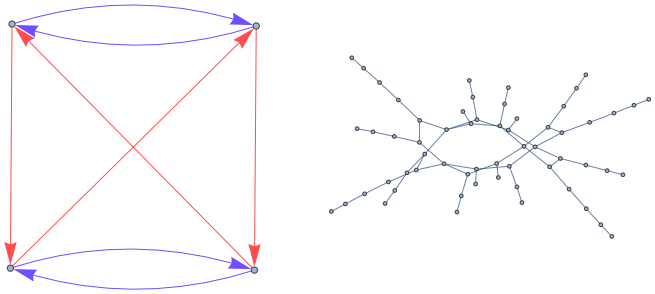


3.32

```
CayleyAutomorphismRow[
 PermutationGroup[{
   Cycles[{{1, 2, 3, 4}}]}]]
```



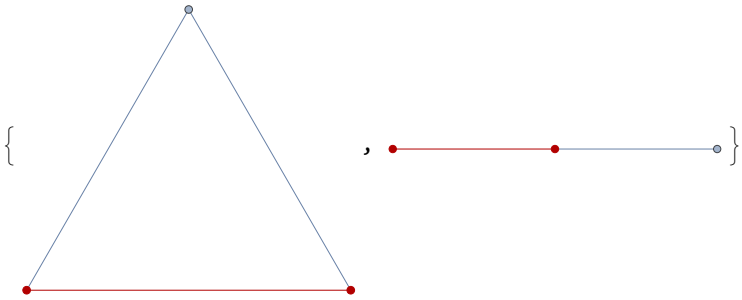
```
CayleyAutomorphismRow[
 PermutationGroup[{
   Cycles[{{1, 2, 3, 4}}],
   Cycles[{{1, 3}, {2, 4}}]}]]
```



3.4 Reconstruction and Solvability

3.33

```
HighlightGraph[
   #,
   Subgraph[#, {1, 2}]] & /@
 {CompleteGraph[3], PathGraph[Range[3]]}
```



3.34

a. 7: there are 7 subgraphs, each having order 6.

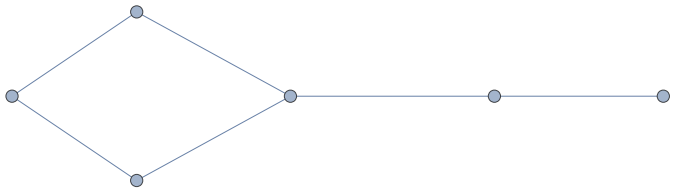
b. $m = \frac{\sum_i m_i}{n-2} = \frac{40}{5} = 8$.

c. $(_i m - m_i) = (2, 2, 3, 3, 2, 2, 2)$.

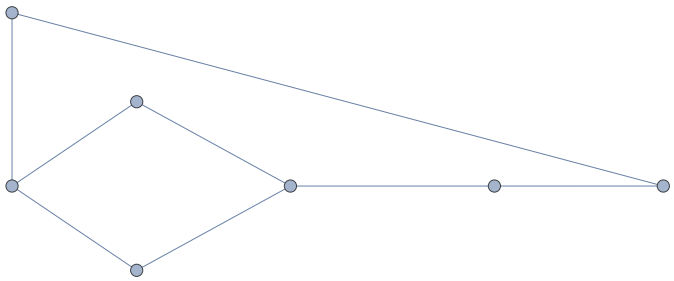
d. Yes; at least two of the subgraphs are connected.

e. By 5/6, G should contain C_6 as a subgraph. So add a vertex to G_1 connecting the top to the bottom. This graph has the requisite order, size, and degree sequence:

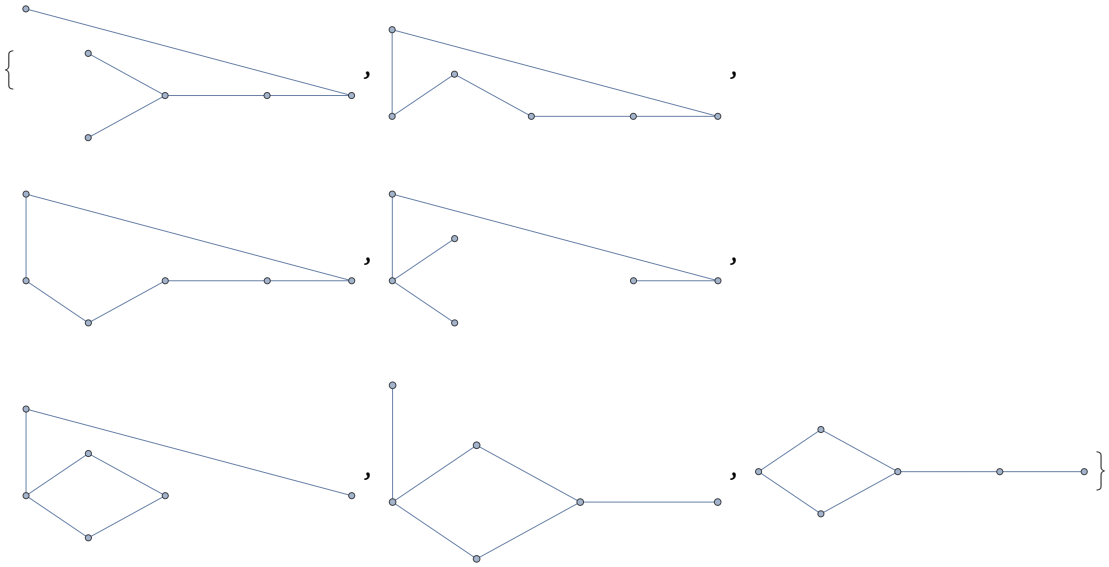
```
g344G1 = Graph[{1 ↔ 2, 1 ↔ 3, 2 ↔ 4, 3 ↔ 4, 4 ↔ 5, 5 ↔ 6}]
```



```
g344G = Graph[
  Join[EdgeList[g344G1], {6 ↔ 7, 7 ↔ 1}],
  VertexCoordinates → Append[
    PropertyValue[{g344G1, #}, VertexCoordinates] & /@ Range[VertexCount[g344G1]],
    {0, 1.5}]]
```



```
Table[Graph[VertexDelete[g344G, i]], {i, VertexCount[g344G]}]
```



3.35

a. 7; 7 subgraphs, each of order 6.

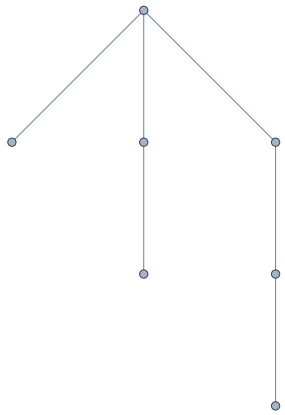
b. $m = \frac{\sum_i m_i}{n-2} = \frac{30}{5} = 6$.

c. Connected; at least two of the subgraphs are connected.

d. $(_i m - m_i) = (1, 3, 1, 1, 2, 2, 2)$.

e. Relative to G_1 there are one vertex and one edge missing. Its degree sequence is $(1, 2, 2, 2, 1)$, so the edge must be added to the middle. Without loss of generality, there are two candidate locations. By G_3, the vertex with degree 3 must be connected to P_3, P_1, P_1. By G_4, it must be connected to P_2, P_2, P_1. By G_5, it must be connected to P_1, P_1, P_2. Therefore, in G, that vertex must be connected to P_1, P_2, P_3:

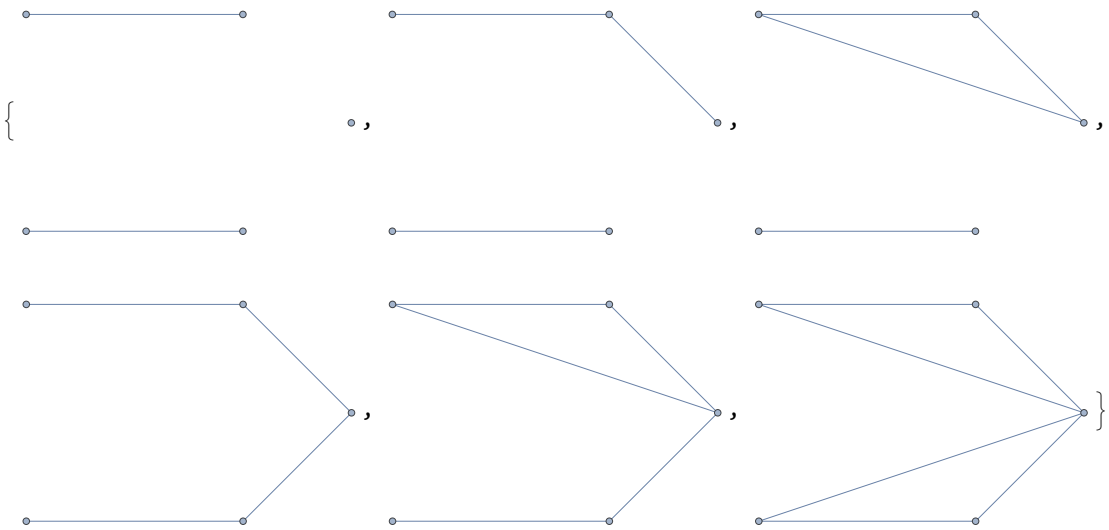
```
Graph[{1 ⟷ 2, 1 ⟷ 3, 3 ⟷ 4, 1 ⟷ 5, 5 ⟷ 6, 6 ⟷ 7}, ImageSize → Small]
```



3.36

By #2, the order of G is 5. There are six possibilities (up to isomorphism) for it:

```
EdgeAdd[
   Graph[
    Range[5],
    {1 ↔ 2, 3 ↔ 4},
    VertexCoordinates → {{-1, +1}, {+1, +1}, {-1, -1}, {+1, -1}, {+2, 0}}],
   #] & /@
 {{}, {2 ↔ 5}, {2 ↔ 5, 1 ↔ 5}, {2 ↔ 5, 4 ↔ 5}, {2 ↔ 5, 4 ↔ 5, 1 ↔ 5}, {2 ↔ 5, 4 ↔ 5, 1 ↔ 5, 3 ↔ 5}}
```



Only for the first three options does every G_i have connected components. For each of the last three options, there exist G_i with unconnected components.

3.37

The criteria can be restated as follows:G is connected, $\forall\, i : \Delta\, G_i \leq 2$, $\nexists\, i : \Delta\, G_i = 1$, $\exists\, i : \Delta\, G_i = 2$. This implies that $\delta\, G > 0$ and $\Delta\, G < 3$, which in turn implies that G is either a path P_n or a cycle C_n. Also, $n \geq 4$ or the minimum degree of a subgraph would be one.

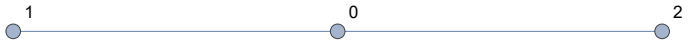
3.38

Since $\mid E_{G_i} \mid \neq \mid E_{G_j} \mid$ for $i \neq j$, the degree of each vertex of G must be different. Also, since $\mid V_{G_i} \mid = 5$ for some i, it is true for all i, and $\mid V_G \mid = 6$. Thus $\Delta G = 5$ and the degree sequence of G must be $(0, 1, 2, 3, 4, 5)$. However, this means that G would have an odd number of vertices of odd degree, which is impossible. Thus there are no solutions.

3.39

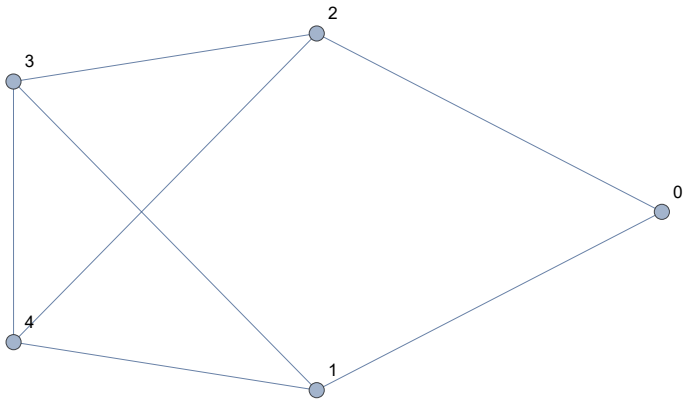
By the first condition, there is a subgraph isomorphic to K_{n-1} with a single edge removed. WLOG call this subgraph G_0. By the second condition there is a subgraph containing only odd vertices; there must be an even number of them. Therefore $n = \mid V_G \mid$ is odd and $\mid V_{G_0} \mid$ is even. WLOG call the vertices of the removed edge of G_0, v_1 and v_2. v_0 must be adjacent to both, otherwise G_1 or G_2 would also contain exactly two nonadjacent vertices v_0, v_2 and v_0, v_1, respectively. It is now clear that there is a single case for $n = 3$:

```
GraphUnion[
 Graph[Range[2], DeleteCases[EdgeList[CompleteGraph[2]], 1 ⟷ 2]],
 Graph[Range[0, 2], {1 ⟷ 0, 2 ⟷ 0}],
 VertexLabels → "Name"]
```



This graph is also a solution. Consider now the cases for $n \geq 5$. The only question is whether v_0 is adjacent to $v_{i\geq 3}$. v_0 cannot be adjacent to all $v_{i\geq 3}$ or G_3 would also still have one pair of nonadjacent vertices v_1, v_2. Similarly, v_0 cannot be adjacent to all but one $v_{i\geq 3}$ (WLOG, say v_3) because again G_3 would have the same single pair of nonadjacent vertices. So WLOG v_0 is not adjacent to two vertices v_3 and v_4. So again for $n = 5$ there is a single case:

```
GraphUnion[
 Graph[Range[4], DeleteCases[EdgeList[CompleteGraph[4]], 1 ⟷ 2]],
 Graph[Range[0, 4], {1 ⟷ 0, 2 ⟷ 0}],
 VertexLabels → "Name"]
```



Now it can be verified that none of the subgraphs contain only odd vertices; so this case is not a solution. For $n \geq 7$, the degrees of $v_{1,2,3,4}$ remain odd so none of $G_{i\geq 1}$ have all odd vertices; so only G_0 could have only odd vertices, and the second condition is not satisfied.

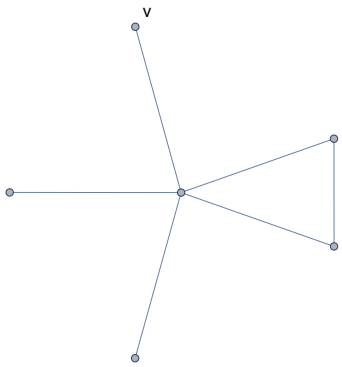
3.40

Every graph of order 3 or less (K_0, K_1, K_2, $\overline{K_2}$, K_3, $\overline{K_3}$, $K_2 \cup K_1$) is a solution. For orders greater than 3, K_n and $\overline{K_n}$ are certainly solutions. Consider any other graph G of order greater than 3. If it is itself regular, then since it is not complete, there is a vertex v that is not adjacent to every other vertex, and $G - v$ is irregular; so these are not solutions. Consider then irregular G: this will have vertices with minimum degree δ and maximum degree Δ. Since $\forall\, v \in G : G - v$ is regular, it must be that $\Delta = \delta + 1$ and G can be considered as partitioned into vertices v_δ and v_Δ. If there is just one v_δ, then $G - v_\Delta$ must be regular of degree $\Delta - 1$; but then the v_Δ must form a complete subgraph that is disconnected from v_δ, implying that $\delta = 0$ and $\Delta = 1$; but the order of G is greater than 3, so this is impossible. So there must be more than one v_δ, in which case $G - v_\delta$ must have the effect of reducing the degree of all the v_Δ by one. Then each v_δ is adjacent to each v_Δ and $|\, v_\delta \,| = \Delta$ and $|\, v_\Delta \,| = \delta$; this then comprises all the edges in the graph (G is bipartite) and $\delta > 1$. But then $G - v_\Delta$ reduces the degree of the v_δ to $\Delta - 2$ while leaving at least one v_Δ, which is then not regular.
So the solutions are the graphs of order 3 or less, and K_n and $\overline{K_n}$.

3.41

A solution should have one additional vertex *v*; by Theorem 14 the solution is connected. Consider the two vertices of degree 1; *v* cannot be connected to either, because then one of the subgraphs would have contained P_2. Then, consider C_3 at the center. *v* also cannot be connected to either vertex of degree 2, because one of the subgraphs would then have at least two vertices of degree 3 or not contain C_3. *v* must therefore be connected to the vertex of degree 3:

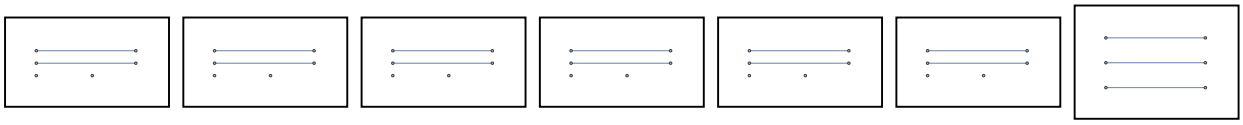
```
Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 1, 3 ↔ 4, 3 ↔ 5, 3 ↔ 6}, ImageSize → Small, VertexLabels → {6 → "v"}]
```



3.42

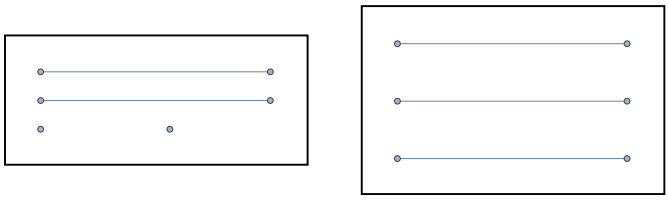
For a), the trivial answer is to supply all the subgraphs:

```
GraphicsRow[
 Table[
  Framed[
   VertexDelete[
    Graph[Range[7], {1 ↔ 2, 3 ↔ 4, 5 ↔ 6}, ImagePadding → 10], i]],
  {i, 7}],
 ImageSize → Full]
```



For b), consider just one copy of the nonisomorphic subgraphs:

```
GraphicsRow[
 Table[
  Framed[
   VertexDelete[
    Graph[Range[7], {1 ⟷ 2, 3 ⟷ 4, 5 ⟷ 6}, ImagePadding → 10], i]],
  {i, {1, 7}}]]
```



Consider the second subgraph. Since a solution must contain a single additional vertex v, suppose that it was connected to at least one of the K_2; then the other solution would be either $3\,K_2$ or contain at least P_3. Therefore v is not connected, and the only solution is $3\,K_2 + K_1$.

For c), consider the two constraints that 1) G must contain $3\,K_2$ and 2) G must not contain a path longer than P_3. Then the additional vertex may be either connected to exactly one or none of the K_2, leaving the only two solutions as $3\,K_2 + K_1$ or $P_3 + 2\,K_2$:

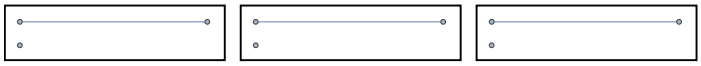
3.43

```
GraphicsRow[
 Table[
  Framed[
   Graph[{1 ⟷ 2}]], {i, 2}]]
```

The solution must be connected by Theorem 14. Therefore the only two solutions are K_3 and P_3.

3.44

```
GraphicsRow[
 Table[
  Framed[
   Graph[Range[3], {1 ⟷ 2}]],
  {i, 3}]]
```



Since the solution must have order 4, its size would be $\frac{\sum m_i}{n-2} = \frac{3}{4-2} = \frac{3}{2}$ which is impossible. However, for any two subgraphs there is a solution $2\,K_2$.

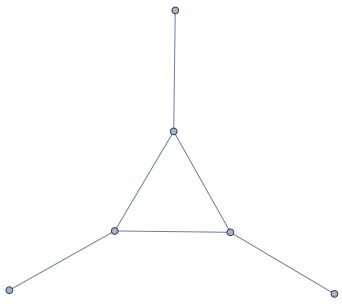
4 Trees

4.1 Bridges

4.1

Because of 4), the graph must contain K_3. Then, a solution is:

```
Graph[{1 ⟷ 2, 2 ⟷ 3, 3 ⟷ 1, 1 ⟷ 11, 2 ⟷ 22, 3 ⟷ 33}, ImageSize → Small]
```



4.2

By counterexample: suppose G with all even degrees had a bridge $u\,v$. Then $G - u\,v$ has two components, each with exactly one vertex of odd degree and any other vertices of even degree. But a component with an odd number of odd vertices is impossible.

4.3

If there were another u—v path other than $u\,v$, then both paths joined would form a cycle.

4.4

First, assume that e_1, e_2 are bridges. Then $G - e_1$ has two components; e_2 is still a bridge in the component G' that contains it, so $G' - e_2$ also has two components. Therefore $G - e_1 - e_2$ has three components.
Next, assume that $G - e_1 - e_2$ has three components. Since G is connected, e_1 must join two of these components, and e_2 must join the other two. Therefore, e_1, e_2 are bridges.

4.5

a) is proved in Theorem 4.4; b) in Corollary 4.6.

4.6

Each edge of G lies on a cycle. This must be the only cycle in G, for if there was another then $\exists\, e \in G : G - e$ has edges on that other cycle, which are then not bridges.

4.2 Trees

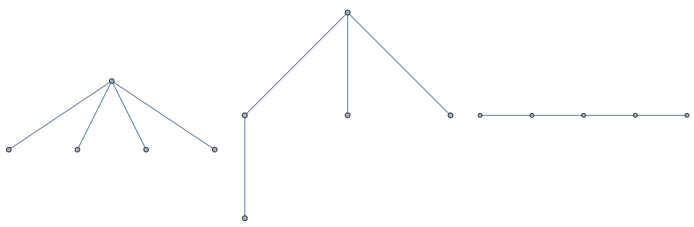
4.7

```
Graphs[n_, p_] :=
 DeleteDuplicates[ (* remove isomorphic duplicates *)
  Select[ (* select per predicate *)
   Graph[Range[n], #] & /@ (* generate graphs *)
    (* from all possible combinations of edges *)
    Subsets[EdgeList[CompleteGraph[n]]],
   p],
  IsomorphicGraphQ]
```

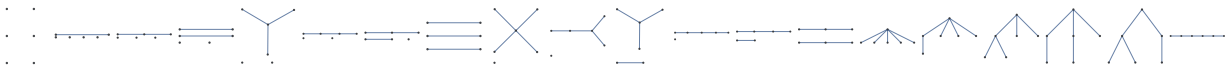
```
Trees[n_] := Graphs[n, ConnectedGraphQ[#] && AcyclicGraphQ[#] &]
```

```
Forests[n_] := Graphs[n, AcyclicGraphQ]
```

```
GraphicsRow[Trees[5], ImageSize → Full]
```



```
GraphicsRow[Forests[6], ImageSize → Full]
```



```
Length[Forests[6]]
```

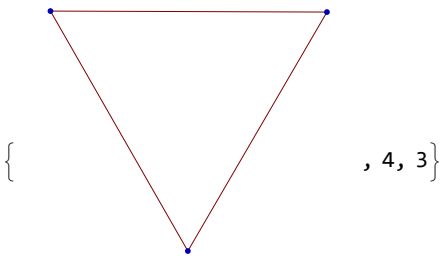
```
20
```

4.8

If a graph of order m has vertices of degree at least 2, then its size is at least $2m$. Obviously the graph is connected; suppose it was acyclic: then the graph would be a tree. But the order of a tree is $m - 1$; so such a graph cannot be acyclic.

4.9

```
#[Graph[Range[4], {1 ⟷ 2, 2 ⟷ 3, 3 ⟷ 1}]] & /@ {GraphPlot, VertexCount, EdgeCount}
```



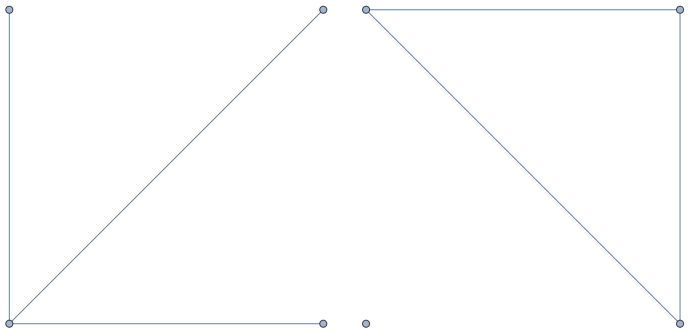
```
{ , 4, 3}
```

4.10

a) Any forest with more than one component is not connected and thus not a tree; but every edge will be a bridge.

b) Consider:

```
GraphicsRow[
 Module[{edges = {1 ⟷ 2, 1 ⟷ 3, 1 ⟷ 4}},
  Graph[
    Range[4], #,
    VertexCoordinates → {1 → {-1, -1}, 2 → {+1, -1}, 3 → {+1, +1}, 4 → {-1, +1}}] & /@
   {
    edges,
    Complement[EdgeList[CompleteGraph[4]], edges]}]]
```



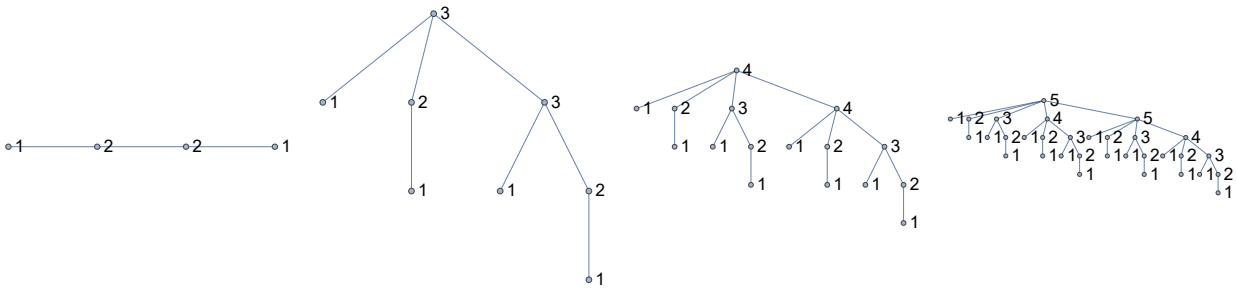
c) The tree with the end-vertices removed will therefore have exactly vertices left. They must somehow be connected to one another; else, they would have been connected to end-vertices only and there would have been multiple components. The only noncyclical way for three vertices to be connected is as P_3; which means the graph was indeed a caterpillar.

4.11

```
g411Internal[n_] := (* generate an 'arm' with some degree;
 return its 'joint' as the first element *)
 If[
  n == 1, (* terminate recursion *)
  {Unique[v]}, (* just the joint *)
  Flatten[
   Module[
    {v}, (* create the joint vertex *)
    Prepend[
     MapAt[
       UndirectedEdge[v, #] &,
       (* replace the joint of the sub-arm with an edge from it to our joint *)
       g411Internal[#],
       1] &
      /@ Range[n - 1], (* create sub-arms of all different degrees *)
     v]]]]
```

```
g411[n_] := Module[
  {g411I,
   left = g411Internal[n],
   right = g411Internal[n]},
  (* join the joints of the two sub-arms *)
  Join[Rest[left], Rest[right], {First[left] ⟷ First[right]}]
 ]
```

```
GraphicsRow[
 Graph[g411[#], VertexLabels → "VertexDegree"] &
  /@ Range[2, 5], (* would need a special case for degree 1 *)
 ImageSize → Full]
```



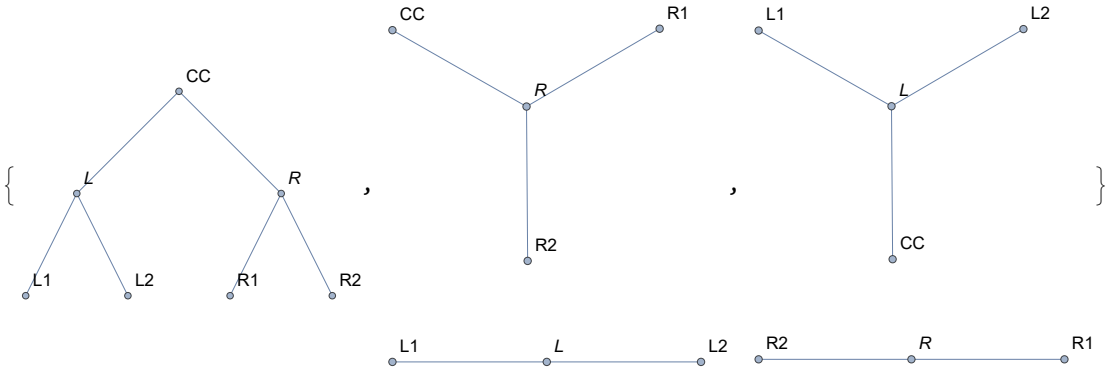
4.12

a) All the graphs of the preceding exercise.

b) Suppose that e_1 such that $G - e_1$ has two isomorphic components; certainly each of these has the same size k. e_2 lies in one of these components, say G', and therefore the size of the component of $G - e_2$ that lies in G' has size less than k; the size of the other component of $G - e_2$ must have size greater than k, which means the components of $G - e_2$ cannot possibly be isomorphic.

c) Consider:

```
Module[
 {g = Graph[{L1 ⟷ L, L2 ⟷ L, L ⟷ CC, CC ⟷ R, R ⟷ R1, R ⟷ R2}, VertexLabels -> "Name"]},
 {
  Graph[g],
  EdgeDelete[g, L ⟷ CC],
  EdgeDelete[g, CC ⟷ R]}]
```



4.13

```
Solve[15 * 1 + 1 * 6 + n * 5 + (21 - 15 - 1 - n) * 3 == 2 * (21 - 1), n]
```

{{n → 2}}

4.14

```
Solve[25 * 1 + 2 * 2 + 3 * 4 + 1 * 5 + 2 * 6 + 2 * d == 2 * (35 - 1), d]
```

{{d → 5}}

4.15

```
Solve[1/2 n * (2 + 3 + 4 + 5) == 50, n]
```

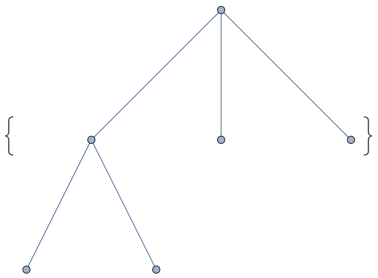
$$\left\{\left\{n \to \frac{50}{7}\right\}\right\}$$

4.16

```
GraphsWithSize[n_, m_, p_] :=
 DeleteDuplicates[ (* remove isomorphic duplicates *)
  Select[ (* select per predicate *)
   Graph[Range[n], #] & /@ (* generate graphs *)
    (* from all possible combinations of edges *)
    Subsets[EdgeList[CompleteGraph[n]], {m}],
   p],
  IsomorphicGraphQ]
```

```
TreesWithDegreeSequence[degrees_List] :=
 GraphsWithSize[
  Length[degrees],
  1/2 Total[degrees],
  AcyclicGraphQ[#] && Sort[VertexDegree[#]] == degrees &]
```

```
TreesWithDegreeSequence[{1, 1, 1, 1, 3, 3}]
```



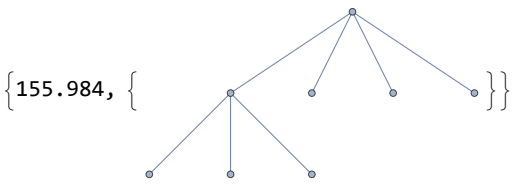
Solving for the order of the graph finds that the above graph is the only one satisfying those criteria:

```
Solve[1/2 (2/3 n * 1 + 1/3 n * 3) == n - 1, n]
```

```
{{n → 6}}
```

4.17

```
TreesWithDegreeSequence[{1, 1, 1, 1, 1, 1, 4, 4}] // Timing
```



b) Solve for the order of such a graph:

```
Solve[1/2 (3/4 n*1 + 1/4 n*4) == n - 1, n]
```

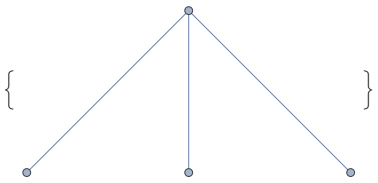
```
{{n → 8}}
```

c) Solve for n_4 (representing one quarter of the order) and m (the 'other' degree) gives just one other graph with the requested properties:

```
Solve[1/2 (3*n4*1 + n4*m) == 4*n4 - 1 && n4 > 0 && m > 0, {n4, m}, Integers]
```

```
{{n4 → 1, m → 3}, {n4 → 2, m → 4}}
```

```
TreesWithDegreeSequence[{1, 1, 1, 3}]
```



d) Solve for n_4 and m:

```
Solve[1/2 (3*n4*m + n4*1) == 4*n4 - 1 && n4 > 0 && m > 0, {n4, m}, Integers]
```

```
{{n4 → 2, m → 2}}
```

```
TreesWithDegreeSequence[{1, 1, 2, 2, 2, 2, 2, 2}] // Timing
```

```
{160.156, { }}
```

4.18

Solve for the number m of vertices of degree 3 as a function of the order n:

```
Solve[1/2 (m*3 + (n - m)*1) == n - 1, m]
```

$$\left\{\left\{m \to \frac{1}{2}\,(-2+n)\right\}\right\}$$

4.19

$2\,(+_i\, n_i\, -\, 1) = +_i\, i\, n_i \;\Rightarrow +_i 2\, n_i\, -\, 2 = +_i n_i \;\Rightarrow +_i (2\, n_i - i\, n_i) = 2 \;\Rightarrow (2\, n_1 - n_1) = 2 + \sum_{i=2}^{\infty} (i-2)\, n_i \;\Rightarrow n_1 = 2 + n_3 + 2\, n_4 + 3\, n_5 + \ldots$

```
2 + 5 + 2*2
```

```
11
```

4.20

a) Any cycle C_n has size $m = n$; so there exist graphs with three cycles having size $m = n \neq n + 2$.
b) By Theorem 3, every nontrivial tree has at least two end-vertices; that is, vertices of degree 1. Therefore, a regular nontrivial tree must be of degree 1, that is, P_2. The only other regular tree is trivial, that is, P_1.

4.21

C_{n+2} is regular of degree 2 and therefore $\overline{C}_{n+2}$ is regular of degree $((n+2)-1)-2 = n-1$. Thus by Theorem 9 T is isomorphic to a subgraph of $\overline{C}_{n+2}$.

4.22

$m_T = n-1$; $m_{\overline{T}} = m_{K_n} - m_T = \frac{1}{2} n(n-1) - (n-1) = \left(\frac{1}{2} n - 1\right)(n-1) = \frac{1}{2}(n-2)(n-1) = m_{K_{n-1}}$.

4.23

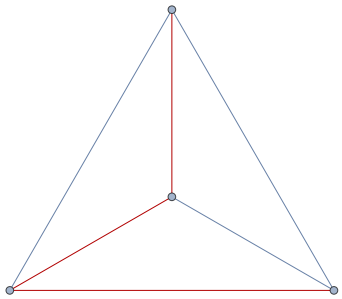
Since $m_T = n-1$, for $\overline{T}$ to also be a tree it would also have $m_{\overline{T}} = n-1$. Using the result from the previous exercise, $m_{\overline{T}} = \frac{1}{2}(n-2)(n-1)$. Thus:

```
Solve[n - 1 == 1/2 (n - 2) (n - 1), {n}]
```

```
{{n → 1}, {n → 4}}
```

That is, the trivial graph, and P_3 with $\overline{P}_3 = P_3$:

```
HighlightGraph[CompleteGraph[4], {1 ↔ 3, 1 ↔ 4, 2 ↔ 4}, ImageSize → Small]
```



4.24

The only tree of 3 vertices (has to have at least two end-vertices) is P_3; so in each selection of 3 vertices of G, two vertex pairs must be connected by an edge in G and one vertex pair must not be connected by an edge in G. There are $\binom{n}{3} = p$ unique ways to pick a set of 3 vertices; since in each of those ways the induced subgraph has two edges we will count $p \cdot 2$ edges in total. On the other hand, in an enumeration of all sets of 3 vertices of G, each vertex pair of G will appear $p \cdot (n-2)$ times. G has $\frac{1}{2} n(n-1)$ vertex pairs; say k of those are actual edges. Then in an enumeration of all sets of 3 vertices of G, we would count $p \cdot (n-2) \frac{k}{\frac{1}{2} n(n-1)}$ edges. A necessary condition for a graph solution G to exist is for $p \cdot 2 = p \cdot (n-2) \frac{k}{\frac{1}{2} n(n-1)} \Rightarrow 1 = (n-2) \frac{k}{n(n-1)} \Rightarrow k = \frac{n(n-1)}{n-2}$ to have an integral solution for n, k:

```
Solve[k == (n (n - 1))/(n - 2) && k ≤ 1/2 n (n - 1), {k}, Integers]
```

```
{{k → ConditionalExpression[0, n == 0 || n == 1]}, {k → ConditionalExpression[6, n == 4]}}
```

We need the additional constraint to eliminate nonsensical solutions where k is greater than the number of edges in a complete graph. So the only possible nontrivial solution is K_4; but this does not have the required property that every subgraph is a tree. So there are no solutions.

[Not sure how to generalize this]

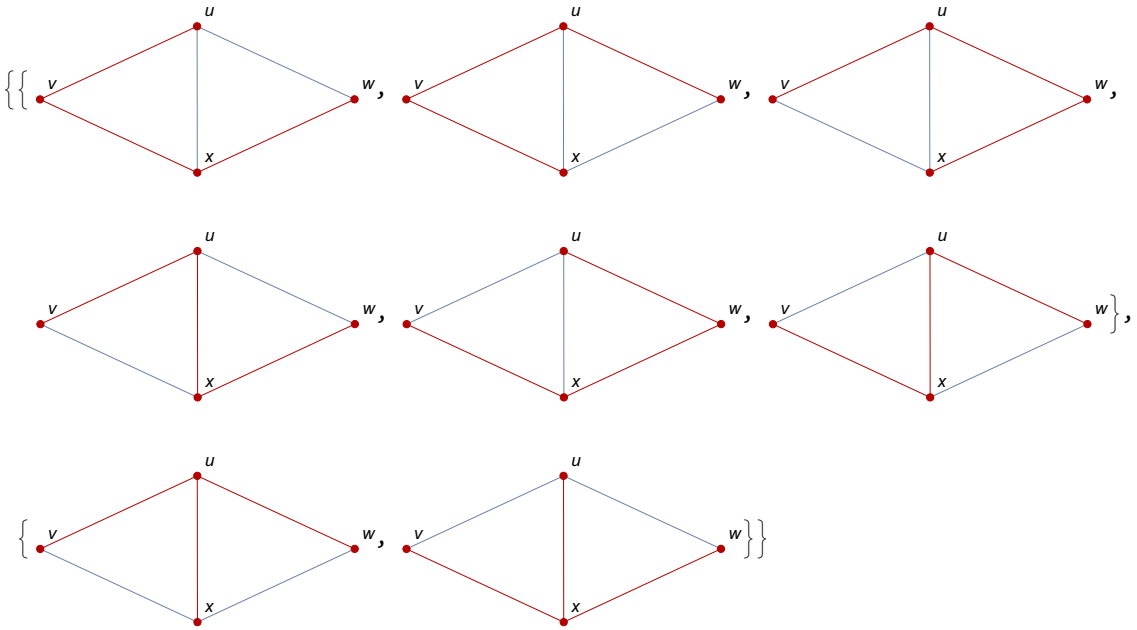
4.3 Minimum Spanning Tree Problem

4.25

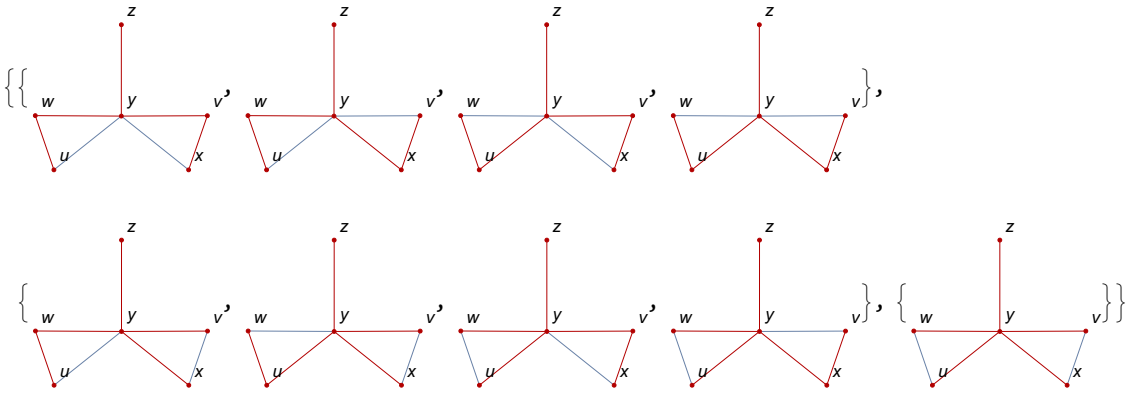
```
FindSpanningTrees[g_] :=
 Select[
  Graph /@ Subsets[
    EdgeList[g],
    {VertexCount[g] - 1}],
  AcyclicGraphQ]
```

```
HighlightIsomorphicSpanningTrees[g_] :=
 Map[
  HighlightGraph[g, #] &,
  Gather[
   DeleteDuplicates[
    FindSpanningTrees[g]],
   IsomorphicGraphQ],
  {2}]
```

```
HighlightIsomorphicSpanningTrees[
 Graph[{u ↔ v, v ↔ x, x ↔ w, w ↔ u, u ↔ x}, VertexLabels → "Name"]]
```




```
HighlightIsomorphicSpanningTrees[
 Graph[{u ↔ w, w ↔ y, y ↔ u, y ↔ z, y ↔ v, v ↔ x, x ↔ y}, VertexLabels → "Name", ImageSize → Tiny]]
```



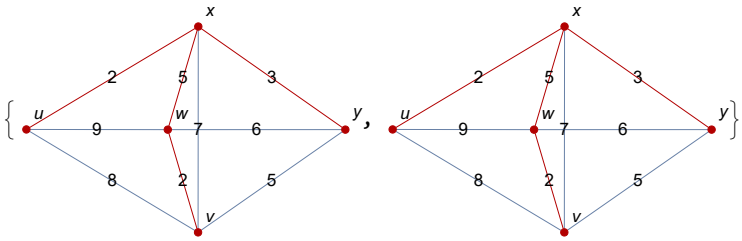
4.26

Assume that an edge *e* is a bridge; then *G* – *e* is disconnected. Since a spanning tree of *G* is a subgraph that contains every vertex of *G*, it follows that *e* must be in any spanning tree. Conversely, assume that an edge *e* is in every spanning tree of *G*; let *e* be incident to vertices *u* and *v*. Suppose that *e* is not a bridge of *G*; then *e* lies on a cycle *C*. Then we could construct a new spanning tree from *C* – *e*, contradicting that *e* is in every spanning tree of *G*. Thus *e* is a bridge.

4.27

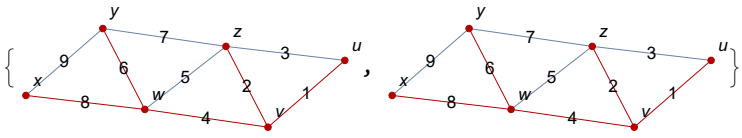
```
(* Highlight minimum spanning trees derived from different methods *)
HighlightMinimumSpanningTrees[g_] :=
 HighlightGraph[
    g,
    FindSpanningTree[g, Method → #]] & /@ {"Prim", "Kruskal"}
```

```
HighlightMinimumSpanningTrees[
 Graph[
  {u ↔ v, u ↔ w, u ↔ x, x ↔ v, x ↔ w, x ↔ y, y ↔ v, y ↔ w, v ↔ w},
  EdgeWeight → {8, 9, 2, 7, 5, 3, 5, 6, 2},
  EdgeLabels → "EdgeWeight", VertexLabels → "Name"]]
```



4.28

```
HighlightMinimumSpanningTrees[
 Graph[
  {u ↔ v, u ↔ z, v ↔ z, v ↔ w, z ↔ y, z ↔ w, w ↔ y, w ↔ x, y ↔ x},
  EdgeWeight → {1, 3, 2, 4, 7, 5, 6, 8, 9},
  EdgeLabels → "EdgeWeight", VertexLabels → "Name"]]
```



4.29

In this case, Kruskal's algorithm offers no choice in the edge selection process; since this produces a minimum spanning tree, the resulting tree is unique.

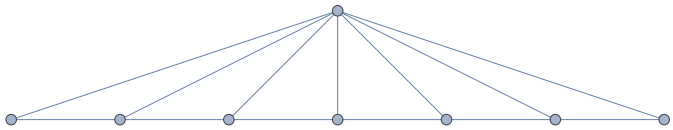
4.30

Any edge not in the spanning tree is necessarily incident with one of its vertices. In the process of building a minimum spanning tree with Prim's algorithm, at any point that some edge was incident with the tree under construction it must have had higher weight than all other edges in the path connecting both its ends.

4.31

Take any graph G with a minimum spanning tree, and add a vertex that is incident to each vertex of G with equal weight that is greater than the weight of any edge of G. In general, this could be constructed in the following manner:

```
Module[
 {g = PathGraph[Range[7], EdgeWeight → ConstantArray[1, 6]]},
 Graph[
  EdgeAdd[g,
   Table[
    i ⟷ VertexCount[g] + 1,
    {i, VertexCount[g]}]], (* can't add weighted edge to graph *)
  VertexCoordinates → Append[
    PropertyValue[{g, #}, VertexCoordinates] & /@ VertexList[g],
    {(VertexCount[g] - 1)/2, 1}]]]
```



Unfortunately there seems to be no way to add a weighted edge to a graph in Mathematica.

4.4 The Number of Spanning Trees

4.32

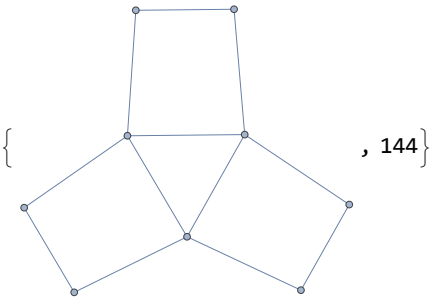
C_n has n spanning trees for $n \geq 3$; P_2 has 1 spanning tree. Suppose there is a graph containing exactly 2 spanning trees T, T'. Let u, v be two vertices that are incident in T but not in T'. Then there must be some other $u - v$ path through another vertex w such that $u - w$ and $w - v$ are in T'. But then $u\,v$, $u - w$, $w - v$ form a cycle with at least three ways to span it.

4.33

Suppose F is a subgraph of a spanning tree of G. Since a tree has no cycles, F has none either. For the converse, suppose F has no cycles. Then we can use F as a basis to generate a spanning tree of G using Prim's algorithm; and therefore F is a subgraph of that spanning tree.

4.34

```
#[Graph[{A ⟷ B, B ⟷ C, C ⟷ A, A ⟷ AB, AB ⟷ BA,
      BA ⟷ B, B ⟷ BC, BC ⟷ CB, CB ⟷ C, C ⟷ CA, CA ⟷ AC, AC ⟷ A}]] & /@
 {Identity, Length[FindSpanningTrees[#]] &}
```



Separate into cases depending on whether 0, 1, or 2 edges from the center cycle A, B, C are present. For the first case (0 edges of ABC) there are 9 separate trees, depending on which of the outer edges is removed. For the 3 subcases of 1 edge of ABC, one of three edges has to be removed from the remaining 'lobe' cycle and one of the six edges from the outer edges. For the 3 subcases of 2 edges of ABC, one of three edges has to be removed from both of the remaining 'lobe' cycles, and one from the outer edges; giving:

```
9 + 3 * 3 * 6 + 3 * 3 * 3 * 3
```

144

Generalizing for the size of the lobes:

```
CountSpanningTrees434[k_] =
 1 * (3 (k - 1)) +
   3 * ((k - 1) * 2 (k - 1)) +
   3 * ((k - 1)^2 * (k - 1)) // Simplify
```

$3\ (-1+k)\ k^2$

```
CountSpanningTrees434 /@ Range[10]
```

{0, 12, 54, 144, 300, 540, 882, 1344, 1944, 2700}

4.35

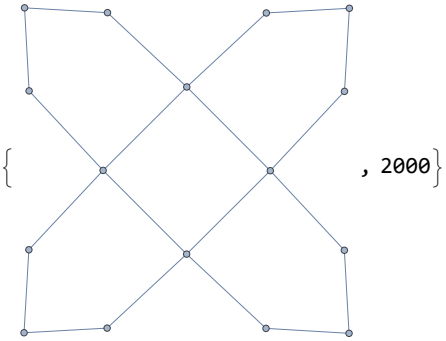
```
CountSpanningTrees435[k_] =
 Binomial[4, 0] * (4 (k - 1)) +
   Binomial[4, 1] * ((k - 1) * 3 * (k - 1)) +
   Binomial[4, 2] * ((k - 1)^2 * 2 * (k - 1)) +
   Binomial[4, 3] * ((k - 1)^3 * (k - 1)) // Simplify
```

$4\ (-1+k)\ k^3$

```
CountSpanningTrees435[5]
```

2000

```
#[Graph[{A ↔ B, B ↔ C, C ↔ DD, DD ↔ A, A ↔ A1, A1 ↔ A2, A2 ↔ A3, A3 ↔ B, B ↔ B1, B1 ↔ B2, B2 ↔ B3,
      B3 ↔ C, C ↔ C1, C1 ↔ C2, C2 ↔ C3, C3 ↔ DD, DD ↔ D1, D1 ↔ D2, D2 ↔ D3, D3 ↔ A}]] & /@
 {Identity, Length[FindSpanningTrees[#]] &}
```



4.36

```
(* count the number of spanning trees of graphs of the general form of Figure 20,
where n is the number of lobes and k the size of each lobe cycle *)
CountSpanningTrees436[n_, k_] = Sum[
  Binomial[n, i] ((k - 1)^i * (n - i) (k - 1)),
  {i, 0, n - 1}]
```

$(-1 + k)\ k^{-1+n}\ n$

```
CountSpanningTrees436[3, 4]
CountSpanningTrees436[4, 5]
CountSpanningTrees436[5, 4]
```

144

2000

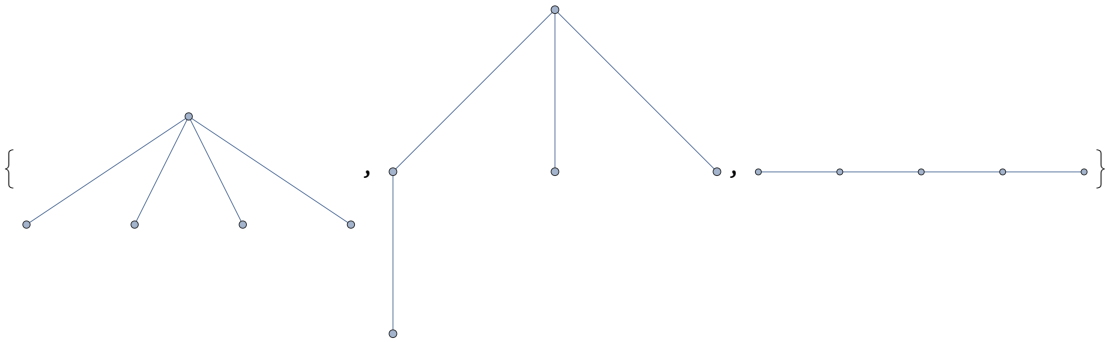
3840

4.37

```
5^(5-2)
```

125

```
Trees[5]
```

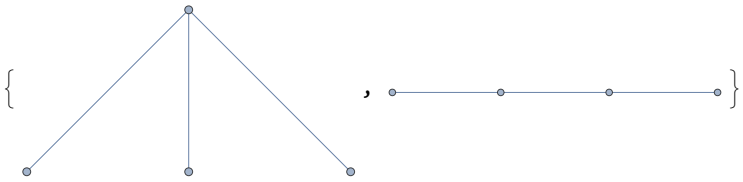


```
5 + 5 * 4 * 3 + 5 * 4 * 3
```

125

4.38

```
Trees[4]
```



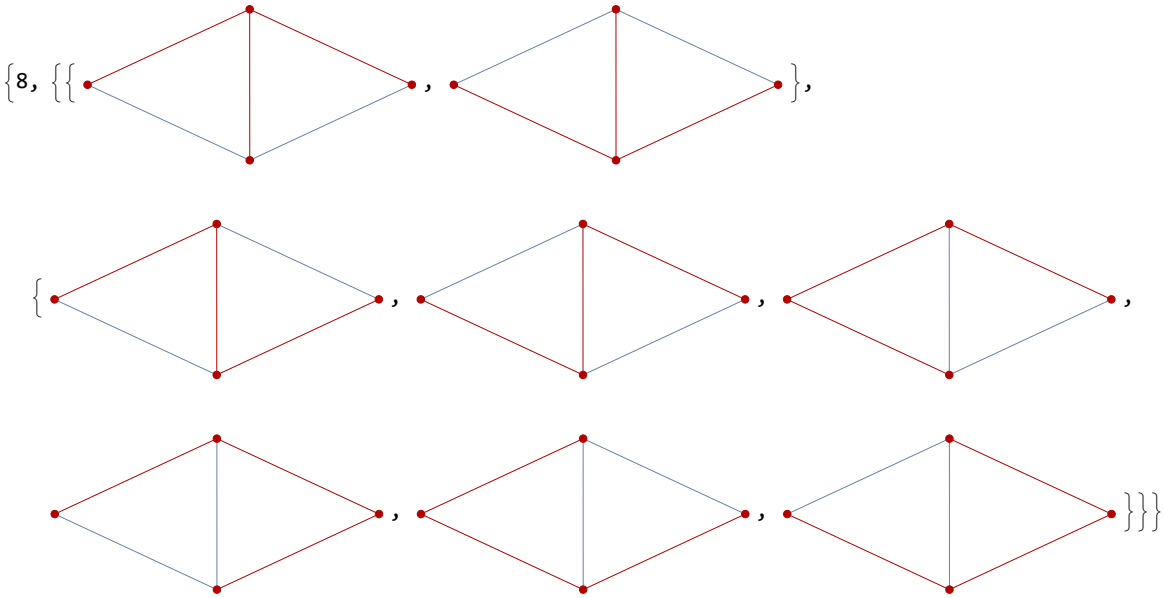
```
CountByMatrixTreeTheorem[g_] :=
 (* number of spanning trees according to the Matrix Tree Theorem *)
 Det[
  (* C matrix of Matrix Tree Theorem, with first row and column removed *)
  Module[
   {ga = AdjacencyMatrix[g]},
   Table[
    If[i == j,
     VertexDegree[g, VertexList[g][[i]]],
     -ga[[i, j]]],
    {i, 2, 4}, {j, 2, 4}]]]
```

```
Total[CountByMatrixTreeTheorem /@ Trees[4]]
```

2

4.39

```
#[Graph[{v1 ⟷ v2, v1 ⟷ v3, v1 ⟷ v4, v3 ⟷ v2, v4 ⟷ v2}]] & /@
 {CountByMatrixTreeTheorem, HighlightIsomorphicSpanningTrees}
```



4.40

Pick an edge $u\,v$ from T' that is not in T. Then there is a different path $u - v$ in T that forms a cycle with $u\,v$. Since T' is a tree and acyclic, there is an edge in the path $u - v$ that is not in T'. Construct T_1 by this procedure of removing that edge and adding $u\,v$. Since this graph is still connected and the number of edges is unchanged, by Theorem 8 T_1 is a tree. Also, all but two of the edges of T were changed to produce T_1. Repeat this procedure until T' has no more edges that are not in T_i.

5 Connectivity

5.1 Cut-Vertices

```
Bridges[g_] :=
 Select[
  EdgeList[g],
  Not[ConnectedGraphQ[EdgeDelete[g, #]]] &]
```

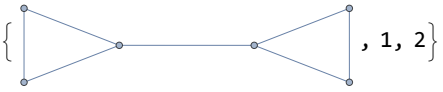
```
CutVertices[g_] :=
 Select[
  VertexList[g],
  Not[ConnectedGraphQ[VertexDelete[g, #]]] &]
```

```
Blocks[g_] :=
 Module[
  {fc = FindFundamentalCycles[g]},
  Join[
   fc,
   List /@ Complement[
     EdgeList[g],
     Flatten[fc, 1]]]]
```

5.1

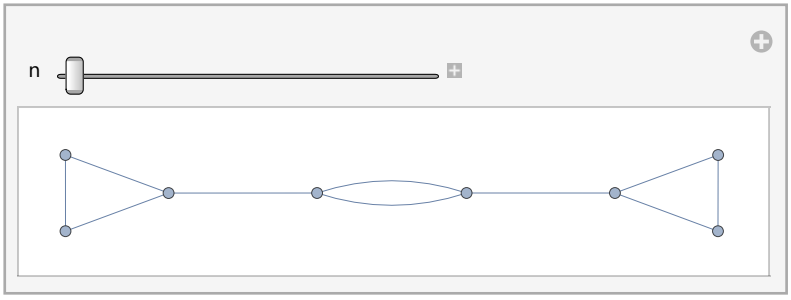
```
#[PathGraph[Range[4]]] & /@ {Identity, Length @* Bridges, Length @* CutVertices}
```

```
#[Graph[{L ⟷ L1, L1 ⟷ L2, L2 ⟷ L, L ⟷ R, R ⟷ R1, R1 ⟷ R2, R2 ⟷ R}]] & /@
 {Identity, Length @* Bridges, Length @* CutVertices}
```



5.2

```
Manipulate[
 Graph[
  Join @@ Table[
    {Ci ⟷ CMod[i+1,IntegerPart[n]], Ci ⟷ Oi, Oi ⟷ Ai, Ai ⟷ Bi, Bi ⟷ Oi}, {i, 0, n - 1}]],
 {n, 2, 10}]
```

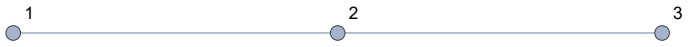


5.3

```
Graph[{A ⟷ B, A ⟷ A1, A1 ⟷ A2, A2 ⟷ A, B ⟷ B1, B1 ⟷ B2, B2 ⟷ B}, VertexLabels → "Name"]
```

Vertices *A* and *B* lie on cycles, yet are both cut-vertices.

```
PathGraph[Range[3], VertexLabels → "Name"]
```



Vertices 1 and 3 do not lie on any cycles, yet neither are cut-vertices.

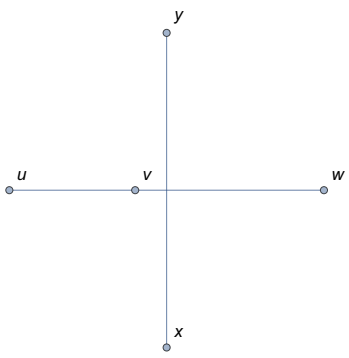
```
#[PathGraph[Range[3]]] & /@ {Identity, Length @* CutVertices, Length @* Bridges}
```

and has two end-vertices; disproving also both c) and d).

5.4

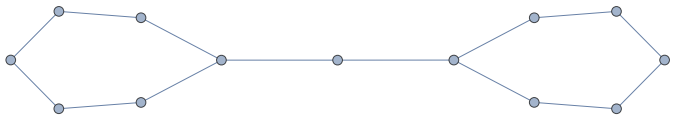
By Corollary 4, if *v* is a cut-vertex of *G* then $\exists\, u, w \in G$ such that *v* lies on every *u*—*w* path of *G*. Now suppose that *v* were also a cut-vertex of $\overline{G}$; then $\exists\, x, y \in \overline{G}$ such that *v* lies on every *x*—*y* path of $\overline{G}$. Then *x*, *y* are not incident in $\overline{G}$ and are incident in *G*. Consider the case that *u* is incident with either *x* or *y* in *G*; then neither *x* nor *y* can be incident with *w* or we would have a *u*—*w* path in *G* not through *v*. This means that in $\overline{G}$, if *u* is not incident with both *x* and *y*, then *x* and *y* are both incident with *w*; but then we have an *x*—*y* path not through *v*. So in $\overline{G}$, *u* must be incident with both *x* and *y*; but then, again, we have a *x*—*y* path not through *v*.

```
Graph[
 {u, v, w, x, y}, {u ⟷ v, v ⟷ w, x ⟷ y},
 VertexCoordinates → {{-1, 0}, {-0.2, 0}, {+1, 0}, {0, -1}, {0, +1}},
 VertexLabels → "Name",
 ImageSize → Small]
```



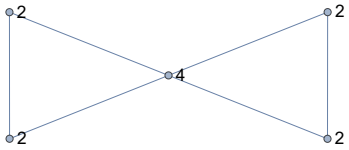
5.5

```
Graph[
 Join[
  Table[L_i ⟷ L_Mod[i+1,6], {i, 0, 5}],
  Table[R_i ⟷ R_Mod[i+1,6], {i, 0, 5}],
  {L_0 ⟷ v, v ⟷ R_0}]]
```



The above connected graph is of order 13; the center vertex is a cut-vertex, but both resulting components are of order 6.

```
Graph[{0 ⟷ 1, 1 ⟷ 2, 2 ⟷ 0, 0 ⟷ 3, 3 ⟷ 4, 4 ⟷ 0}, VertexLabels → "VertexDegree", ImageSize → Small]
```



The above connected graph has only even vertices, yet the center is a cut-vertex. Also, it does not contain a bridge.

```
Graph[{0 ⟷ 1}, ImageSize → Small]
```


Finally, the above connected graph has a bridge but no cut-vertex.

5.6

First, suppose that G has a cut-vertex v; then $\exists\, u, w \in G$ such that v lies on every u—w path. v is incident with three vertices, say v_1, v_2, v_3. WLOG, let all u—w paths be u—v_1—v—v_2—w: disregarding for a moment v_3, the paths must all be this way for if there were a path u—v_2—v—v_1—w we would also have u—v_1—w not through v. For the same reason, either u or v might pass through v_3—v, but not both; again WLOG suppose then that the other u—w paths are u—v_1—v—v_3—w. Then $v_1 - v$ is a bridge. Conversely, suppose that G has a bridge. Then by Theorem 1, since both vertices incident with the bridge are of degree $3 \geq 2$, they are both cut-vertices.

5.7

By Theorem 4.3, T has at least two end-vertices; let u be one of them. Let v be a vertex that is farthest from u; then by Theorem 5, v is not a cut-vertex and thus an end-vertex. Now consider the vertex v' that is incident with v. Note first that v' cannot be the same as u, because both v and u are end-vertices which would mean that T is of order $2 < 3$; therefore v' has at least one other incident edge that lies on the path to u. Now suppose that there are other edges incident to v'; the vertices on the other side of any such edges must also be end-vertices; for if they were not, then those would lie on paths to end-vertices that were more distant to u then v was. Therefore, every vertex adjacent to v' is an end-vertex, with the possible exception of the adjacent vertex that lies on the path to u.

5.8

a) Take some v that is an end-vertex of a spanning tree of G. If v is also an end-vertex of G itself, then obviously it is not a cut-vertex of G. Then assume that v is not an end-vertex of G; then there are at least two vertices u, w adjacent to it. Then if v were a cut-vertex, $G - v$ would be disconnected with u, w in distinct components, and by Theorem 3, v then lies on every u—w path. But then a spanning tree of G, which necessarily includes u, w would have to include both edges $u\,v$ and $v\,w$ and v would not be an end-vertex of the tree. Therefore v cannot be a cut-vertex.

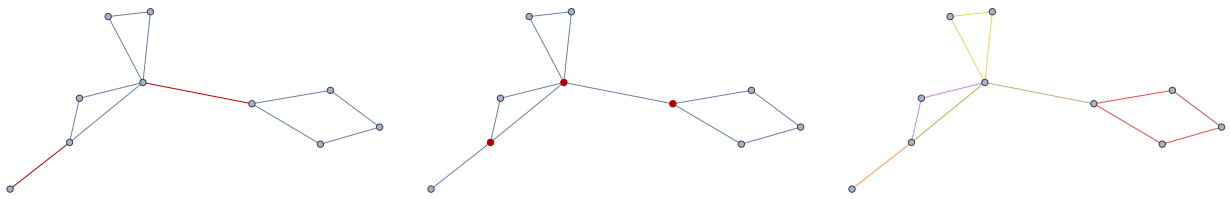
b) Every connected graph contains a spanning tree (Theorem 4.10), and every nontrivial tree has at least two end-vertices (Theorem 4.3). By a) then those end-vertices cannot be cut-vertices of the graph.

c) Weigh the edges incident to v minimally, that is with lower weight than every other edge in G. No such edge can be incident with any other such edge other than by v; then, Prim's algorithm will generate a tree containing all those edges.

d) Suppose G has no other vertices than the two that are not cut-vertices; then $G = P_2$. So assume G has other vertices, which are all cut-vertices; pick any one v of those. By c) there exists a spanning tree T of G that contains all edges incident to v. Now suppose that v is of degree greater than 2; in that case, T has more than two end-vertices, and by a) those are not cut-vertices of G; contradicting the assumption that G has only two. So all vertices of G are cut-vertices of degree 2 except for the two that are not cut-vertices; thus G is a path.

5.9

```
GraphicsRow[
 Module[
  {g = Graph[{q ↔ r, r ↔ s, s ↔ t, r ↔ t, t ↔ u, u ↔ v, v ↔ t, t ↔ w, w ↔ x, x ↔ y, y ↔ z, z ↔ w}]},
  HighlightGraph[g, #[g]] & /@
   {Bridges, CutVertices, Blocks}],
 ImageSize → Full]
```



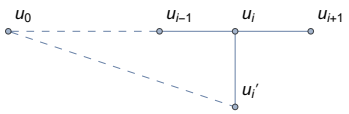
5.10

First, a graph of size 2 is at least of order 3. By Theorem 7, a graph of at least order 3 is nonseparable iff every two vertices lie on a common cycle. Thus the problem can be restated as follows: prove that for a connected graph of at least order 3, every two vertices lie on a common cycle iff every two adjacent edges lie on a common cycle.
The forward equivalence is clear: given any two adjacent edges $u - v - w$, consider u, w. Since every two vertices lie on a common cycle there is some other u—w path that does not go through v; this path together with $u - v - w$ is a cycle, and

$u - v$, $v - w$ lie on a common cycle.

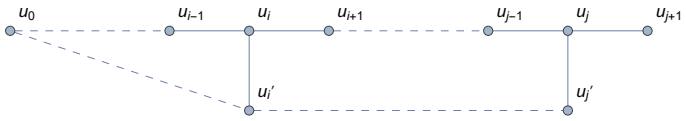
Now consider the reverse equivalence: let it be given that every two adjacent edges lie on a common cycle. Consider any two vertices u, v; since G is connected, there is a path $P = u_{i:0\ldots n-1}$, $u_0 = u$, $u_{n-1} = v$. By induction: let $u_{i>0}$ such that u_0, u_i lie on a common cycle; u_i exists since at least u_0, u_1 are adjacent. Let P' be the part of the common cycle that is not on P; and let u_i' be the vertex adjacent to u_i on P':

```
Graph[
 {u0 ⟷ ui-1, ui-1 ⟷ ui, ui ⟷ ui+1, ui ⟷ ui′, ui′ ⟷ u0},
 VertexCoordinates → {{0, 0}, {2, 0}, {3, 0}, {4, 0}, {3, -1}},
 EdgeStyle → {u0 ⟷ ui-1 → Dashed, ui′ ⟷ u0 → Dashed},
 VertexLabels → "Name",
 ImageSize → Small]
```



Continue this process from u_i: since u_i', u_{i+1} are adjacent they lie on a common cycle.

```
Graph[
 {u0 ⟷ ui-1, ui-1 ⟷ ui, ui ⟷ ui+1, ui ⟷ ui′,
  ui′ ⟷ u0, ui+1 ⟷ uj-1, uj-1 ⟷ uj, uj ⟷ uj+1, uj ⟷ uj′, uj′ ⟷ ui′},
 VertexCoordinates → {{0, 0}, {2, 0}, {3, 0}, {4, 0}, {3, -1}, {6, 0}, {7, 0}, {8, 0}, {7, -1}},
 EdgeStyle → {u0 ⟷ ui-1 → Dashed, ui′ ⟷ u0 → Dashed, ui+1 ⟷ uj-1 → Dashed, uj′ ⟷ ui′ → Dashed},
 VertexLabels → "Name",
 ImageSize → Medium]
```



Until $u_{j+1} = v$. Then we can construct a common cycle from P and v—…—u_j'—u_i'—u.

```
Graph[
 {u0 ⟷ ui-1, ui-1 ⟷ ui, ui ⟷ ui+1, ui ⟷ ui′, ui′ ⟷ u0,
  ui+1 ⟷ uj-1, uj-1 ⟷ uj, uj ⟷ uj+1, uj ⟷ uj′, uj′ ⟷ ui′, uj+1 ⟷ un-1, un-1 ⟷ uj′},
 VertexCoordinates → {{0, 0}, {2, 0}, {3, 0}, {4, 0}, {3, -1},
   {6, 0}, {7, 0}, {8, 0}, {7, -1}, {10, 0}},
 EdgeStyle → {u0 ⟷ ui-1 → Dashed, ui′ ⟷ u0 → Dashed, ui+1 ⟷ uj-1 → Dashed,
   uj′ ⟷ ui′ → Dashed, uj+1 ⟷ un-1 → Dashed, un-1 ⟷ uj′ → Dashed},
 VertexLabels → "Name",
 ImageSize → Large]
```

5.11

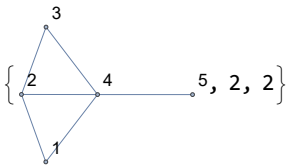
Suppose G were separable; then it would have a cut-vertex v and two components in $G - v$. At most, this could have reduced the degree of each vertex of G to no lower than $\frac{1}{2}n - 1$; and then the order of either component would have to be at least $\frac{1}{2}n$. But then the order of G would have had to have been at least $2\left(\frac{1}{2}n\right) + 1 = n + 1$, which is a contradiction.

5.12

If there are three blocks, there are $\binom{3}{2} = 3$ pairs of distinct blocks, and by Corollary 9b these can have at most three vertices in common. Since G is connected, the blocks must have at least two vertices in common. By Corollary 9c these common vertices are cut-vertices; so $k \in \{2, 3\}$.

5.13

```
Module[
 {g = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 1, 2 ↔ 4, 4 ↔ 5}, VertexLabels → "Name", ImageSize → Tiny]},
 {g, GraphDiameter[g], GraphDistance[g, 1, 3]}]
```



5.14

a) Suppose that the induced subgraph were not connected. Since G was connected, v was connected to G_1 through another vertex of G_2. But then there are vertices of both components that are connected through a path that doesn't pass through v, and then v could not be a cut-vertex.

In the graph below, 3 is a cut-vertex, but the corresponding induced subgraph (highlighted) is not a block:

```
Module[
 {g = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 1, 3 ↔ 4, 4 ↔ 5, 5 ↔ 6, 6 ↔ 4}, VertexLabels → "Name"]},
 HighlightGraph[g, VertexDelete[g, {1, 2}]]]
```

5.15

*** INCOMPLETE ***

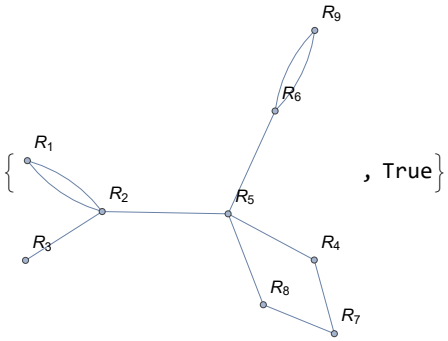
6 Traversability

6.1 Eulerian Graphs

```
EulerianTrailGraphQ[g_] :=
 (* does adding any edge make an Eulerian cycle *)
 AnyTrue[
  (* add an edge between the pair to make a cycle *)
  EdgeAdd[g, UndirectedEdge @@ #] & /@
   (* all pairs of vertices *)
   Subsets[VertexList[g], {2}],
  EulerianGraphQ]
```

6.1

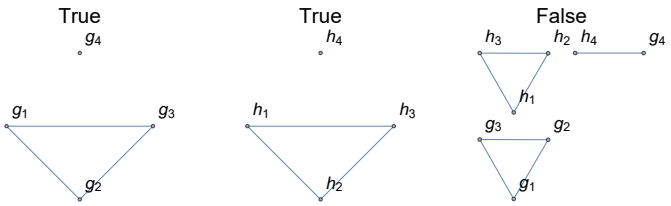
```
Module[
 {g = Graph[
    {R1 ⟷ R2, R1 ⟷ R2, R2 ⟷ R3, R4 ⟷ R5, R2 ⟷ R5, R5 ⟷ R6, R4 ⟷ R7, R7 ⟷ R8, R5 ⟷ R8, R6 ⟷ R9, R6 ⟷ R9},
    VertexLabels → "Name", ImageSize → Small]},
 {g, EulerianTrailGraphQ[g]}]
```



6.2

It cannot be said that *G* is Eulerian, because it might be disconnected:

```
Module[
 {
  G = RenameSubscriptGraph[
    VertexAdd[
     CompleteGraph[3, VertexLabels → "Name"], 4],
    g],
  H = RenameSubscriptGraph[
    VertexAdd[
     CompleteGraph[3, VertexLabels → "Name"], 4],
    h]},
 GraphicsRow[
  Show[#, PlotLabel → EulerianGraphQ[#]] & /@ {
    G, H,
    EdgeAdd[
     GraphUnion[G, H, VertexLabels → "Name"],
     g4 ↔ h4]},
  Scaled[0.2]]]
```

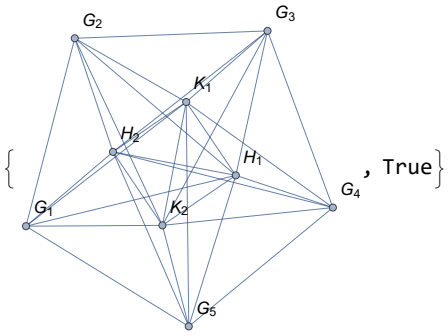


6.3

Restating: let G, H, K be the pairwise disjoint connected regular graphs, where G and $\overline{G}$ are Eulerian and H, K are not. Since H is not Eulerian, by Theorem 6.1 it must have at least one vertex of odd degree; and since it is regular, therefore all vertices of H have the same odd degree $\deg_H$. Since H is regular, by Theorem 2.1 it has even order. Similarly, all vertices of K have the same odd degree $\deg_K$ and K is of even order. Now G is Eulerian and regular, so by Theorem 6.1 all its vertices are of the same even degree $\deg_G$; Theorem 2.1 does not say anything about its order. However, $\overline{G}$ is also Eulerian and regular, so all its vertices are also of the same even degree $\deg_{\overline{G}}$. Therefore each vertex of G, $\overline{G}$ is connected to $\deg_G$ vertices of G and *not* connected to $\deg_{\overline{G}}$ vertices of G, and since $\deg_G + \deg_{\overline{G}}$ is even, the order of G is odd; and thus since G is regular, by Theorem 2.1 $\deg_G$ is even.

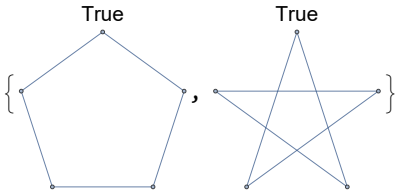
Now then, in $G + (H + K)$, note first that the components are mutually disjoint. Then, each vertex from the G component has degree $\deg_G + \text{order}_H + \text{order}_K$, which is even; each vertex of the H component has degree $\deg_H + \text{order}_K + \text{order}_G$, which is is also even; and similarly each vertex of the K component has even degree. Since every vertex of the join has even degree, by Theorem 6.1 it is Eulerian.

```
#[
   GraphJoin[
    RenameSubscriptGraph[CycleGraph[5], G],
    GraphJoin[
     RenameSubscriptGraph[CompleteGraph[2], H],
     RenameSubscriptGraph[ CompleteGraph[2], K],
     VertexLabels → "Name"],
    VertexLabels → "Name"]] & /@ {Show, EulerianGraphQ}
```

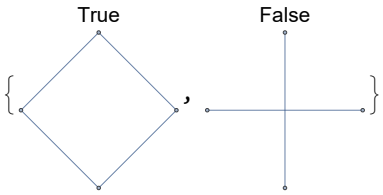


6.4

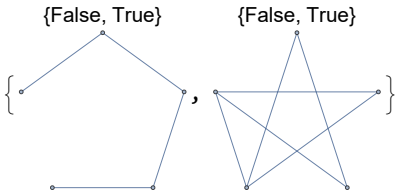
```
Module[ (* a. *)
 {g = CycleGraph[5]},
 Show[#, PlotLabel → EulerianGraphQ[#], ImageSize → Tiny] &
  /@ {g, GraphComplement[g]}]
```




```
Module[ (* b. *)
 {g = CycleGraph[4]},
 Show[#, PlotLabel → EulerianGraphQ[#], ImageSize → Tiny] &
  /@ {g, GraphComplement[g]}]
```

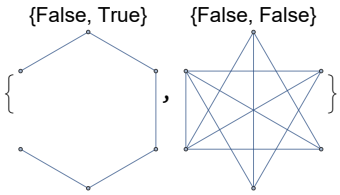


```
Module[ (* c. *)
 {g = EdgeDelete[CycleGraph[5], 1 ⟷ 2]},
 Show[#, PlotLabel → {EulerianGraphQ[#], EulerianTrailGraphQ[#]}, ImageSize → Tiny] &
  /@ {g, GraphComplement[g]}]
```

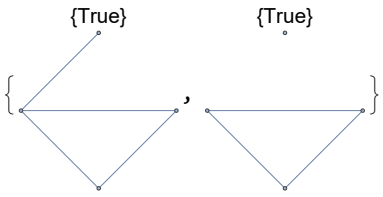


For d., note that because G has a Eulerian trail it must have exactly 2 vertices of odd degree. Similarly $\overline{G}$ must not have 2 odd vertices; but since it is Eulerian, it must have at least one odd vertex. Now if G were of odd order, then every vertex has an even number of neighbors; so any odd vertex in G is also odd in $\overline{G}$, and so $\overline{G}$ would also have 2 odd vertices, which is a contradiction. G must therefore be of even order, and every vertex has an odd number of neighbors; so any odd vertex in G is even in $\overline{G}$. This means that G cannot be of order 2 or $\overline{G}$ would consist of exactly 2 even vertices, which is a contradiction. Also, G cannot be of order 4 or $\overline{G}$ would have exactly 2 odd vertices, which is also a contradiction. So we try a graph of order 6:

```
Module[ (* d. *)
 {g = EdgeDelete[CycleGraph[6], 1 ⟷ 2]},
 Show[#, PlotLabel → {EulerianGraphQ[#], EulerianTrailGraphQ[#]}, ImageSize → Tiny] &
  /@ {g, GraphComplement[g]}]
```




```
Module[ (* e. *)
 {g = EdgeAdd[CycleGraph[3], 1 ⟷ 4]},
 {Show[g, PlotLabel → {EulerianTrailGraphQ[g]}, ImageSize → Tiny],
  Module[
   {h = EdgeDelete[g, 1 ⟷ 4]},
   Show[h, PlotLabel → {EulerianGraphQ[h]}, ImageSize → Tiny]]}
]
```



6.5

The vertices of K_5 are all of even degree, so the graph is Eulerian. Consider K_5 with a single edge removed: the only edge that

can be added to it is the one that was just removed, yielding K_5.

6.6

For G to be non-Eulerian it (Theorem 6.1) has to have at least one vertex of odd degree. Because it is regular, then all vertices have (the same) odd degree. By Theorem 2.6 then the order of G is even, and therefore all vertices of $\overline{G}$ are of (the same) even degree. Then, if $\overline{G}$ is connected, by Theorem 6.1 it is Eulerian.

6.7

a. The order of $\overline{G} \cong F$ is also equal to n and odd; since u and v are connected to each other and every vertex of G and $\overline{G} \cong F$, the degrees of u, v are equal and odd. Therefore H is not Eulerian.

b. Since G is r-regular of odd order, by Theorem 2.6 r must be even. Also, $\overline{G} \cong F$ is also of the same odd order and regular of even order. Therefore H contains exactly two vertices u and v of odd order, and by Corollary 6.2 contains a Eulerian trail.

```
Manipulate[
 #[
    Module[
     (* generate 2-regular graph of odd degree *)
     {K = RandomGraph[
        DegreeGraphDistribution[
         Table[2, k * 2 + 1]]]},
     Module[
      (* generate H as described in the problem*)
      {H = GraphUnion[
         RenameSubscriptGraph[K, G],
         RenameSubscriptGraph[GraphComplement[K], F]]},
      EdgeAdd[
       H,
       Append[
        Flatten[{u ⟷ #, v ⟷ #} & /@ VertexList[H]],
        u ⟷ v]]]] & /@ {
   Show[SetProperty[#, VertexLabels → "Name"]] &,
   EulerianGraphQ,
   EulerianTrailGraphQ},
 {k, 1, 5, 1}]
```

k

```
{Show[SetProperty[EdgeAdd[GraphUnion[
    RenameSubscriptGraph[ , G], RenameSubscriptGraph[ , F]],
  {u, GraphUnion[RenameSubscriptGraph[ , G],
    RenameSubscriptGraph[ , F], v, u ⊷ v}], VertexLabels → Name]], False,
 EulerianTrailGraphQ[EdgeAdd[GraphUnion[RenameSubscriptGraph[ , G],
   RenameSubscriptGraph[ , F]],
  {u, GraphUnion[RenameSubscriptGraph[ , G],
    RenameSubscriptGraph[ , F], v, u ⊷ v}]]]}
```

c. False, since by b. *H* has a Eulerian trail.

6.8

a. Consider the multigraph *G*' constructed from *G* by duplicating each edge; then *G*' has all vertices of even degree and is

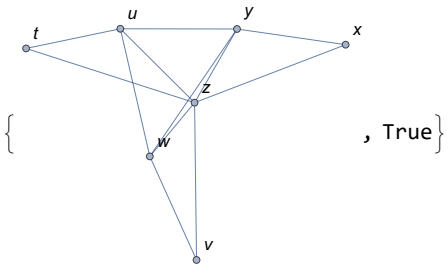
Eulerian. Its Eulerian circuit corresponds to a closed walk of G containing every edge twice; it is spanning because G is connected.

b. Similarly, a multigraph G'' constructed by triplicating every edge of G is Eulerian iff all vertices are of even degree; this is true iff all vertices of G are of even degree, that is, if G itself is Eulerian.

6.2 Hamiltonian Graphs

6.9

```
Module[
 {G = Graph[
    Join[
     z ⟷ # & /@ {t, u, v, w, x, y},
     {x ⟷ y, y ⟷ u, u ⟷ w, w ⟷ y, t ⟷ u, v ⟷ w}], VertexLabels → "Name"]},
 {
  G,
  Apply[And, (* Holds for any cardinality of S *)
   AllTrue[(* Holds for each possible S of the cardinality *)
     Length[ConnectedGraphComponents[
         VertexDelete[G, #]]] & /@
      Subsets[VertexList[G], {#}],
     LessEqualThan[#]] & /@
    Range[1, VertexCount[G] - 1]
    (* all possible cardinalities of a proper nonempty subset *)
  ]}]
```

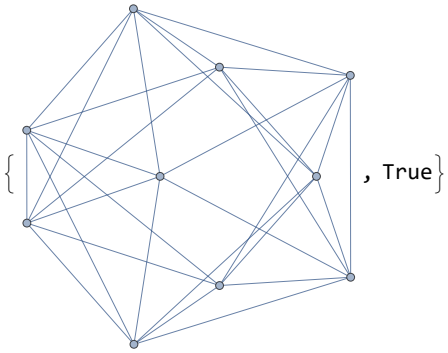


This shows that the requirement of $k(G - S) \leq |S|$ is necessary but not sufficient for a graph G to be Hamiltonian.

6.10

Since $\forall\, u, v \in G : \forall\, w \in G, G - u, G - u - v :\ \deg w \geq 5$, $\forall\, w, x \in G, G - u, G - u - v :\ \deg w + \deg x \geq 10$ so by Theorem 6 G, $G - u$, $G - u - v$ are Hamiltonian.

```
Module[
 {G = RandomGraph[DegreeGraphDistribution[Table[6, 10]]]},
 (* random 6-regular graph of order 10 *)
 {
  G,
  AllTrue[(* are all Hamiltonian *)
   VertexDelete[G, #] & /@ Subsets[VertexList[G], 2], (* removing up to 2 vertices *)
   HamiltonianGraphQ]}]
```



6.11

Since $\forall\, n : \forall\, u \in C_n : \deg_{\overline{C_n}} u = n - 2 - 1 = n - 3$, we have $\forall\, n : \forall\, u, v \in \overline{C_n} : \deg u + \deg v = (n-3) + (n-3) = 2\,(n-3)$.

```
Reduce[2 (n - 3) ≥ n, n] // TraditionalForm
```

$n \geq 6$

By Theorem 6 $\overline{C}_{n\geq 6}$ is Hamiltonian. Additionally, for $\overline{C}_5$ the degree of each vertex being $n - 3 = 2$ we see that $\overline{C}_5 = C_5$, which is obviously Hamiltonian.

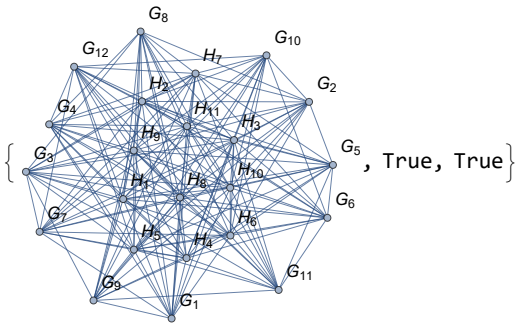
6.12

Note first that $\forall\, g \in G, h \in H :\ \deg_{G+H} g = 3 + 11 = 14,\ \deg_{G+H} h = 4 + 12 = 16$.

a. Since the degree of very vertex is even, by Theorem 1 $G + H$ is Eulerian.

b. Since for any two vertices $\forall\, u, v \in G + H : \deg u + \deg v \geq 14 + 14 = 28 \geq 23$, by Theorem 6 $G + H$ is Hamiltonian.

```
#[
   GraphJoin[
    RenameSubscriptGraph[RandomGraph[DegreeGraphDistribution[Table[3, 12]]], G],
    RenameSubscriptGraph[RandomGraph[DegreeGraphDistribution[Table[4, 11]]], H],
    VertexLabels → "Name"]] & /@ {
  Show, EulerianGraphQ, HamiltonianGraphQ}
```



, True, True}

6.13

a. Any Eulerian graph with the addition of an unconnected vertex. The graph cannot be Hamiltonian because the added vertex cannot be reached by any path.

b. All K_n where n is even have vertices with odd degree, so for n even and $n > 2$ are Eulerian. Additionally, by Theorem 6 they are Hamiltonian for:

```
Reduce[2 (n - 1) ≥ n, n] // TraditionalForm
```

$n \geq 2$

c. Any K_n with a single edge removed is Hamiltonian for:

```
Reduce[2 (n - 1 - 1) ≥ n, n] // TraditionalForm
```

$n \geq 4$

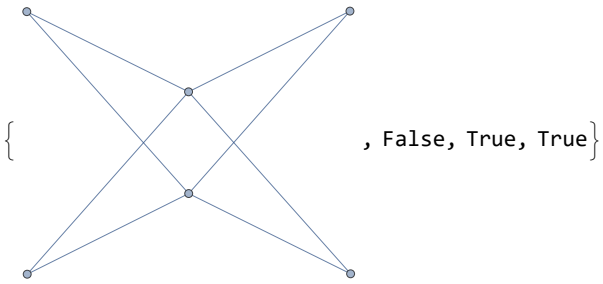
From b. K_n is Eulerian; then K_n with a single edge removed is not Eulerian but has a Eulerian trail.

d. Any $C_{n>2}$ with an additional unconnected vertex cannot be Hamiltonian. It is also not Eulerian, but is a Eulerian trail.

6.14

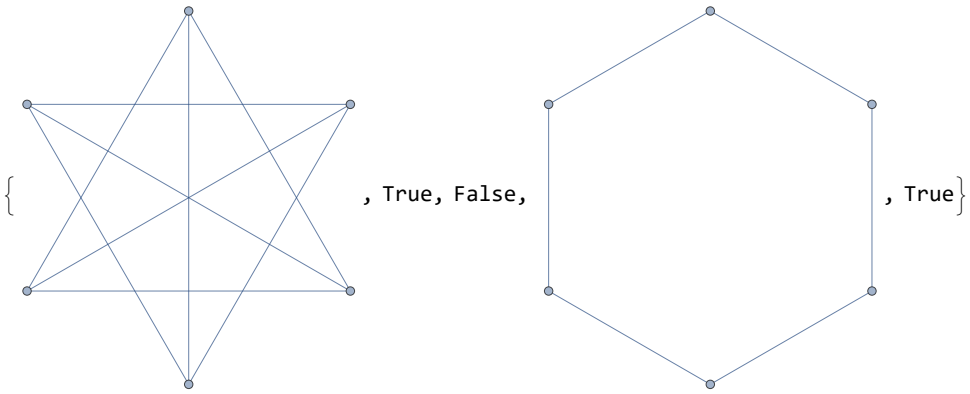
a. A Eulerian graph has a Eulerian *circuit*; a Hamiltonian graph has a Hamiltonian *cycle*. Construct a graph in which it is impossible to traverse all vertices without going through some other vertex more than once, for example:

```
#[
   Graph[
    Flatten[{a ⟷ b#, b# ⟷ c} & /@ Range[4]]]] & /@
 {Show, HamiltonianGraphQ, EulerianGraphQ, KVertexConnectedGraphQ[#, 2] &}
```



b. For G to be Hamiltonian, the vertices should be of 'sufficiently high' degree. For $\overline{G}$ to be Eulerian its vertices must all be of even degree; because G is not Eulerian all of its vertices then will be of odd degree, which implies that the order of G is even. Lastly, G cannot be complete or $\overline{G}$ will be disconnected. So take K_6 with C_6 removed:

```
Module[
 {g = EdgeDelete[CompleteGraph[6], EdgeList[CycleGraph[6]]]},
 Module[
  {h = GraphComplement[g]},
  {g, HamiltonianGraphQ[g], EulerianGraphQ[g], h, EulerianGraphQ[h]}]]
```

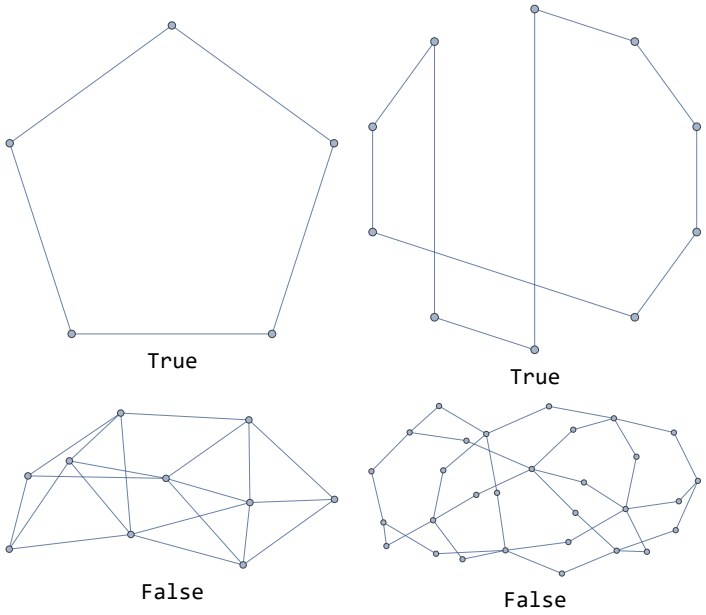


6.15

If the subdivision graph $G^*$of a graph G is Hamiltonian, then it has a cycle traversing all vertices of G^*, and this cycle certainly traverses the vertices added in the subdivision; which means that it traverses all the edges of G. Therefore, G is Eulerian. However, consider that while traversing these vertices corresponding to *all the edges* of G, the Hamiltonian cycle of G^* also traverses the vertices corresponding to the ones originally in G exactly once; this means that the degree of each vertex in G is exactly 2. That is to say, the theorem is true, but (rather trivially) it only applies cycles.

```
SubdivisionGraph[g_] :=
 Graph[
  Flatten[
   {#[[1]] ↔ w#, w# ↔ #[[2]]} & /@ EdgeList[g]]]
```

```
Grid[{
    Labeled[#, EulerianGraphQ[#]],
    Module[
     {sdg = SubdivisionGraph[#]},
     Labeled[sdg, HamiltonianGraphQ[sdg]]]
   } &
  /@ {
   CycleGraph[5],
   RandomGraph[{10, 20}]}]
```



6.16

a. If r is even, then G is Eulerian by Theorem 6.1. Conversely, since G is of even order, if r is odd then $\overline{G}$ is $\bar{r}$-regular with $\bar{r}$ even and therefore $\overline{G}$ is Eulerian.

b. Given r, for any pair u, v of nonadjacent vertices (in fact, for any pair of vertices at all) $\deg u + \deg v = 2r \geq |G|$ or $2r < |G|$. In the former case, G is Hamiltonian by Theorem 6.6. In the latter case, consider $\overline{G}$: the it is $\bar{r}$-regular with $\bar{r} = (|G| - 1) - r$ so for any pair u, v of vertices in $\overline{G}$,
$\deg u + \deg v = 2((|G| - 1) - r) = 2|G| - 2 - 2r > 2|G| - 2 - |G| = |G| - 2$ and since G is of even order $\deg u + \deg v \geq |G|$ and $\overline{G}$ is Hamiltonian.

6.17

$G(3)$ adds a large number of vertices as the order of G increases; this presents an upper limit to the order of G. In a given Hamiltonian cycle of $G(3)$ the best case requires that for each vertex of G, each of its two adjacent edges is incident with one

of the added vertices (and not with a different vertex of G). Since the added vertices are neighbors only of vertices of G, similarly each of them must itself be incident with two vertices of G. This implies that the number of added vertices can be at most equal to the order of G:

```
Reduce[Binomial[|G|, 3] ≤ |G| ∧ |G| ≥ 3, |G|, Integers] // TraditionalForm
```

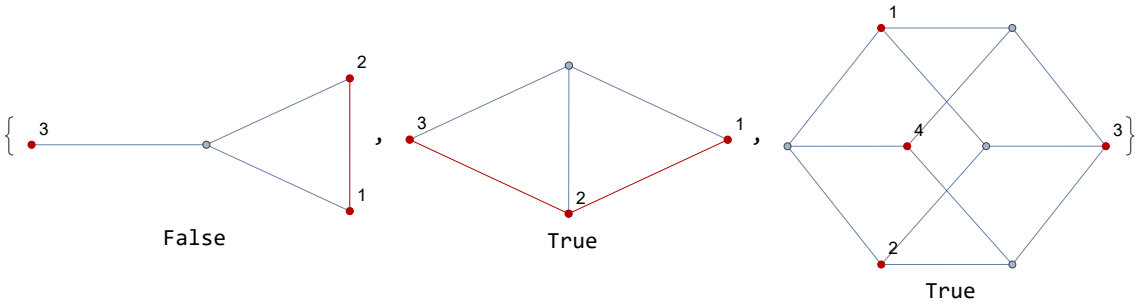
$|G| = 3 \vee |G| = 4$

This is a necessary but not sufficient condition.

```
G3[g_] :=
 HighlightGraph[ (* highlight the original graph;
  because the addend graph also contains the original vertices *)
  GraphUnion[g, (* union the original graph with the addend edges and vertices *)
   Graph[
    Flatten[
     Function[x, x ⟷ v#] /@ # &
      /@ Subsets[VertexList[g], {3}]]]],
  g, VertexLabels → "Name"]
```

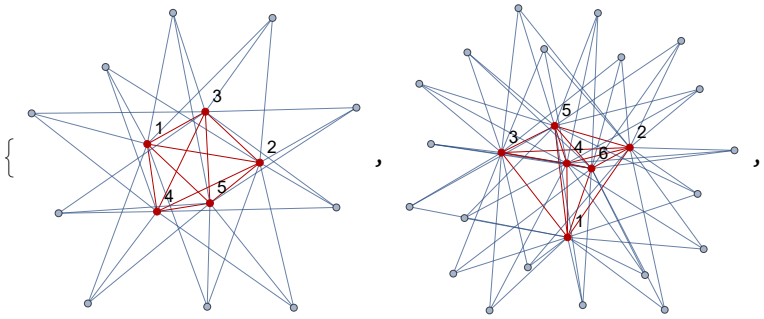
For $|G| = 3$, suppose G were disconnected; then $G(3)$ is not Hamiltonian. If G is connected, then $G(3)$ is Hamiltonian. For $|G| = 4$, since even $\overline{K}_4(3)$ is Hamiltonian, all graphs G of order 4 have $G(3)$ Hamiltonian:

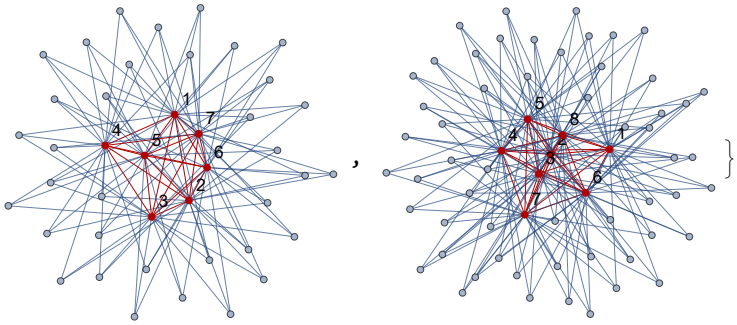
```
Module[
   {g3 = G3[#]},
   Labeled[g3, HamiltonianGraphQ[g3]]] & /@
 {Graph[Range[3], {1 ⟷ 2}], PathGraph[Range[3]], Graph[Range[4], {}]}
```



For illustration, even the maximally connected higher-order graphs do not have Hamiltonian $G(3)$:

```
Module[
   {g3 = G3[#]},
   Labeled[g3, HamiltonianGraphQ[g3]]] & /@
 CompleteGraph /@ Range[5, 8]
```





6.18

The bound of Corollary 7 is not 'sharp' in the sense that it is a necessary condition. Certainly for orders n of 3 and 4 the degree of each vertex must be at least $\frac{1}{2} n \geq 2$ for a cycle to exist and a graph can be Hamiltonian; but for higher orders n obviously a cycle (all vertices of degree 2, and thus less than $\frac{1}{2} n$) is Hamiltonian. The question is whether the statement of the Corollary that *all* graphs satisfying a looser vertex degree requirement must be Hamiltonian can have a lower bound; say $\forall\, v \in V(G) : \deg v \geq \frac{1}{2} |G| - 1$; or, can a non-Hamiltonian counterexample be found. Now, for any even n we can construct a disconnected graph as the union of two components $K_{\frac{1}{2}n}$ where the looser degree requirement holds and is obviously non-Hamiltonian. For an odd n we can construct a graph from one $K_{\frac{1}{2}n}$ and one $K_{\frac{1}{2}n-1}$, where each vertex in the latter is connected to some single vertex v in the former. Again, the looser degree requirement holds, and while this graph is not disconnected, v is a cut-vertex and so again this graph is non-Hamiltonian. Therefore the bound of the Corollary is indeed 'sharp'.

6.19

The degree of each G_1 vertex in the union is $\forall\, g \in V(G_1) : \deg g \geq \frac{1}{2} |G_1| + \frac{1}{2} |G_2| = \frac{1}{2} (|G_1| + |G_2|) = \frac{1}{2} |G|$; similarly for the degree of the G_2 vertices. Therefore Corollary 7 holds for G.

6.20

The order is at least 3 by axiom; obviously the graph is connected since it contains Hamiltonian paths; so remains to be shown only that the graph contains no cut-vertex. Suppose it did; then removing that vertex would separate the graph into

two components, contradicting the assumption that there exists a Hamiltonian path with that vertex initial. Therefore, the graph is 2-connected.

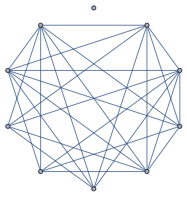
6.21

Suppose to the contrary that there is a graph G such that for all nonadjacent $u, v \in V(G)$: $\deg u + \deg v \geq |G| - 1$ that does *not* contain a Hamiltonian path; construct the join $G + K_1$ by adding a new vertex k and edges from it to every vertex of G. Any nonadjacent u, v in G are still nonadjacent in the join, and k is not nonadjacent to any vertex; thus it happens that for every nonadjacent u, v of the join $\deg_{G+K_1} u + \deg_{G+K_1} v \geq (|G| - 1) + 2 = |G| + 1 = |G + K_1|$, so by Theorem 6 the join is Hamiltonian. By assumption that G is not Hamiltonian it follows that every Hamiltonian cycle of the join must traverse k; but then G contains a Hamiltonian path, which is a contradiction.

6.22a

Let one vertex be disconnected (of degree zero); and consider the remaining graph of order 9 and size 28. K_9 is of size $\frac{1}{2}\ 9 \cdot 8 = 36$; so K_9 with eight arbitrary edges removed is of the required size. Since this graph is disconnected it is not Hamiltonian.

```
Module[
 {g = VertexAdd[ (* add one disconnected vertex *)
    EdgeDelete[ (* remove 8 arbitrary edges *)
     CompleteGraph[9],
     RandomSample[EdgeList[CompleteGraph[9]], 8]], (* 8 arbitrary edges *)
    10]},
 Labeled[
  Labeled[Show[g, ImageSize → Tiny],
   {VertexCount[g], EdgeCount[g]}],
  HamiltonianGraphQ[g]]]
```



```
{10, 28}
False
```

6.22b

The degrees a, b of the remaining two vertices can be deduced:

```
Reduce[
  1/2 (5 × 5 + 3 × 6 + a + b) == 28 ∧ (* degree sum must correspond to graph size *)
   0 ≤ a < 5 ∧ (* at least one vertex must have low enough degree not to trigger Theorem 6 *)
   0 ≤ b < 10 (* other degree must be meaningful *),
  {a, b}, Integers] // TraditionalForm
```

$a = 4 \land b = 9$

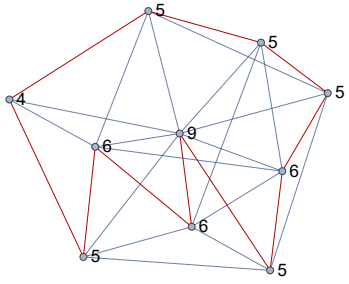
This is Hamiltonian by Theorem 11:

```
IsPosyHamiltonian[degrees_List] :=
 AllTrue[
  Range[⌊1/2 Length[degrees] - 1⌋],
  Length[Select[degrees, LessEqualThan[#]]] < # &]
```

```
IsPosyHamiltonian[{5, 5, 5, 5, 5, 6, 6, 6, 4, 9}]
```

```
True
```

```
Module[
 {g = RandomGraph[
    DegreeGraphDistribution[{5, 5, 5, 5, 5, 6, 6, 6, 4, 9}],
    VertexLabels → "VertexDegree",
    ImageSize → Small]},
 HighlightGraph[
  g,
  FindHamiltonianCycle[g]]]
```



6.23a

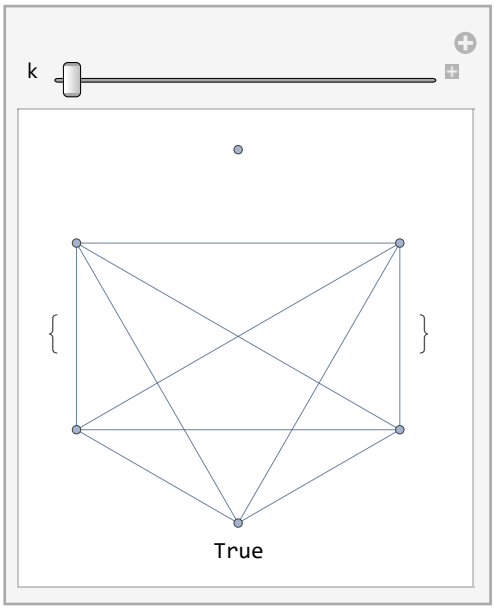
Since this is a generalization of 6.22 (where *k* was 5), try the same approach as 6.22a and see if we can construct a graph with the given properties in which one vertex is of degree zero and the remaining vertices are a complete graph with a certain number *l* edges removed:

```
Reduce[1/2 * (2 k - 1) * (2 k - 2) - l == k^2 + k - 2, {k, l}, Integers] // TraditionalForm
```

$c_1 \in \mathbb{Z} \land k = c_1 \land l = c_1^2 - 4\,c_1 + 3$

Then construct such a graph (which is obviously not Hamiltonian) and verify that the required properties hold:

```
Manipulate[
 Module[
  {h = Module[
     {K = CompleteGraph[2 k - 1]},
     VertexAdd[
      EdgeDelete[K,
       Take[EdgeList[K], 3 - 4 k + k^2]], (* remove l edges *)
      {2 k}]]}, (* add the single degree-0 vertex *)
  {Labeled[h, VertexCount[h] == 2 k ∧ EdgeCount[h] == k^2 + k - 2]}],
 {k, 3, 10, 1}]
```



6.23b

Again, deduce the degrees of the remaining two vertices:

```
Reduce[
  1/2 (k * k + (k - 2) (k + 1) + a + b) == k^2 + k - 2 ∧ (* degree sum must correspond to graph size *)
   0 ≤ a < k ∧ (* at least one vertex must have low enough degree not to trigger Theorem 6 *)
   0 ≤ b < 2 k (* other degree must be meaningful *),
  {a, b}, Integers] // TraditionalForm
```

$c_1 \in \mathbb{Z} \land c_1 \geq 0 \land k = c_1 + 1 \land a = c_1 \land b = 2\,c_1 + 1$

That is, the two remaining vertices have degree $c_1 = k - 1$ and $2\,c_1 + 1 = 2\,(k-1) + 1 = 2\,k - 1$. For the purposes of Theorem 11 we only need to consider vertices with degree less than $\frac{|G|}{2} = \frac{2k}{2} = k$, which for $k \geq 3$ is only the one with degree $k - 1$.

The number of vertices with degree at most $k - 1$ (that is, 1) is less than $k - 1$ if $1 < k - 1 \iff k > 2 \iff k \geq 3$, which is true by axiom; so the Theorem holds and G is Hamiltonian.

6.24a

Label the $k + 1$ vertices of degree 2 as l_i and the k others as r_i. Since a Hamiltonian cycle must traverse all l_i it must contain all $2\,(k+1)$ edges between them and the r_i. But since there are only k vertices r_i that means that some r_i must be traversed twice; which is a contradiction.

```
AllTrue[
 Range[100],
 ¬ HamiltonianGraphQ[
    EdgeAdd[
     RenameSubscriptGraph[CompleteGraph[#], r],
     (* construct r_i as K_k for maximal Hamiltonian opportunity *)
     Flatten[Table[{l_i ⟷ r_i, l_i ⟷ r_Mod[i+1,#,1]}, {i, # + 1}]]
     (* add l_i by distributing edges to r_i *)
    ]] &]
```

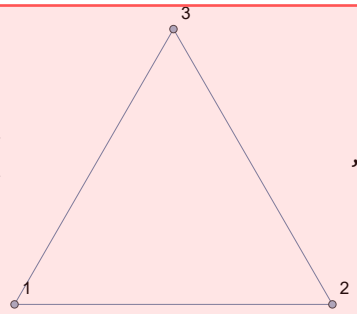
```
True
```

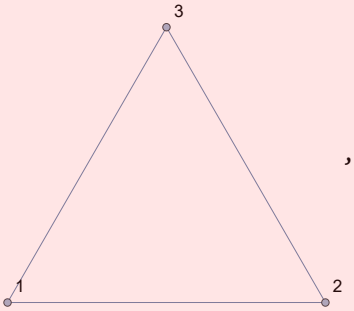
6.24b

```
Manipulate[
 Module[
  {g = EdgeAdd[
     RenameSubscriptGraph[CycleGraph[k, VertexLabels → "Name", ImageSize → Small], r],
     Flatten[Table[{l_i ⟷ r_i, l_i ⟷ r_Mod[i+1,k,1]}, {i, k}]]]},
  HighlightGraph[g, FindHamiltonianCycle[g]]],
 {k, 3, 20, 1}]
```

k

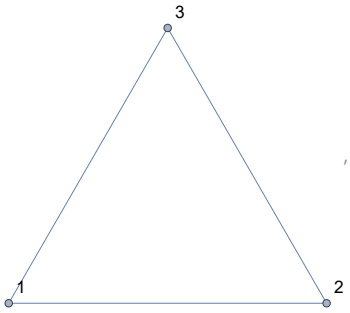
```
HighlightGraph[EdgeAdd[RenameSubscriptGraph[ , r],
  {l1 ⟷ r1, l1 ⟷ r2, l2 ⟷ r2, l2 ⟷ r3, l3 ⟷ r3, l3 ⟷ r1}],
 FindHamiltonianCycle[EdgeAdd[RenameSubscriptGraph[ , r],
  {l1 ⟷ r1, l1 ⟷ r2, l2 ⟷ r2, l2 ⟷ r3, l3 ⟷ r3, l3 ⟷ r1}]]]
```



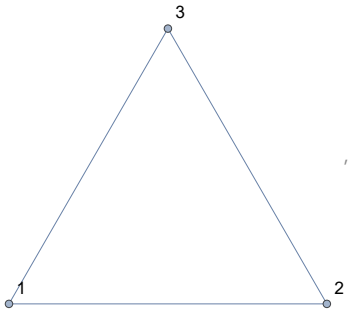


EdgeAdd: A graph object is expected at position 1 in

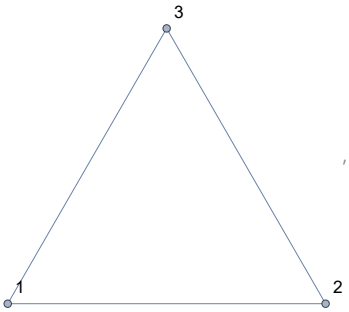
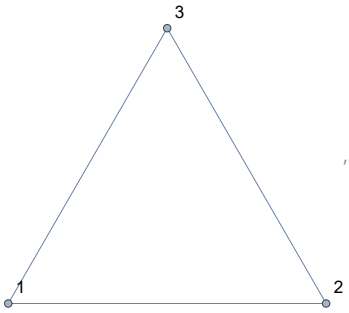
EdgeAdd[RenameSubscriptGraph[, r], {l1 ⟷ r1, l1 ⟷ r2, l2 ⟷ r2, l2 ⟷ r3, l3 ⟷ r3, l3 ⟷ r1}].



FindHamiltonianCycle: A graph object is expected at position 1 in

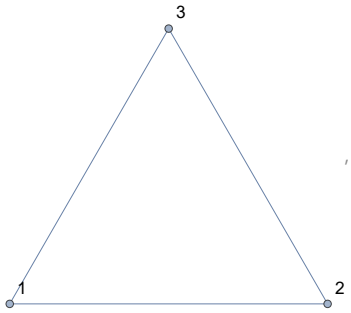
FindHamiltonianCycle[EdgeAdd[RenameSubscriptGraph[ , r], {l1 ⟼ r1, l1 ⟼ r2, l2 ⟼ r2, l2 ⟼ r3, l3 ⟼ r3, l3 ⟼ r1}]].

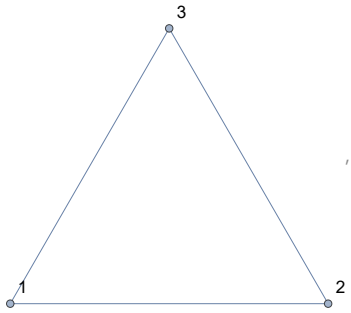
HighlightGraph: A graph object is expected at position 1 in

HighlightGraph[EdgeAdd[RenameSubscriptGraph[ , r], {l1 ⟼ r1, l1 ⟼ r2, l2 ⟼ r2, l2 ⟼ r3, l3 ⟼ r3, l3 ⟼ r1}], FindHamiltonianCycle[EdgeAdd[RenameSubscriptGraph[ , r], {l1 ⟼ r1, l1 ⟼ r2, l2 ⟼ r2, l2 ⟼ r3, l3 ⟼ r3, l3 ⟼ r1}]]].

HighlightGraph: A graph object is expected at position 1 in

HighlightGraph[EdgeAdd[RenameSubscriptGraph[, r], {l1 ⟼ r1, l1 ⟼ r2, l2 ⟼ r2, l2 ⟼ r3, l3 ⟼ r3, l3 ⟼ r1}], FindHamiltonianCycle[EdgeAdd[RenameSubscriptGraph[, r], {l1 ⟼ r1, l1 ⟼ r2, l2 ⟼ r2, l2 ⟼ r3, l3 ⟼ r3, l3 ⟼ r1}]]].





6.3 Exploration: Hamiltonian Walks

```
WalkLength[g_] := (* return operator that returns lengths of walks of the given graph *)
 Function[s, (* function taking vertex list *)
  Total[(* total distance of all subsequences *)
   GraphDistanceMatrix[g][[#[[1]]]][[#[[2]]]] & /@
    (* distance between the two vertices of a subsequence *)
    Subsequences[s, {2}]]](* for all subsequences of length 2 *)
```

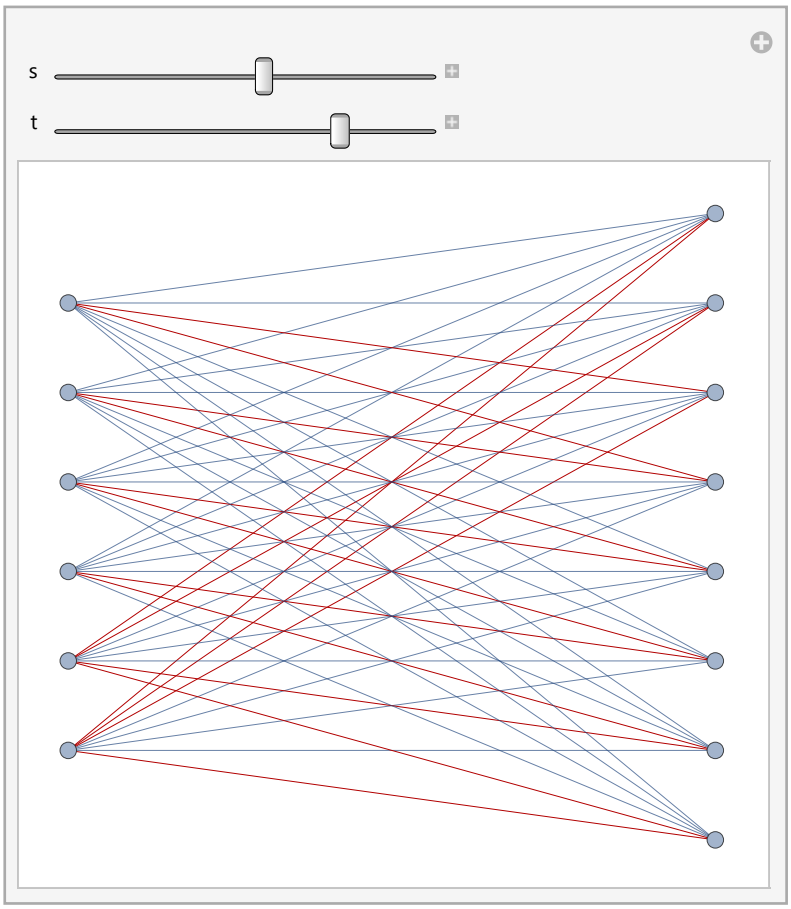
```
DisplayWalk[g_, w_List] := (* display a walk overlayed onto a graph *)
 Labeled[
  Show[
   g,
   Graphics[{
     Red, Dashed,
     Arrow[
      GraphEmbedding[g][[#]] & /@
       Subsequences[w, {2}]]}]],
  {w, WalkLength[g][w]}]
```

```
FindHamiltonianWalk[g_, o_: MinimalBy] :=
 First[
  o[
   Append[#, First[#]] & /@ (* close walk *)
    Permutations[VertexList[g]],
   (* for all possible permutations of the entire vertex list *)
   WalkLength[g],
   1 (* just return the first minimal walk *)]]
```

6.25

Since for all distinct $u, v \in K_n : d(u, v) = 1$, obviously $h(K_n) = h^+(K_n) = n$. For any $K_{s,t}$ a Hamiltonian walk of length $s + t$ can be constructed by alternating between the bipartite sets and picking any as-yet untraversed vertex from that set, so $h(K_{s,t}) = 2 \max\{s, t\}$:

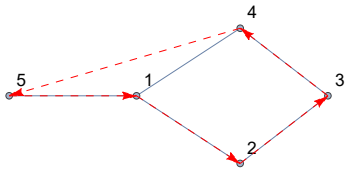
```
Manipulate[
 Module[
  {st = Max[{s, t}]},
  HighlightGraph[
   CompleteGraph[{s, t}],
   ReplacePart[
    Flatten[
     {Mod[#, s, 1] ↦ Mod[s + #, t, s + 1], Mod[s + #, t, s + 1] ↦ Mod[# + 1, s, 1]} & /@ Range[st]],
    2 st → (s + t) ↦ 1]]],
 {s, 1, 10, 1}, {t, 1, 10, 1}]
```



The maximum distance in $K_{s,t}$ is 2; a cyclic ordering of maximal length can be constructed by traversing all the vertices in one bipartite set followed by all the vertices in the other set, resulting in a path of length
$h^+(K_{s,t}) = 2(s-1) + 1 + 2(t-1) + 1 = 2(s+t-1)$.

6.26

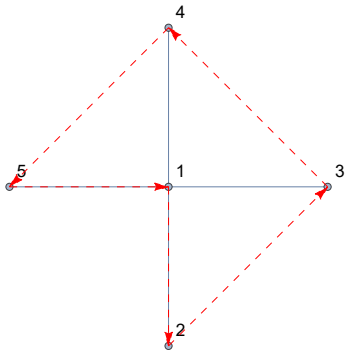
```
Module[
 {g = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 1, 1 ↔ 5}, VertexLabels → "Name", ImageSize → Small]},
 DisplayWalk[g, FindHamiltonianWalk[g]]]
```




```
{{1, 2, 3, 4, 5, 1}, 6}
```

6.27

```
Module[
 {g = StarGraph[5, VertexLabels → "Name", ImageSize → Small]},
 DisplayWalk[g, FindHamiltonianWalk[g]]]
```




```
{{1, 2, 3, 4, 5, 1}, 8}
```

6.28a

A walk of maximal length is constructed by sequentially traversing maximally distant (opposite) vertices:

```
Simplify[(1/2 n - 1) (1/2 n + 1/2 n - 1) + (1/2 n + 1)] // TraditionalForm
```

$$\frac{1}{2}\left(n^2-2\,n+4\right)$$

For example:

```
% /. n → 8
```

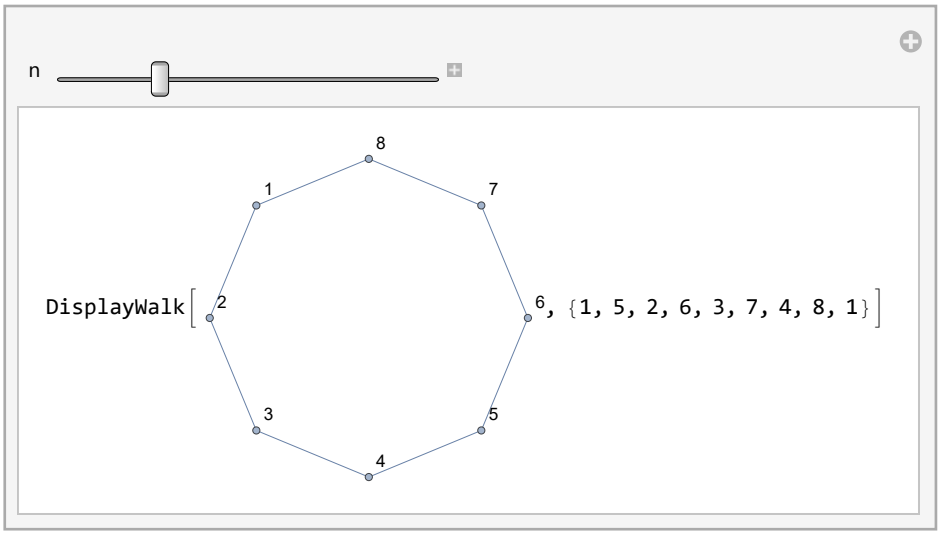
26

Verify by exhaustively searching for the length of a maximal walk:

```
Module[
 {g = CycleGraph[8]},
 WalkLength[g][
  FindHamiltonianWalk[g, MaximalBy]]]
```

26

```
Manipulate[
 Module[
  {graph = CycleGraph[n, VertexLabels → "Name", ImageSize → Small],
   walk = Append[Flatten[Transpose[Partition[Range[n], 1/2 n]]], 1]},
  DisplayWalk[graph, walk]],
 {{n, 8}, 4, 20, 2}]
```



6.28b

Similar approach as in a):

```
FullSimplify[n ⌊1/2 n⌋, Mod[n, 2] == 1 ∧ Element[n, Integers]] // TraditionalForm
```

$$n \left\lfloor \frac{n}{2} \right\rfloor$$

That didn't work. Be more explicit about *n* being odd:

```
Simplify[(2 m + 1) ⌊1/2 (2 m + 1)⌋, Element[m, Integers]] // TraditionalForm
```

$$m\,(2\,m+1)$$

```
% /. m → 1/2 (n - 1) // TraditionalForm
```

$$\frac{1}{2}(n-1)\,n$$

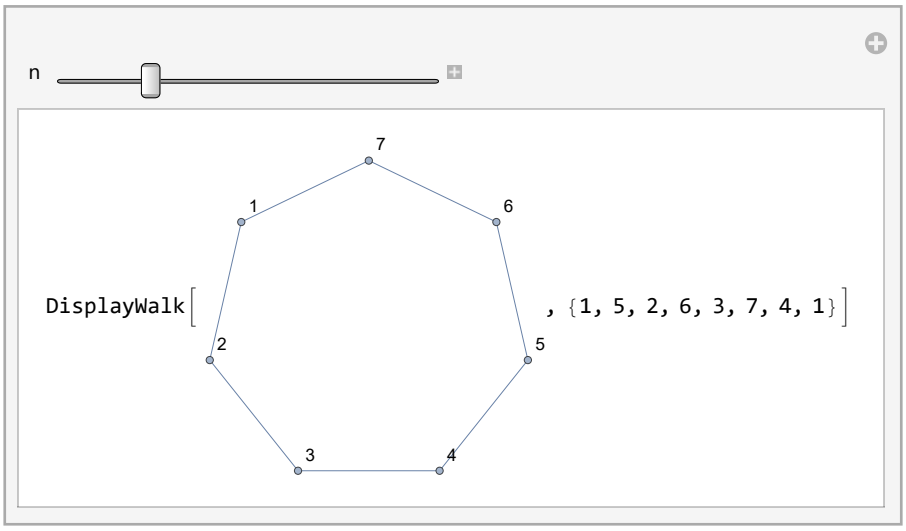
```
% /. n → 7 // TraditionalForm
```

21

```
Module[
 {g = CycleGraph[7]},
 WalkLength[g][
  FindHamiltonianWalk[g, MaximalBy]]]
```

21

```
Manipulate[
 Module[
  {graph = CycleGraph[n, VertexLabels → "Name", ImageSize → Small],
   walk = Flatten[Transpose[ArrayReshape[Range[n], {2, ⌈1/2 n⌉}, {1}]]]},
  DisplayWalk[graph, walk]],
 {{n, 7}, 3, 21, 2}]
```



6.29

a) Trivially from the definition of diameter as the maximum distance between any two vertices.

b) Obviously not; an example is h^+ from 6.28, where for $n > 2$: $\frac{1}{2}\left(n^2 - 2n + 4\right) < \frac{1}{2}n^2 = n \cdot \frac{1}{2}n = n \cdot \text{diam}(C_n)$.

6.30

The length of a minimal walk is equal to the length of a maximal walk if the distance between any two vertices is always equal; that is, equal to one: that is, the complete graphs.

```
AllTrue[ (* hypothesis holds for all sample graphs *)
 Select[ (* only connected graphs *)
  Flatten[Table[Table[(* generate random set of graphs *)
     RandomGraph[UniformGraphDistribution[n, m]],
     {m, n, 1/2 n (n - 1)}], {n, 2, 6}]],
  ConnectedGraphQ],
 Module[
   {wl = WalkLength[#]},
   (* minimal and maximal total distance of Hamiltonian walk equal *)
   (wl[FindHamiltonianWalk[#, MinimalBy]] == wl[FindHamiltonianWalk[#, MaximalBy]]) ==
    CompleteGraphQ[#] (* iff the graph is complete *)] &]
```

```
True
```

7 Digraphs

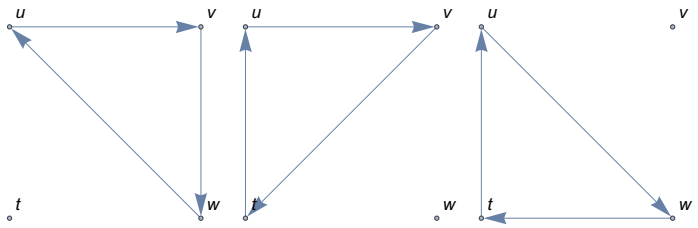
7.1 Strong Digraphs

▫ 7.1

a) $\forall\, u, v \in D : \exists\, w \in D, w \neq u, v$. Since $D - w$ is strong, it contains a u—v path as does D; therefore D is strong.

b) This follows because there is no single orientation of D that results in strong subgraphs. Let $t, u, v, w \in V(D)$ be the vertices of D, and suppose that all four subgraphs are strong. Now for $D - t$ to be strong, it must form a directed cycle; without loss of generality, pick one orientation. This then also fixes the orientation in the cycle $D - w$, which in turn fixes the orientation of $D - v$:

```
GraphicsRow[
 Graph[
    {t, u, v, w}, #,
    VertexCoordinates → {{-1, -1}, {-1, +1}, {+1, +1}, {+1, -1}}, VertexLabels → "Name"] & /@
  {{u ↔ v, v ↔ w, w ↔ u}, {u ↔ v, v ↔ t, t ↔ u}, {t ↔ u, u ↔ w, w ↔ t}},
 ImageSize → Medium]
```



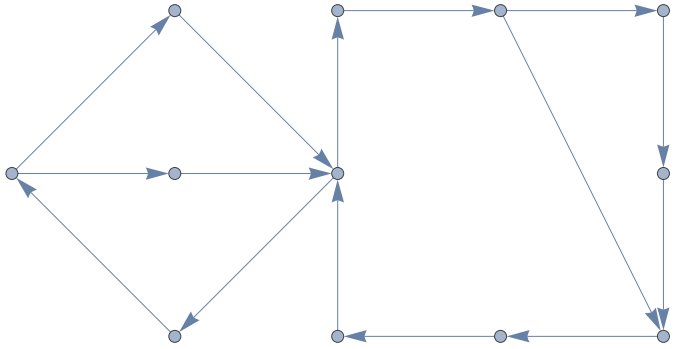
But this results in two opposing directions for the u—w edge, which contradicts the assumption that D is oriented.

7.2

Note that “Eulerian orientation” is not actually defined in the text; but from Wikipedia and Theorem 4 we can conclude it to mean: an orientation of a graph resulting in a directed graph that is Eulerian. Then, a Eulerian directed graph has an underlying graph that is also Eulerian by the same Eulerian circuit. Also, an orientation can be defined on a Eulerian graph by its Eulerian circuit so that this orientation results in a Eulerian directed graph.

7.3

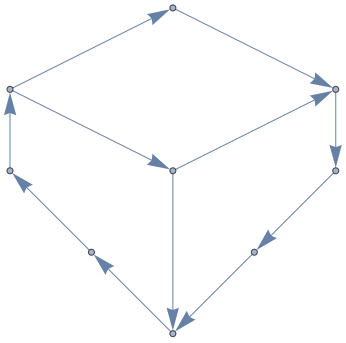
```
Graph[
 Range[12],
 {1 ↔ 2, 2 ↔ 5, 1 ↔ 3, 3 ↔ 5, 5 ↔ 4, 4 ↔ 1, 5 ↔ 6,
  6 ↔ 7, 7 ↔ 8, 8 ↔ 9, 9 ↔ 10, 7 ↔ 10, 10 ↔ 11, 11 ↔ 12, 12 ↔ 5},
 VertexCoordinates → {{-2, 0}, {-1, +1}, {-1, 0}, {-1, -1}, {0, 0},
   {0, +1}, {+1, +1}, {+2, +1}, {+2, 0}, {+2, -1}, {+1, -1}, {0, -1}}]
```



```
ConnectedGraphQ[%]
```

```
True
```

```
Graph[
 Range[9],
 {1 ↔ 2, 1 ↔ 3, 2 ↔ 4, 3 ↔ 4, 4 ↔ 5, 5 ↔ 6, 6 ↔ 7, 3 ↔ 7, 7 ↔ 8, 8 ↔ 9, 9 ↔ 1},
 VertexCoordinates →
  {{-2, 0}, {0, +1}, {0, -1}, {+2, 0}, {+2, -1}, {+1, -2}, {0, -3}, {-1, -2}, {-2, -1}},
 ImageSize → Small]
```



```
ConnectedGraphQ[%]
```

```
True
```

7.4

Trivially: suppose D is strongly connected. Now for every u, v in $\vec{D}$, there is a v—u path in D that corresponds to a u—v path in $\vec{D}$; therefore, $\vec{D}$ is strongly connected. Mutatis mutandis for the converse.

7.5

(⇒) Suppose that G is strongly connected. Then for any edge cut separating $V(G)$ into A, B, for any $u \in A$, $v \in B$ there is a u—v path in G; this path must contain some adjacent $a \in A$, $b \in B$.

(⇐) Suppose that for every edge-cut separating the vertex set into A and B there is an arc from A to B. Now take any u, $v \in G$ and show there is a directed path from u to v. Iteratively: start with $A = \{u\}$, $B = V(G) \backslash A$. There is an arc from u to a vertex $u_1 \in B$. If $u_1 = v$ then we are done; otherwise, consider $A_1 = \{u, u_1\}$, $B_1 = V(G) \backslash A_1$. Again, there is an arc from some vertex of A_1 to a $u_2 \in B_1$. Continue this process until we have $A_n = \{u, u_1, \ldots, u_n\}$ and $u_{n+1} = v$; then $(u, u_1, \ldots, u_n, v)$ is a u—v path. The n must exist because $v \in B$ and $\left|V(G)\right|$ is finite. Since there is a directed path for each pair of vertices, the graph is strongly connected.

7.6

Recall that in a directed graph, the presence of an arc between two vertices doesn't preclude the existence of the reverse arc. Therefore we can solve for:

```
Reduce[n id == ∑_{i=0}^{n-1} i, {id}] // TraditionalForm
```

$$\text{id} = \frac{n}{2} - \frac{1}{2} \vee n = 0$$

For example, for $n = 3$: id = $\frac{3}{2} - \frac{1}{2} = 1$. In the following graph each vertex has indegree 1 but different outdegree:

```
Graph[Range[3], {2 ↔ 3, 3 ↔ 1, 3 ↔ 2}, ImageSize → Small, VertexLabels → "VertexOutDegree"]
```

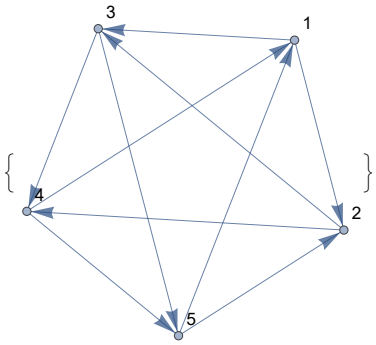
7.2 Tournaments

```
Tournaments[n_] := (* generate all tournaments of given order, up to isomorphism *)
 DeleteDuplicates[(* remove isomorphic duplicates *)
  Module[
   {el = EdgeList[CompleteGraph[n]]},
   (* generate the (undirected) edge list of a complete graph *)
   Function[ei, (* apply an orientation *)
     Graph[
      MapIndexed[(* orient the edge list *)
       DirectedEdge @@
         If[BitGet[ei, First[#2] - 1] == 0, #1, Reverse[#1]] &,
       el],
      VertexLabels → "Name"]
    ] /@ Range[0, 2^(Length[el]-1)]] (* all possible orientations *),
  IsomorphicGraphQ]
```

7.7

Brute force:

```
Select[
 Flatten[Tournaments /@ Range[3, 5]], (* all tournaments of order 3 through 5 *)
 Function[g, (* return whether all single-edge reorientations are Hamiltonian *)
  AllTrue[
   EdgeAdd[EdgeDelete[g, #], Reverse[#]] & /@ EdgeList[g],
   HamiltonianGraphQ]]]
```



7.8

If the outdegree of every vertex in a tournament of order n is x, then the indegree is $(n-1)-x$. By the 'First Theorem',

```
Reduce[Sum[x, {i, 0, n-1}] == Sum[(n - 1) - x, {i, 0, n-1}] ∧ n > 0, {x}, Integers] // TraditionalForm
```

$c_1 \in \mathbb{Z} \land c_1 \geq 0 \land n = 2\,c_1 + 1 \land x = c_1$

```
(n - 1) - x /. n → 1 + 2 x // TraditionalForm
```

x

7.9

(⇒) By Theorem 8 there is a Hamiltonian path $(v_0, \ldots, v_{n-1})$. Since T is transitive, $\forall\, i, j : i < j$ there is an i—j arc, and this defines the orientation of all the arcs. Therefore od $v_i = (n - 1) - i$, which are unique.

(⇐) The outdegree sequence must therefore be $\{n - 1, \ldots, 0\}$. Pick the vertex with highest outdegree; this is incident with all other vertices. From the remaining $n - 1$ vertices pick the one with highest outdegree $n - 2$; this again is incident with all other vertices. Continue until the last vertex with zero outdegree remains. The sequence of vertices so chosen is a Hamiltonian path and all arcs point 'in the direction of' the path, showing that the tournament is transitive.

7.10

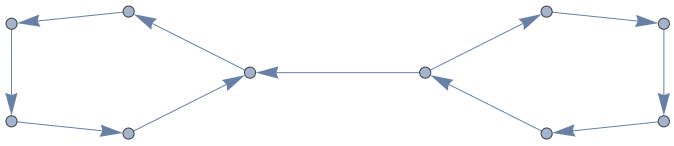
Let $d(u, v) = k$ and a shortest path between the vertices be $(u, v_1, \ldots, v_{k-1} = v)$. Since this is a shortest path all u—v arcs are incident *to* u, and id $u \geq k - 1$.

7.11

By the 'First Theorem' $\sum_i$ id $v_i = \sum_i$ od v_i, so id $v_{n-1} + \sum_{i<n-1}$ id v_i = od $v_{n-1} + \sum_{i<n-1}$ od $v_i \geq$ od $v_{n-1} + (n - 1) + \sum_{i<n-1}$ id v_i and id $v_{n-1} \geq$ od $v_{n-1} + (n - 1)$. Then it must be that id $v_{n-1} = n - 1$ and od $v_{n-1} = 0$. Since nothing is reachable from v_{n-1}, the tournament is not strongly connected.

7.12a

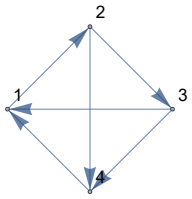
```
EdgeAdd[
 GraphUnion[
  RenameSubscriptGraph[CycleGraph[5, DirectedEdges → True], outer],
  RenameSubscriptGraph[CycleGraph[5, DirectedEdges → True], inner]],
 outer1 ↔ inner1]
```



Clearly every vertex lies on a cycle, yet the graph is not strongly connected.

7.12b

```
Graph[
 {1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 1, 3 ↔ 1, 2 ↔ 4},
 VertexCoordinates -> {{-1, 0}, {0, +1}, {+1, 0}, {0, -1}},
 VertexLabels → "Name", ImageSize → Tiny]
```



Even though the tournament is strongly connected, there is no Hamiltonian 1—3 nor 3—1 path.

7.12c

In the above graph, $3 \longleftrightarrow 1$ is an arc but does not lie on a Hamiltonian cycle.

7.13

Suppose there are u—v and v—u paths of minimal and equal length. Since either $u \longleftrightarrow v$ or $v \longleftrightarrow u$, $d(u, v) = d(v, u) = 1$, which is impossible.

7.14

By the 'First Theorem' again, $\sum_i^n \text{od}\, v_i = \sum_i^n \text{id}\, v_i \Rightarrow n \cdot x = n \cdot ((n-1) - x) \Rightarrow x = (n-1) - x \Rightarrow 2x = n - 1 \Rightarrow x = \frac{1}{2}(n-1)$. This has a solution for odd and not for even n.

7.15

By induction. For any strongly connected tournament T_n of order n, T_n is Hamiltonian and contains a cycle of order n. Furthermore, by Theorem 11 it contains a strongly connected subtournament $T_{n-1} = T_n - v$ of order $n - 1$.

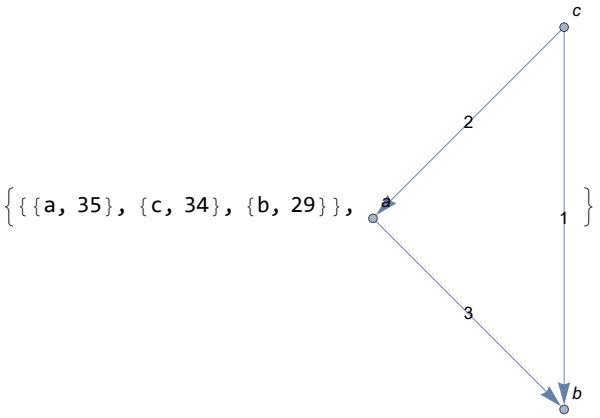
7.3 Decision-Making

```
ElectionFirstChoice[l_] := (* count first choices *)
 SortBy[
  {#[[1, 1]], Total[#[[All, 2]]]} & /@
   GatherBy[
    {#[[2]], First[#]} & /@ l,
    First],
  -Last[#] &]
```

```
ElectionTotalOrdering[l_] := (* count total ordering *)
 Module[
  {t =
    Module[
       {total = Total[#[[All, 2]]]},
       If[total ≥ 0, (* *** ignoring zero case; should remove edge *)
        {#[[1, 1]], total},
        {Reverse[#[[1, 1]]], -total}]] & /@
     GatherBy[
      Flatten[
       Function[e,
         If[
            Hash[#[[1]]] < Hash[#[[2]]],
            {#, First[e]},
            {Reverse[#], -First[e]}] & /@
          Subsequences[Rest[e], {2}]] /@
        l,
       1],
      First]},
  Graph[
   DirectedEdge @@@ t[[All, 1]],
   EdgeWeight → t[[All, 2]],
   VertexLabels → "Name", EdgeLabels → "EdgeWeight", ImageSize → Small]]
```

7.16

```
Through[{ElectionFirstChoice, ElectionTotalOrdering}[
  {{18, a, b, c},
   {17, a, c, b},
   {13, b, a, c},
   {16, b, c, a},
   {16, c, a, b},
   {18, c, b, a}}]]
```



Considering first choices only, a should win; but considering the total ordering, c should.

7.17

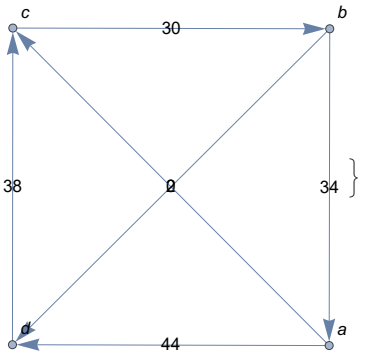
```
ElectionRunoff[l_, cases_] :=
 ElectionFirstChoice[
  DeleteCases[(* remove the last-choice winner *)
    cases] /@
   l]
```

```
ElectionRunoffNoLast[l_] :=
 ElectionRunoff[l,
  ElectionFirstChoice[l][[-1]][[1]]]
```

```
ElectionRunoffNoLastTwo[l_] :=
 ElectionRunoff[l,
  ElectionFirstChoice[l][[-2]][[1]] | ElectionFirstChoice[l][[-1]][[1]]
  (* *** shouldn't use patterns but equality *)
 ]
```

```
Through[
 {ElectionFirstChoice, ElectionRunoffNoLast, ElectionRunoffNoLastTwo, ElectionTotalOrdering}[{
   {12, a, b, c, d},
   {11, a, b, d, c},
   {28, a, c, b, d},
   {10, a, c, d, b},
   {27, a, d, b, c},
   {26, a, d, c, b},
   {11, b, a, c, d},
   {10, b, a, d, c},
   {25, b, c, a, d},
   {9, b, c, d, a},
   {24, b, d, a, c},
   {29, b, d, c, a},
   {10, c, a, d, b},
   {9, c, a, d, b},
   {22, c, b, a, d},
   {12, c, b, d, a},
   {21, c, d, a, b},
   {20, c, d, b, a},
   {11, d, a, b, c},
   {8, d, a, c, b},
   {21, d, b, a, c},
   {7, d, b, c, a},
   {20, d, c, a, b},
   {25, d, c, b, a}
  }]]
```

```
{{{a, 114}, {b, 108}, {c, 94}, {d, 92}},
 {{c, 139}, {b, 136}, {a, 133}}, {{b, 215}, {a, 193}},
```



The college tournament of paired comparisons is not transitive; although one might argue that even in this case, a is the winner because it has the highest outdegree. Who *should* win is a subjective question; if voters are already being troubled to express a ranked preference, I would suggest that some method of awarding points based on order retains the most information in the result. This, however, has no connection to graph theory.

7.18

The outcome can be derived from the family tournament in Figure 7.18; H→F, T→GM, C→H, T→C. This is one way of resolving the cycle in the tournament, but the outcome is dependent on the arbitrary order in which the resolution occurs. The outcome

in this particular case (C) happens to coincide with Edwin's preference.

7.4 Wine Bottle Problems

As an aside, while 'wine bottle problems' appeared to have some similarity to groups, I don't find a way to model them as such: if we consider the group elements as 'bottle states' or configurations and 'pourings' as the operations, then there is no identity element that is unmoved by any operation; if we approach it as a permutation group and view the elements themselves as the pourings and the operation as composition, then some operations are undefined (for example, the same pouring cannot be applied twice in a row).

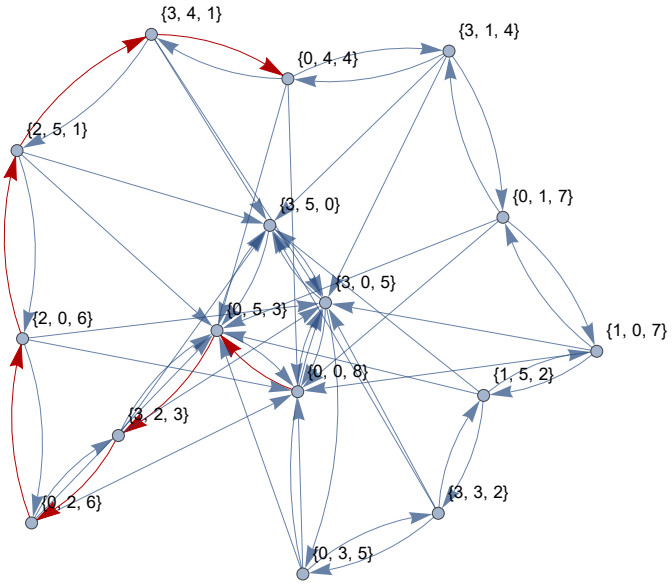
```
WineBottlePour[i_, j_, B_, BM_] :=
 Module[
  {poured = Min[B[[i]], BM[[j]] - B[[j]]]}, (* amount that will be poured *)
  ReplacePart[B, {i → B[[i]] - poured, j → B[[j]] + poured}]]
```

```
WineBottleOperate[el_List, m_, v_] := (* from the given established list of edges,
 maximum bottle capacity, and starting vertex, return the recursively reachable edges *)
 Union @@ (
   Module[
      {e = v ↔ Pour[#[[1]], #[[2]], v, m]}, (* graph edge created by the pouring *)
      If[Equal @@ e ∨ MemberQ[el, e], (* self-edge or edge already found *)
       el, (* terminate recurstion *)
       If[FreeQ[el, e[[2]]], (* destination configuration not already found? *)
        WineBottleOperate[Append[el, e], m, e[[2]]],
        (* recursively determine reachable configurations *)
        Append[el, e](* otherwise just add the new transition
         to the existing configuration *)
       ]]] & /@
    Select[(* all possible pourings between two distinct bottles *)
     Tuples[Range[Length[m]], 2],
     Apply[Unequal]])
```

7.19a

Seems like the smallest number of pourings is 1 + 7 = 8; one to fill the 8-liter bottle, then the 7 pourings shown:

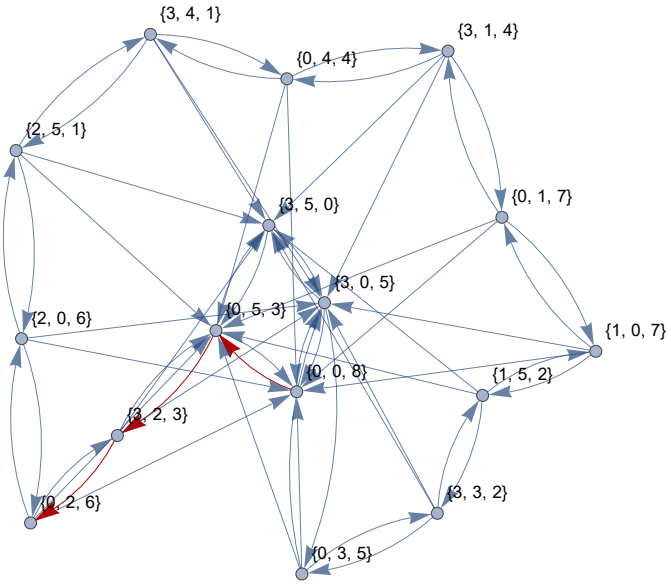
```
Module[
 {g = Graph[
    WineBottleOperate[{}, {3, 5, 8}, {0, 0, 8}],
    VertexLabels → "Name"]},
 HighlightGraph[
  g,
  DirectedEdge @@@ Subsequences[
    FindShortestPath[g, {0, 0, 8}, {0, 4, 4}],
    {2}]]]
```



7.19b

Four; the initial, and:

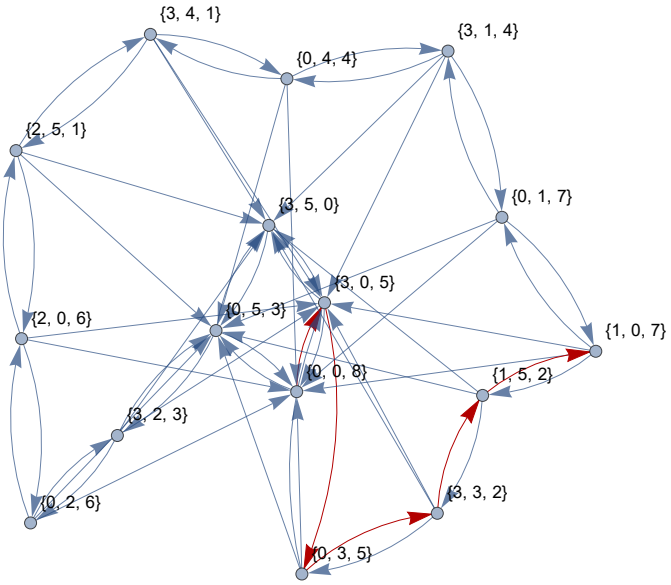
```
Module[
 {g = Graph[
    WineBottleOperate[{}, {3, 5, 8}, {0, 0, 8}],
    VertexLabels → "Name"]},
 HighlightGraph[
  g,
  DirectedEdge @@@ Subsequences[
    FindShortestPath[g, {0, 0, 8}, {0, 2, 6}],
    {2}]]]
```



7.19c

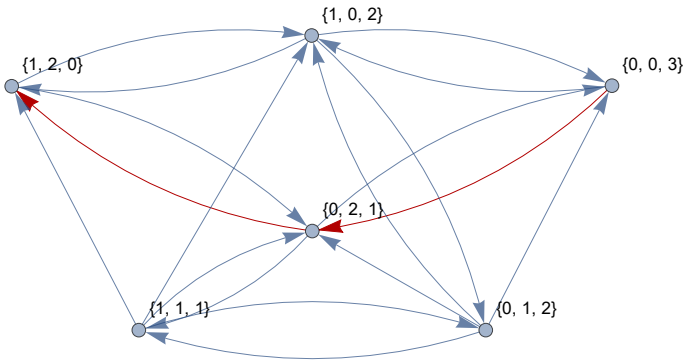
Six; the initial, and:

```
Module[
 {g = Graph[
    WineBottleOperate[{}, {3, 5, 8}, {0, 0, 8}],
    VertexLabels → "Name"]},
 HighlightGraph[
  g,
  DirectedEdge @@@ Subsequences[
    FindShortestPath[g, {0, 0, 8}, {1, 0, 7}],
    {2}]]]
```



7.20

```
Module[
 {g = Graph[
    WineBottleOperate[{}, {1, 2, 9}, {0, 0, 3}],
    VertexLabels → "Name"]},
 HighlightGraph[
  g,
  DirectedEdge @@@ Subsequences[
    FindShortestPath[g, {0, 0, 3}, {1, 2, 0}],
    {2}]]]
```



8 Matchings and Factorizations

```
ConstructBipartiteGraph[edges_List] :=
 Module[
  {top = First /@ edges,
   bottom = Sort[DeleteDuplicates[Flatten[Rest /@ edges]]]},
  Graph[
   Join[top, bottom],
   Flatten[
    Thread[
       Function[{x}, UndirectedEdge[First[#], x]][Rest[#]] & /@ edges],
   VertexLabels → "Name",
   VertexCoordinates → Join[
     Table[{i - 1/2 Length[top], +1}, {i, 0, Length[top] - 1}],
     Table[{i - 1/2 Length[bottom], -1}, {i, 0, Length[bottom] - 1}]]]]
```

8.1 Matchings

Theorem 8.7

$\alpha' G + b' G = |G|$

(≤) A maximal cover of $\alpha' G$ edges coincides with $2\,\alpha' G$ vertices; the remaining $|G| - 2\,\alpha' G$ vertices can be covered by as many edges, so a minimal cover has no more than $\beta' G \le \alpha' G + (|G| - 2\,\alpha' G) = |G| - \alpha' G$ edges, and $\alpha' G + \beta' G \le |G|$.

(≥) A minimal edge cover of G induces a subgraph F with size $\beta' G$. Because F is minimal, it contains no paths of length 3 or more (or cycles); in other words, F consists of components which are stars (or trees). Thus a matching of $|G| - \beta' G$ edges can be generated by picking one edge from each component, and $\alpha' G \ge |G| - \beta' G$ $\Rightarrow \alpha' G + \beta' G \ge |G|$.

A forest of order n and size $n - k$ has k components. The size of F is $\beta' G = |F| - |F| + \beta' G = |F| - (|F| - \beta' G)$, so F has $|F| - \beta' G = |G| - \beta' G$ components.

8.1

```
Reduce[
  Between[EdgeCount[CompleteGraph[n]], {0, 6}] ∧ n > 1 (* nontrivial *),
  {n}, Integers] // TraditionalForm(* u₀ *)
```

$n = 2 \vee n = 3 \vee n = 4$

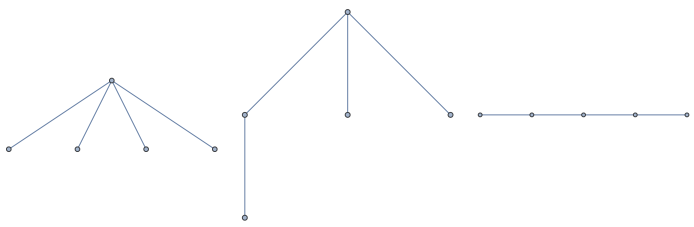
```
EdgeCount[CompleteGraph[n]] /. n → {2, 3, 4} // TraditionalForm
```

$\{1, 3, 6\}$

```
;
```

The number of distinct paths in a tree is always 1 (u_1). A transitive tournament has no cycles (u_2). A tree of order 6 has size 5; every edge in a tree is a bridge, so a tree of order 6 has 5 bridges (u_3). By Theorem 2.6 an r-regular graph of order 7 exists iff r is even; nonempty implies $r > 0$ (u_4). A tree of order 5 has 4 edges, so the degree cannot be larger than 4; a tree is connected so the maximum degree must be at least 2; the following are examples of trees of all three possibilities (u_5):

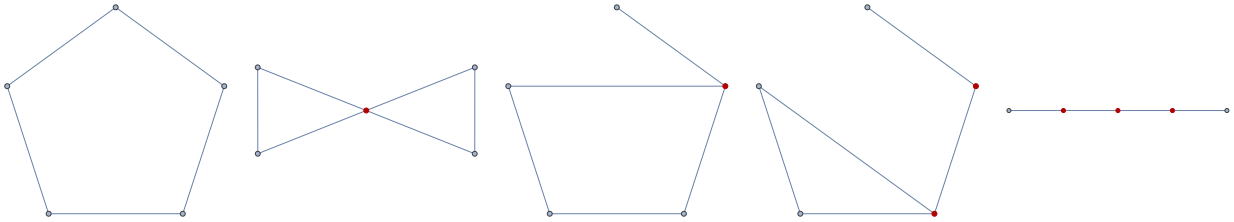
```
GraphicsRow[
 RandomGraph[DegreeGraphDistribution[#]] & /@ {{4, 1, 1, 1, 1}, {3, 2, 1, 1, 1}, {2, 2, 2, 1, 1}}]
```



(u_6) Suppose all vertices in a graph of order 5 are cut-vertices: then none of those vertices can be end-vertices, which is to say, all vertices have degree at least 2; which means that the graph is connected and contains either C_5 or a 'bowtie' constructed

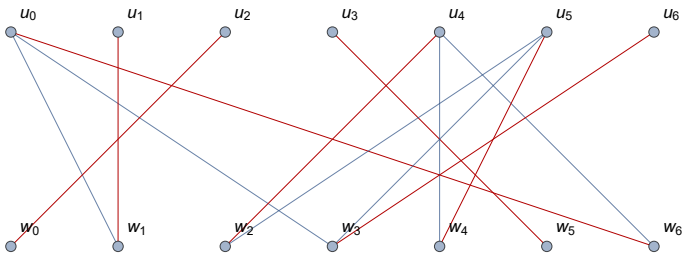
from two abutting C_3s; and only the latter has a single cut-vertex. Now suppose that only 4 of the vertices are cut-vertices: again, those four have to be of degree at least 2; one of those could be a cut-vertex if it is adjacent to the remaining end vertex; and there are only two possibilities for the putative cut-vertices to be connected, one of which produces a configuration with just two cut-vertices. Now suppose that only 3 of the vertices are cut-vertices: P_5 is such a graph.

```
GraphicsRow[{
  CycleGraph[5],
  HighlightGraph[Graph[{c ↔ 1, 1 ↔ 2, 2 ↔ c, c ↔ 3, 3 ↔ 4, 4 ↔ c}], {c}],
  HighlightGraph[EdgeAdd[CycleGraph[4], {4 ↔ 5}], {4}],
  HighlightGraph[EdgeAdd[CycleGraph[3], {3 ↔ 4, 4 ↔ 5}], {3, 4}],
  HighlightGraph[PathGraph[Range[5]], {2, 3, 4}]},
 ImageSize → Full]
```



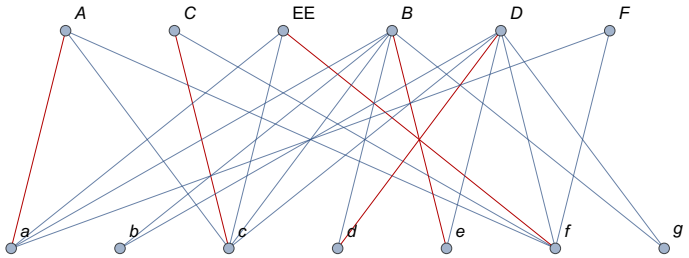
The resultant graph is shown, with a perfect matching highlighted:

```
Module[
 {g = ConstructBipartiteGraph[{
      {u0, w1, w3, w6},
      {u1, w1},
      {u2, w0},
      {u3, w5},
      {u4, w2, w4, w6},
      {u5, w2, w3, w4},
      {u6, w3}}]},
 HighlightGraph[g, FindIndependentEdgeSet[g]]]
```



8.2

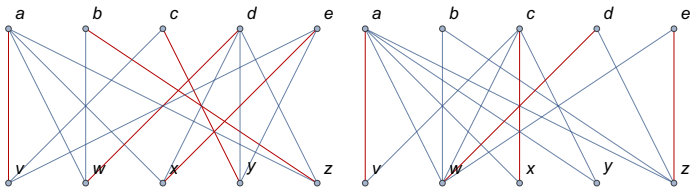
```
Module[
 {g = ConstructBipartiteGraph[{
      {A, a, c, f},
      {C, c, f},
      {EE, a, c, f},
      {B, a, b, c, d, e, g},
      {D, b, c, d, e, f, g},
      {F, a, f}}]},
 HighlightGraph[g, FindIndependentEdgeSet[g]]]
```



A perfect matching is not possible; specifically, $N\{A, C, E, F\} = \{a, c, f\}$ and so Hall's condition is not satisfied.

8.3

```
GraphicsRow[
 Module[
    {g = ConstructBipartiteGraph[#]},
    HighlightGraph[g, FindIndependentEdgeSet[g]]] & /@ {
   {
    {a, v, w, x, z},
    {b, w, z},
    {c, v, y},
    {d, w, x, y, z},
    {e, v, x, y}},
   {
    {a, v, w, x, y, z},
    {b, w, z},
    {c, v, w, x, y},
    {d, w, z},
    {e, w, z}}}]
```



There is a perfect matching for G_1, but not for G_2: $N\{b, d, e\} = \{w, z\}$ and so Hall's condition is not satisfied.

8.4

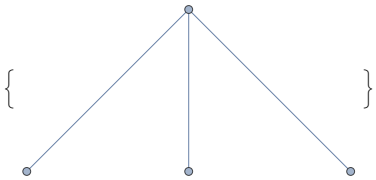
For any subset $X \subseteq U$ each of the vertices of X have distinct degree. Since G is connected, the degree is at least 1, and so the maximal degree in X is at least $|X|$, and $|N(X)| \geq |X|$, satisfying Hall's condition.

8.5

A tree of order $2k$ is of size $2k - 1$. Assuming the tree has a perfect matching of order k, the other perfect matching would have to consist only of the remaining edges, of which there are $(2k - 1) - k = k - 1$. Therefore a tree can have at most one perfect matching.

8.6a

```
Select[(* all without perfect matchings *)
 DeleteDuplicates[(* remove isomorphisms *)
  Select[(* only connected graphs *)
   Graph[Range[4], #] & /@
    Subsets[EdgeList[CompleteGraph[4]]],
   (* all permutations of edges of an order-4 graph *)
   ConnectedGraphQ],
  IsomorphicGraphQ],
 Length[FindIndependentEdgeSet[#]] < 2 &]
```



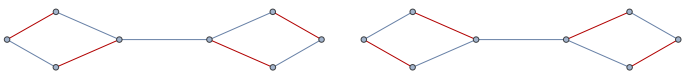
8.6b

Assume the contrary of a graph G of even order $|G|$ that does not have a perfect matching; then $\alpha' < \frac{1}{2}|G|$ and $\beta' = |G| - \alpha' > \frac{1}{2}|G|$. Pick one of the edges of *****

The Wikipedia entry on claw-free graphs gives an outline of the original proof by Sumner (1974).

8.7

```
GraphicsRow[
 Module[
  {g = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 1, 3 ↔ 5, 5 ↔ 6, 6 ↔ 7, 7 ↔ 8, 8 ↔ 5}]},
  Labeled[
     HighlightGraph[g, #],
     IndependentEdgeSetQ[g, #]] & /@ {
    {1 ↔ 2, 3 ↔ 4, 5 ↔ 6, 7 ↔ 8},
    {2 ↔ 3, 4 ↔ 1, 6 ↔ 7, 8 ↔ 5}}]]
```



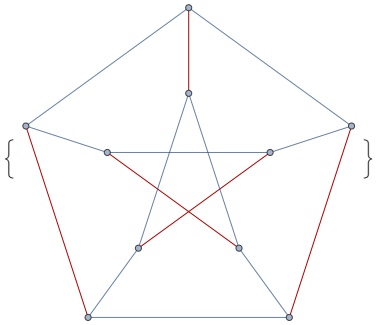
8.8

Begin by characterizing all perfect matchings of the Petersen graph by the number of spokes used, and then counting how many edges remain available in the outer cycle (OC) and the inner cycle (IC). Note that P_5 has 10 vertices and thus needs 5 independent edges to form a perfect matching:

- 0 spokes: this leaves 2 edges in IC and 2 in OC: impossible
- 1 spoke: this leaves 2 edges in IC and 2 in OC, so 1 possibility (up to isomorphism)
- 2 spokes: the spokes are adjacent either in OC or in IC, so without loss of generality assume they are adjacent in OC: this leaves only 1 edge in IC and 1 edge in OC: impossible
- 3 spokes: similarly, without loss of generality assume the spokes are adjacent in OC: this leaves 0 edges in IC and 1 in OC: impossible
- 4 spokes: this leaves 0 edges in IC and OC: impossible
- 5 spokes: 1 possibility (up to isomorphism)

None of the perfect matchings with 1 spoke are disjoint: a rotation leaves either intersecting edges on the OC or the IC. Obviously the perfect matchings with 1 spoke are not disjoint with the one with 5 spokes. So therefore the Petersen graph does not have disjoint perfect matchings.

```
Module[
 {g = PetersenGraph[5, 2]},
 HighlightGraph[g, #] & /@
  DeleteDuplicates[ (* all disjoint perfect matchings *)
   Select[(* all perfect matchings *)
    Subsets[EdgeList[g], {5}], (* all picks of 5 edges *)
    IndependentEdgeSetQ[g, #] &], (* picks that form a matching *)
   IntersectingQ]]
```



8.9

In other words, if $\alpha_{G_i} = i \wedge \alpha_{G_i} + \alpha'_{G_i} = 5 \Rightarrow \alpha_{G_i} = i \wedge \alpha'_{G_i} = 5 - i$. For any i, G_i is certainly of smallest order if $|G_i| = 2$:

8.10

Pick the 4 independent vertices and call the two remaining vertices A and B; since the independent vertices are not mutually adjacent but the graph is connected, they must be either:

- all adjacent to only the same remaining vertex (without loss of generality) *A*; in which case *A* must be adjacent to *B*, and thus $\alpha_G = 5$;
- all adjacent to some combination of both *A* and *B*; in which case there are at least two independent edges and $\alpha'_G \geq 2$.

8.11

In other words, a class of graphs for which the number of independent vertices is the same as the minimal number of covering edges: e.g., $K_{1,n}$.

8.12a

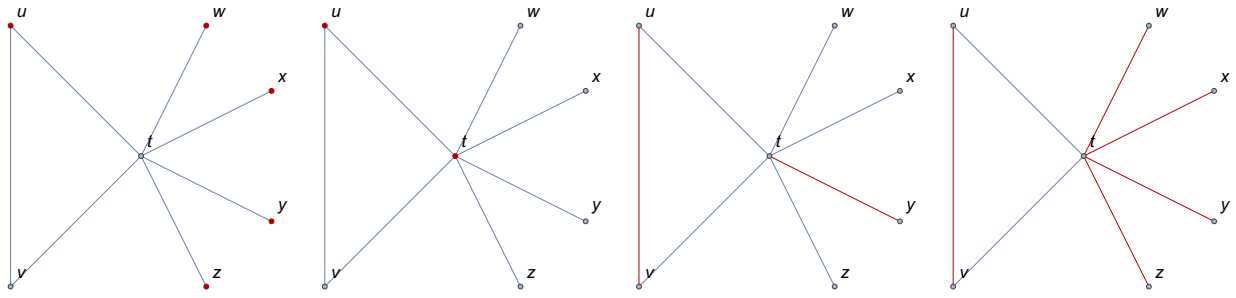
In P_3, the outer vertices constitute a covering but do not contain the minimum covering of the internal vertex. (false)

8.12b

In P_4, the inner edge is a (maximal) independent set of edges that is not contained in the maximum independent set of the outer two edges. (false)

8.13

```
GraphicsRow[
 Module[
  {g = Graph[
     {u, v, t, w, x, y, z},
     {u ↔ v, u ↔ t, v ↔ t, t ↔ w, t ↔ x, t ↔ y, t ↔ z},
     VertexCoordinates → {{-2, +2}, {-2, -2}, {0, 0}, {+1, +2}, {+2, +1}, {+2, -1}, {+1, -2}},
     VertexLabels → "Name"]},
  HighlightGraph[g, #[g]] & /@ {
    FindIndependentVertexSet,
    FindVertexCover,
    FindIndependentEdgeSet,
    FindEdgeCover}],
 ImageSize → Full]
```



8.14

G has a perfect matching iff $\alpha'_G = \frac{1}{2}|G| \Rightarrow \beta'_G = |G| - \alpha'_G = \frac{1}{2}|G| = \alpha'_G$.

8.15

In $G_1 + G_2$, each vertex of G_1 is connected to every vertex of G_2. Now considering an independent edge set of α_G, since $|G_2| > |G_1|$ we can have $|G_1|$ independent edges from the vertices of G_1 incident to some $|G_1|$ vertices of G_2, leaving $|G_2| - |G_1|$ vertices in G_2 yet without edges. But $\alpha'_{G_2} \geq \frac{1}{2}(|G_2| - |G_1|)$ so we can have that many independent edges in G_2 and connect all $|G_1|$ independent edges from G_1 to the remaining vertices of G_2; so that $\alpha'_G = |G_1| + \frac{1}{2}(|G_2| - |G_1|) = \frac{1}{2}(|G_1| + |G_2|)$. Then G has a perfect matching and $\alpha'_G = \beta'_G$.

8.16

A lower bound for the minimum vertex cover occurs in the best case, where all vertices of G are incident to some vertex of maximal degree Δ; in that case, $\beta = \frac{|G|}{\Delta+1}$.

8.2 Factorization

```
(* This isn't fool-proof: it assumes that any 1-factor contributes to a factorization,
and I don't know that that's true.  So far it gives meaningful results, though. *)
MyFactorize[g_, f_] := Module[
  {matching = f[g]},
  Module[
   {g1 = EdgeDelete[g, matching]},
   If[
    EmptyGraphQ[g1],
    {matching},
    Append[MyFactorize[g1, f], matching]]]]
```

Theorem 11

Every 3-regular bridgeless graph contains a 1-factor ⇐ (Theorem 10)

$\forall S \subset VG : k_{odd}(G - S) \leq |S|$

$G - S$ has no odd components ⇒ $k_{odd}(G - S) = 0$

$G - S$ has odd components G_i, and $3\,k_{odd}(G - S) \leq 3\,|S|$

at least $3\,k_{odd}(G - S)$ edges in G between vertices of S and vertices of $G - S$

for each G_i, let X_i be the set of edges in G between S and G_i

There are $k_{odd}(G - S)$ such X_i, and $|X_i| \geq 3$

$|X_i|$ is odd

$+_{v \in G_i} \deg_{G_i} v$ is even (Theorem 2.1)

$+_{v \in G_i} \deg_G v$ is odd

$\deg_G v = 3$ ⇐ G is 3-regular

$|G_i|$ is odd ⇐ G_i is an odd component

$|X_i| > 1$

G is bridgeless

no more than $3\,|S|$ edges between vertices of S and vertices of $G - S$

$\forall s \in S : \deg_G s = 3$ (some of the edges could be between vertices of S)

Theorem 13

Petersen Graph PG is not 1-factorizable ⇐

suppose PG *is* 1-factorizable

$\exists\, F_1, F_2, F_3$ 1-factors ⇐

PG is 3-regular

the spanning subgraph H : EH = $EF_1 \cup EF_2$ is a union of cycles

H is 2-regular

$H = 2\, C_5$ ⇐

H is not a single cycle ⇐

PG is not Hamiltonian

C_5 is the smallest cycle in PG

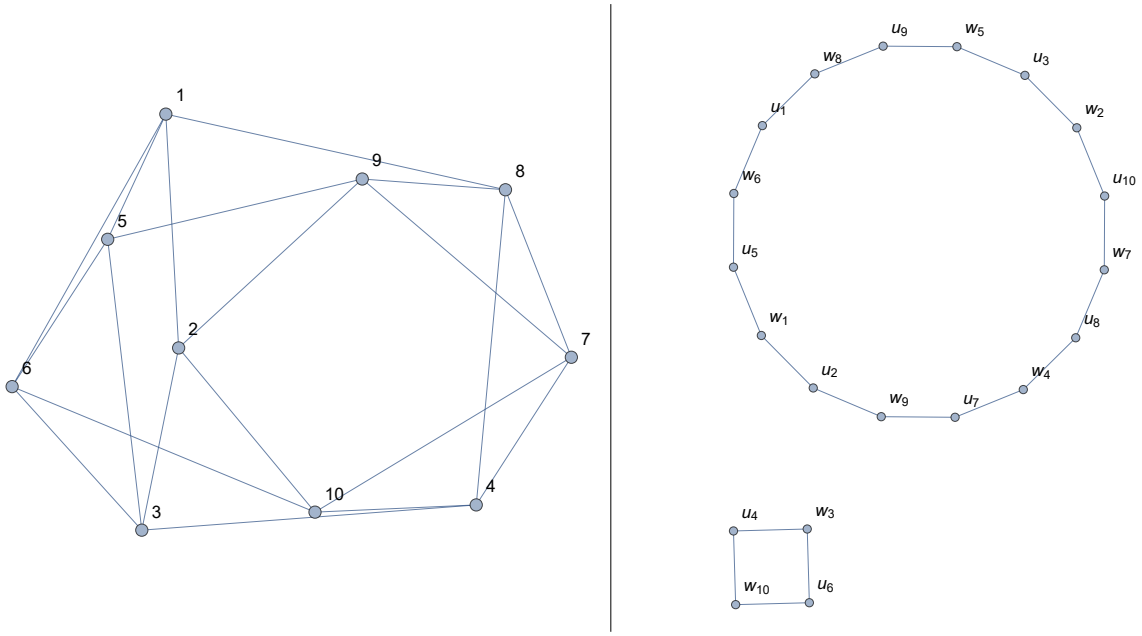
|VPG| = 10

contradiction ⇐

C_5 does not contain a 1-factor

Theorem 16

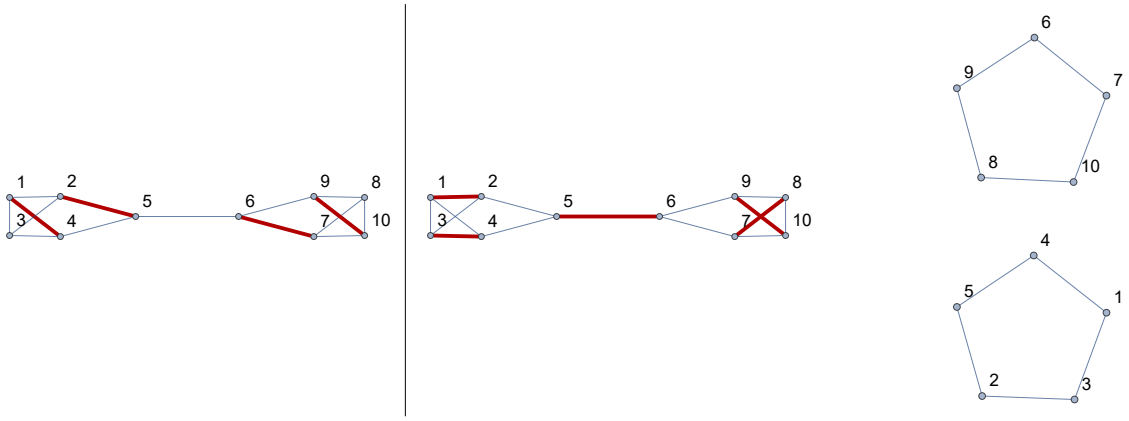
```
Module[
 {g = RandomGraph[DegreeGraphDistribution[Table[4, 10]]]},
 GraphicsRow[{
   Graph[g, VertexLabels → "Name"],
   Graph[
    u_#[[1]] ↦ w_#[[2]] & /@ FindEulerianCycle[g][[1]],
    VertexLabels → "Name"]},
  ImageSize → Full, Dividers -> {Center, False}]]
```



8.17

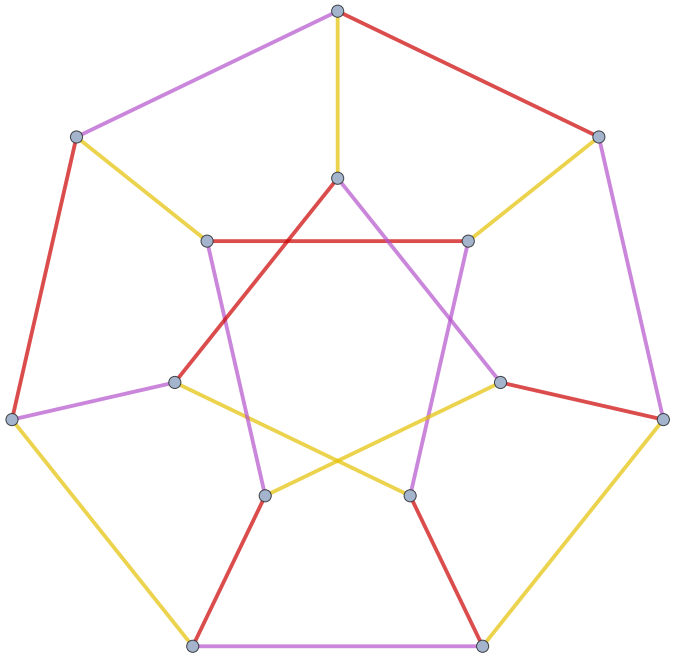
As cubic (3-regular) graphs, since G_1 and G_3 have fewer than three bridges they have 1-factors by Theorem 12. A 1-factor of G_1 cannot lack the bridge, because the two lobes are of odd order and cannot contain a 1-factor. The 1-factor that does contain the bridge leaves two C_5 when removed, which also cannot contain 1-factors, so G_1 is not factorizable:

```
Module[
 {g = Graph[{1 ↔ 2, 1 ↔ 3, 1 ↔ 4, 2 ↔ 3, 3 ↔ 4,
     2 ↔ 5, 4 ↔ 5, 5 ↔ 6, 6 ↔ 7, 6 ↔ 9, 7 ↔ 8, 8 ↔ 9, 7 ↔ 10, 8 ↔ 10, 9 ↔ 10},
    VertexLabels → "Name",
    GraphHighlightStyle → "Thick"]},
 GraphicsRow[
  Join[
   {HighlightGraph[g, {1 ↔ 4, 2 ↔ 5, 6 ↔ 7, 9 ↔ 10}]},
   #[{1 ↔ 2, 3 ↔ 4, 5 ↔ 6, 7 ↔ 8, 9 ↔ 10}] & /@
    {HighlightGraph[g, #] &, Graph[EdgeDelete[g, #]] &
    }],
  ImageSize → Full, Dividers → {{False, True}, None}
 ]]
```



The answer key states that G_3 is not factorizable, but this seems to give one:

```
Module[
 {g = PetersenGraph[7, 2, EdgeStyle → Thick]},
 HighlightGraph[g, MyFactorize[g, FindIndependentEdgeSet]]]
```



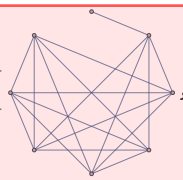
As for G_2, considering that if a perfect matching did not contain each of the three bridges the corresponding attached nodes (being of odd order) cannot possibly contain a 1-factor. Since a perfect matching must therefore contain the three bridges, the $K_{1,3}$ remaining at the center cannot possibly contain a 1-factor; so G_2 does not contain a 1 factor and is not factorizable.

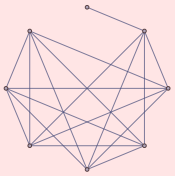
8.18

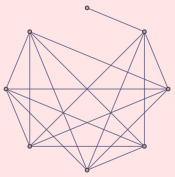
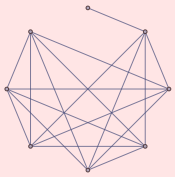
Generalize the construction of G_2 of Exercise 17 to find arbitrary odd-regular graphs that do not contain a 1-factor:

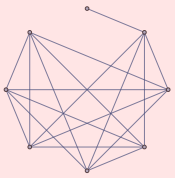
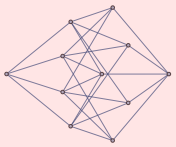
```
Manipulate[
 Module[
  {g = GraphUnion @@ Append[
      Map[
       RenameSubscriptGraph[
        Module[
         {t = Table[i, {i, (r + 1) - (r - 1) + 1, r + 1}]},
         EdgeAdd[
          EdgeAdd[
           EdgeDelete[
            CompleteGraph[r + 1],
            Apply[UndirectedEdge] /@ Partition[t, 2]],
           # ⟷ (r + 2) & /@ t],
          (r + 2) ⟷ (r + 3)]],
        # ] &,
       Range[r]],
      Graph[
       Join @@ Append[
        Module[
         {t = Table[v_i, {i, r}]},
         Function[i,
          i_{r+3} ⟷ # & /@ Drop[t, {i}]] /@ Range[r]],
        c ⟷ v_# & /@ Range[r]]]]},
  Labeled[g, {
    RegularGraphQ[g],
    Length[EdgeList[FindIndependentEdgeSet[g]]] == 1/2 Length[VertexList[g]]}]],
 {{r, 5}, 3, 9, 2}]
```

r

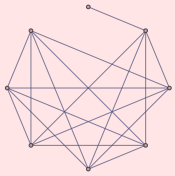
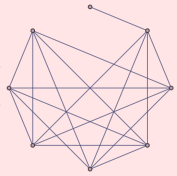
GraphUnion[RenameSubscriptGraph[ , 1],

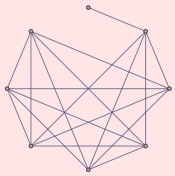
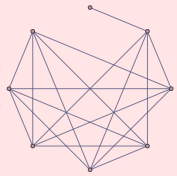
RenameSubscriptGraph[ , 2],

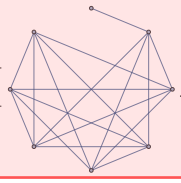
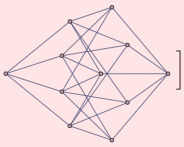
RenameSubscriptGraph[ , 3], RenameSubscriptGraph[ , 4],

RenameSubscriptGraph[ , 5], ]

{RegularGraphQ[GraphUnion[

RenameSubscriptGraph[ , 1], RenameSubscriptGraph[ , 2],

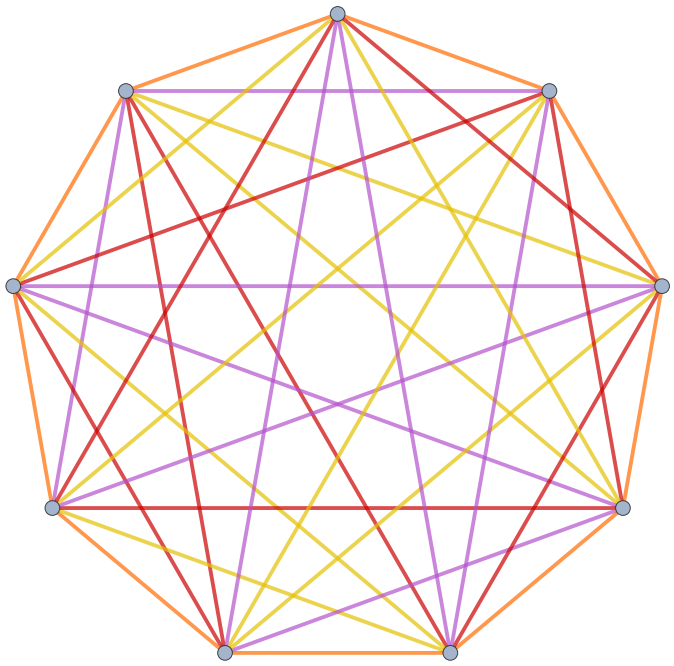
RenameSubscriptGraph[ , 3], RenameSubscriptGraph[ , 4],

RenameSubscriptGraph[ , 5], ]], False}

8.19

This is equivalent to asking for a C_9-factorization of K_9. By Theorem 17, such a factorization exists.

```
Module[
 {g = CompleteGraph[9, EdgeStyle → Thick]},
 HighlightGraph[g, MyFactorize[g, FindHamiltonianCycle[#][[1]] &]]]
```



8.20

The proof of Theorem 11 (see above) works by establishing a 'high enough' lower bound to $|X| = \cup_i X_i$, the number of edges in G between the components G_i and S. If the requirement that G is bridgeless is removed entirely, the lower limit drops from $3\,k_{odd}(G-S)$ to $k_{odd}(G-S)$, so the resultant inequality $k_{odd}(G-S) \le 3\,|S|$ isn't strong enough to satisfy Tutte's condition. However, if G has at most 2 bridges then $|X| \ge 3\,k_{odd}(G-S) - 2$ and $k_{odd}(G-S) - \frac{2}{3} \le |S| \Rightarrow k_{odd}(G-S) \le |S| + \frac{2}{3} \Rightarrow k_{odd}(G-S) \le |S|$, since obviously $|S|$ is integral.

8.21

Letting S be the partition of $K_{3,5}$ that contains 3 vertices, we have $k_{odd}(G-S) = 5 \not\le 3$.

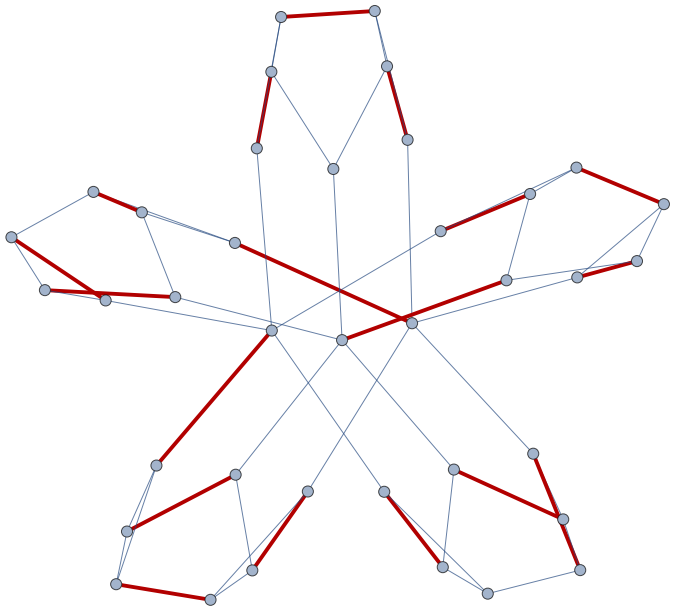
8.22a

The inequality works when the lower bound is 'high enough' and the upper bound is 'low enough'. If G contains two vertices of degree 5, then the lower bound $|X| = \cup_i |X_i|$ *could* be raised by 4 if those two vertices were contained in some G_i; but since this is not generally true, the best we can still do is $|X| \ge 3\,k_{odd}(G-S)$. On the upper bound, it could still be that $|X| \le 3\,|S|$ if the two vertices of degree 5 were, for example, both in S. But since this is not generally true (for example, the vertices could be both in some G_i), the best we can do is $|X| \le |S| + 2$. But then $3\,k_{odd}(G-S) \le |S| + 2 \Rightarrow k_{odd}(G-S) \le |S| + \frac{2}{3} \Rightarrow k_{odd}(G-S) \le |S|$ and Tutte's condition remains satisfied.

8.22b

The three vertices of degree 5 allow a greater number of edges X from S to $\cup_i G_i$:

```
Module[
 {g = GraphUnion @@ Function[
       l,
       EdgeAdd[
        RenameSubscriptGraph[%269, l],
        # ⟷ l_# & /@ Range[3]]
      ] /@ {a, b, c, d, e}},
 Labeled[
  HighlightGraph[g,
   FindIndependentEdgeSet[g],
   GraphHighlightStyle → "Thick"],
  Counts[VertexDegree[g]]]]
```



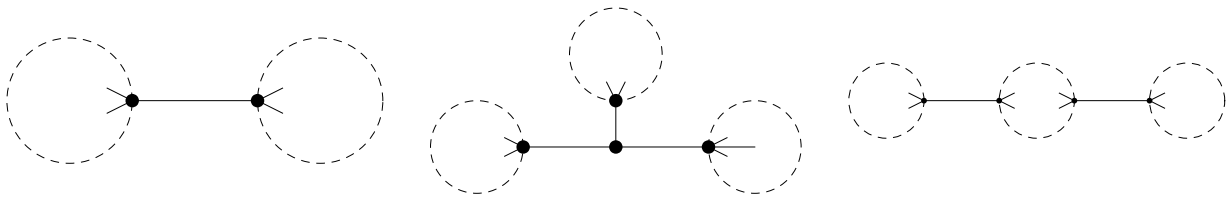
‹| 5 → 3, 3 → 35 |›

The lobes are of odd order and so cannot have a 1-factor without one of the three edges connecting them to the graph center. Since there are only three vertices in the center, there are two lobes containing a vertex that is not incident to any edge of the matching.

8.23

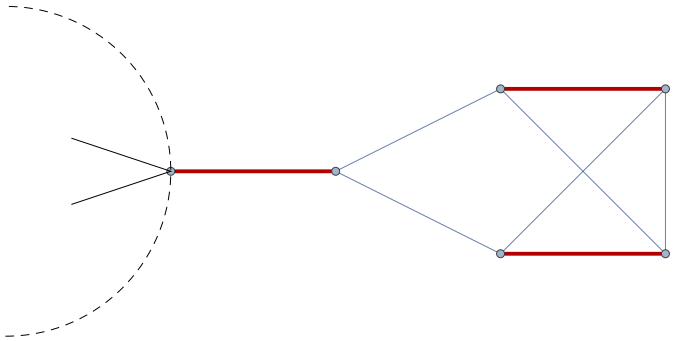
A 3-regular graph with a single bridge is depicted in the figure on the left. If there are multiple bridges, they cannot be adjacent because it would force the existence of a third bridge which lies on a different pathh; as in the center figure. Thus, a 3-regular graph with multiple bridges must have the bridges non-adjacent, as in the figure on the right.

```
Module[
 {MyNode}, (* can't define local function here *)
 MyNode[i___] := {
   Dashed,
   Circle[],
   Dashing[{}],
   Rotate[{
       Line[{{0.6, +0.2}, {1.0, 0.0}, {0.6, -0.2}}],
       Point[{{1.0, 0.0}}]},
      #,
      {0, 0}] & /@ i};
 GraphicsRow[{
   Graphics[{
     PointSize[Large],
     Translate[MyNode[{0}], {0, 0}],
     Translate[MyNode[{Pi}], {4, 0}],
     Line[{{1, 0}, {3, 0}}]}],
   Graphics[{
     PointSize[Large],
     Translate[MyNode[{0}], {0, 0}],
     Line[{{1, 0}, {3, 0}, {3, 1}}],
     Point[{{3, 0}}],
     Translate[MyNode[{3/2 Pi}], {3, 2}],
     Line[{{3, 0}, {6, 0}}],
     Translate[MyNode[{Pi}], {6, 0}]}],
   Graphics[{
     Translate[MyNode[{0}], {0, 0}],
     Line[{{1, 0}, {3, 0}}],
     Translate[MyNode[{0, Pi}], {4, 0}],
     Line[{{5, 0}, {7, 0}}],
     Translate[MyNode[{Pi}], {8, 0}]}]},
  ImageSize → Full]]
```



Now consider one of the lobes, with the given graph attached to it:

```
Show[
 Graph[
  {1 ↔ 2, 2 ↔ 3, 2 ↔ 4, 3 ↔ 5, 3 ↔ 6, 4 ↔ 5, 4 ↔ 6, 5 ↔ 6},
  VertexCoordinates → {{0, 0}, {2, 0}, {4, +1}, {4, -1}, {6, +1}, {6, -1}},
  GraphHighlight -> {1 ↔ 2, 3 ↔ 5, 4 ↔ 6},
  GraphHighlightStyle → "Thick"],
 Graphics[{
   Dashed,
   Circle[{-2, 0}, 2, {-1/2 Pi, +1/2 Pi}],
   Dashing[{}],
   Line[{{-1.2, +0.4}, {0, 0}, {-1.2, -0.4}}]
  }]]
```

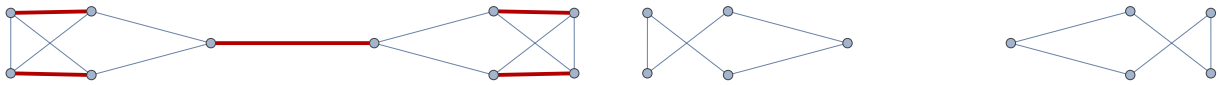


For this graph, there cannot be a 1-factor that does not contain the bridge, because the appendage would not contain a perfect matching. By Theorem 12 then, the graph contains a 1-factor that also contains the bridge. By extension, all 3-regular graphs in which the bridges are configured as in the above right figure contain a 1-factor, that also contains each of the bridges.

8.24

Removing a 1-factor from a 3-regular graph results in a 2-regular graph, which is to say, a graph in which the components are cycles. This is a 2-factor of the graph. **Note:** It's unclear why the graph has to be bridgeless:

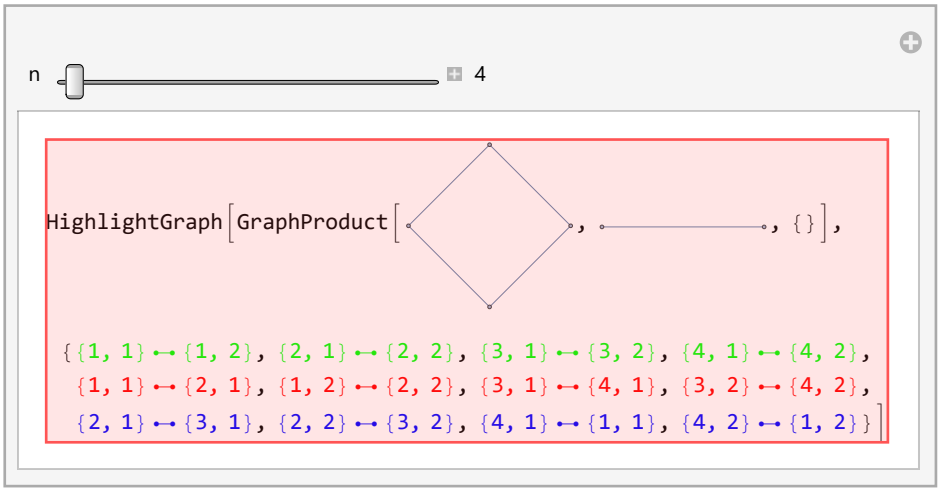
```
Module[
 {g = Graph[
    {1 ↔ 2, 3 ↔ 4, 1 ↔ 3, 1 ↔ 4, 2 ↔ 3, 2 ↔ 5,
     4 ↔ 5, 5 ↔ 6, 6 ↔ 7, 6 ↔ 8, 7 ↔ 9, 8 ↔ 10, 7 ↔ 10, 8 ↔ 9, 9 ↔ 10},
    GraphHighlightStyle → "Thick"],
  h = {1 ↔ 2, 3 ↔ 4, 5 ↔ 6, 7 ↔ 10, 8 ↔ 9}},
 GraphicsRow[{
   HighlightGraph[g, h],
   Graph[EdgeDelete[g, h], VertexCoordinates → GraphEmbedding[g]]},
  ImageSize → Full]]
```



8.25

Observe first that $C_n \times K_2$ is 3-regular. For even n, a 1-factor can be found in the K_2-part of the product, as shown below left. For odd n, a 1-factor can be found as shown below right. The resulting 2-factors are of even length and is 1-factorable. The graphs are therefore all 1-factorable.

```
Manipulate[
 Module[
  {g, of1, of2, of3},
  g = GraphProduct[CycleGraph[n], CompleteGraph[2], {}];
  If[
   EvenQ[n],
   of1 = {#, 1} ⟷ {#, 2} & /@ Range[n];
   of2 = Join @@ Table[{i, j} ⟷ {i + 1, j}, {i, 1, n, 2}, {j, 2}];
   of3 = Join @@ Table[{i, j} ⟷ {Mod[i + 1, n, 1], j}, {i, 2, n + 1, 2}, {j, 2}],
   of1 = Join[
     {{1, 1} ⟷ {1, 2}},
     Table[{i, j} ⟷ {i + 1, j}, {i, 2, n, 2}, {j, 2}]];
   of2 = Join[
     {{1, 1} ⟷ {2, 1}, {1, 2} ⟷ {2, 2}, {n, 1} ⟷ {n, 2}},
     Table[{i, 1} ⟷ {i, 2}, {i, 3, n - 1}]];
   of3 = Join[
     {{n, 1} ⟷ {1, 1}, {n, 2} ⟷ {1, 2}, {2, 1} ⟷ {2, 2}},
     Table[{i, j} ⟷ {i + 1, j}, {i, 3, n - 1, 2}, {j, 2}]]];
  HighlightGraph[
   g,
   Join[
    Style[#, Thick, Green] & /@ of1,
    Style[#, Thick, Red] & /@ of2,
    Style[#, Thick, Blue] & /@ of3]]],
 {n, 4, 20, 1, Appearance → "Labeled"}]
```

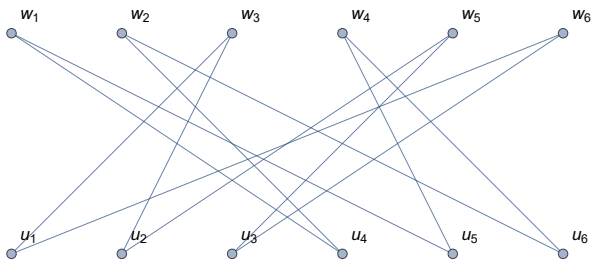


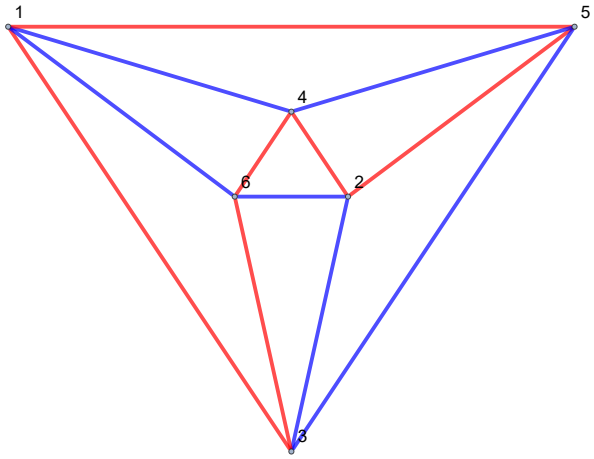
8.26

The 2-factor containing $u_1 \longleftrightarrow v_1$ must also contain $u_5 \longleftrightarrow v_5$; and must therefore pass through at least u_2 or u_3. If it passes through u_2, it must also pass through u_3 and u_4, since the graph is 4-regular and u_3, u_4 obviously cannot form a cycle independently. Mutatis mutandis if it passes through u_3. Similarly the 2-factor must pass through all of v_2, v_3, v_4. This means the 2-factor passes through all vertices and is thus Hamiltonian.

8.27

```
Module[
 {g = CompleteGraph[
    {2, 2, 2},
    VertexLabels → "Name",
    VertexCoordinates → {{-10, +5}, {+2, -1}, {0, -10}, {0, +2}, {+10, +5}, {-2, -1}}],
  e = {1 ↔ 3, 3 ↔ 5, 5 ↔ 4, 4 ↔ 2, 2 ↔ 3, 3 ↔ 6, 6 ↔ 4, 4 ↔ 1, 1 ↔ 6, 6 ↔ 2, 2 ↔ 5, 5 ↔ 1}},
 GraphicsRow[{
   Graph[
    Join @@ ({u#, w#} & /@ Range[6]),
    u#[[1]] ↔ w#[[2]] & /@ e,
    VertexCoordinates → Join @@ ({{#, -1}, {#, +1}} & /@ Range[6]),
    VertexLabels → "Name"],
   HighlightGraph[g,
    Join[
     Style[#, Red, Thick] & /@ {1 ↔ 3, 3 ↔ 6, 6 ↔ 4, 4 ↔ 2, 2 ↔ 5, 5 ↔ 1},
     Style[#, Blue, Thick] & /@ {1 ↔ 4, 4 ↔ 5, 5 ↔ 3, 3 ↔ 2, 2 ↔ 6, 6 ↔ 1}]]},
  ImageSize → Full]]
```





8.28

Removing the two 1-factors results in a 4-regular graph. By Theorem 16 that graph is 2-factorable. Taking the union of one of the 2-factors with one of the 1-factors results in a 3-regular spanning graph of G, that is, a 3-factor.

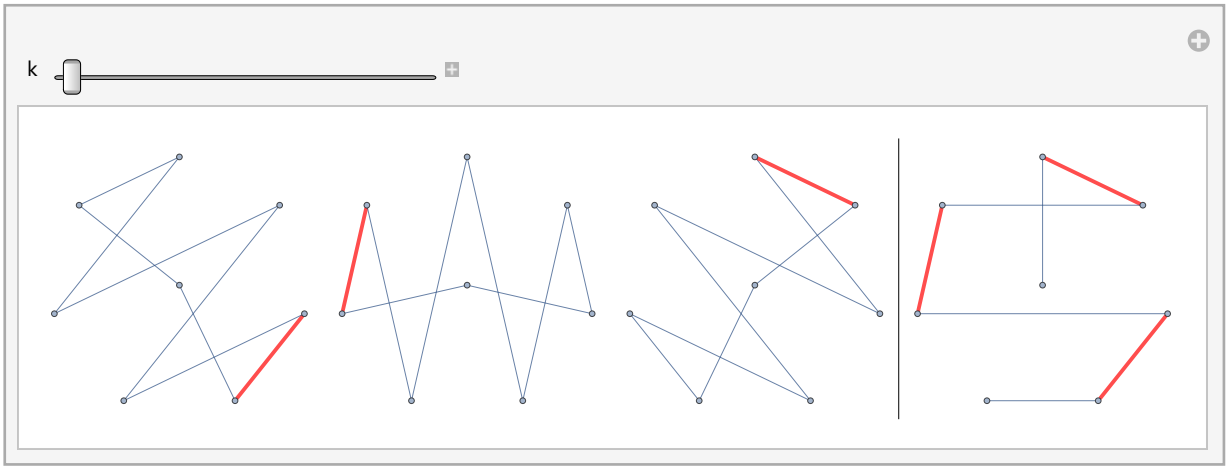
8.29

By Theorem 20, any complete graph of even order $2k$ can be factored into $k - 1$ Hamiltonian cycles and a single 1-factor; if we can pick one edge from each of the cycles (leaving $k - 1$ Hamiltonian paths of $2k - 1$ edges each) and adjoin them to the

1-factor (having k edges), that could be converted into a kth Hamiltonian path (having $k + (k - 1) = 2k - 1$ edges).

I believe there are errors in the proof of Theorem 20; there are no "edges parallel to $v_0 v_1$" or "edges parallel to $v_0 v_{k+1}$". It seems this can be fixed by substituting "perpendicular", but that is an empirical observation based on the cases of $k = 3, 5$ and not mathematically sound. In fact, the proof of the Theorem itself seems equally fuzzy. Thus, solution by demonstration:

```
Manipulate[(* *** still incorrect for odd k *)
 Module[
  {MHC, MOF, k2 = 2 k, k21 = 2 k - 1},
  MHC[r_] := (* generate cyclic Hamiltonian cycle with highlighted edge *)
   HighlightGraph[
    Graph[
     Range[0, k21],
     Join[
      {0 ↔ r + 1, 0 ↔ k + r + 1},
      Table[Mod[k2 + r - i, k21, 1] ↔ Mod[r + i + 1, k21, 1], {i, k - 1}],
      Table[Mod[2 + r - i, k21, 1] ↔ Mod[r + i + 1, k21, 1], {i, k - 1}]],
     VertexCoordinates → Prepend[CirclePoints[k21], {0, 0}]],
    Style[If[EvenQ[r], r + 1 ↔ r + 2, Mod[r + k, k21, 1] ↔ Mod[r + k + 1, k2, 1]], Red, Thick]];
  MOF[] := Graph[ (* generate 1-factor with highlighted edges contributed from cycles *)
    Range[0, k21],
    Join[
     {0 ↔ k},
     Table[k2 - i ↔ i, {i, k - 1}],
     Table[Style[If[EvenQ[r], r + 1 ↔ r + 2,
        Mod[r + k, k21, 1] ↔ Mod[r + k + 1, k21, 1]], Red, Thick], {r, 0, k - 2}]],
    VertexCoordinates → Prepend[CirclePoints[k2 - 1], {0, 0}]];
  GraphicsRow[
   Append[Table[MHC[r], {r, 0, k - 2}], MOF[]],
   Dividers -> {k → True, False},
   ImageSize → Full]],
 {k, 4, 8, 1}]
```



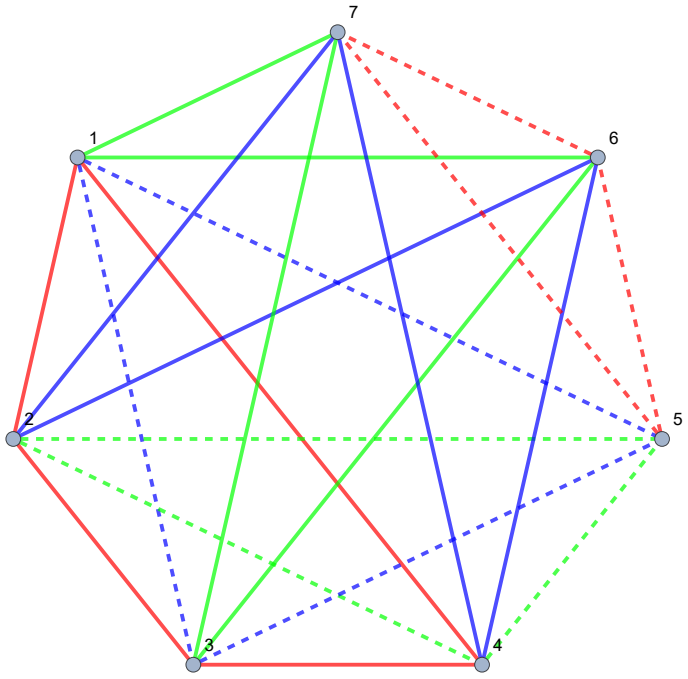
8.30

Since $27 =_6 3$, by Theorem 19 there exists a Kirkman triple system of order 27. By the text preceding that Theorem, $n = 6k + 3 \Rightarrow k = \frac{n-3}{6} = 4$ there is a 9 C_3-factorization of K_{27}. Since each vertex of K_{27} is incident to 26 other vertices, there must be 13 such factors.

8.31

For no 2-factor to be a Hamiltonian cycle, it has to consist of a cycle of length 3 and a cycle of length 4.

```
HighlightGraph[
 CompleteGraph[7, VertexLabels → "Name"],
 Join[
  Apply[
    Function[(* generate a list of edges of a cycle of given length,
     renaming, and applying a style *)
     {l, r, s},
     Style[#, s] & /@ Replace[EdgeList[CycleGraph[l]], r, {2}]
    ]] /@ {
    {4, {}, {Thick, Red}},
    {3, {1 → 5, 2 → 6, 3 → 7}, {Thick, Red, Dashed}},
    {4, {2 → 6, 4 → 7}, {Thick, Green}},
    {3, {1 → 5, 3 → 4}, {Thick, Green, Dashed}},
    {4, {1 → 6, 3 → 7}, {Thick, Blue}},
    {3, {2 → 5}, {Thick, Blue, Dashed}}
   }]]
```



8.32

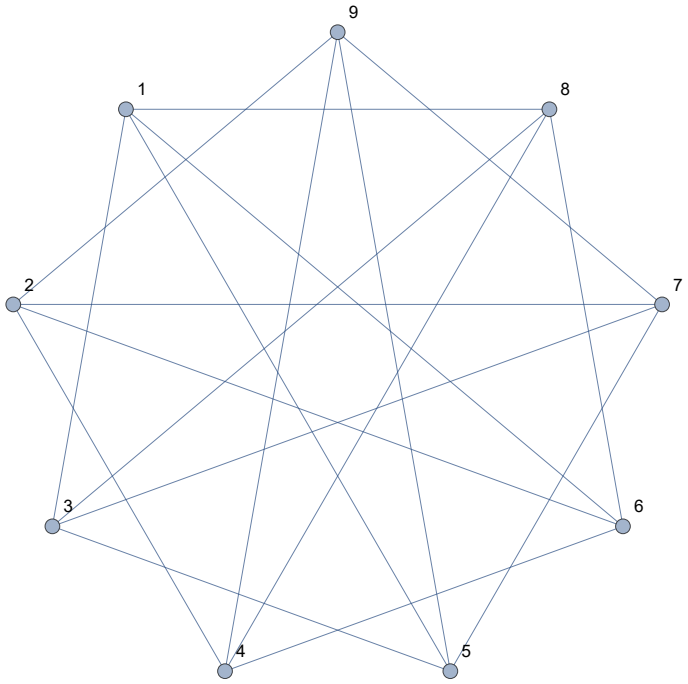
For this factorization to exist, the factors must consist of one each of the following partitioning of cycles:

```
IntegerPartitions[9, All, Range[3, 9]]
```

```
{{9}, {6, 3}, {5, 4}, {3, 3, 3}}
```

The factors of C_9 and $3\,C_3$ are symmetrical, so both can be removed first, without loss of generality:

```
EdgeDelete[
 CompleteGraph[9, VertexLabels → "Name"],
 Join[
  EdgeList[CycleGraph[9]],
  Replace[EdgeList[CycleGraph[3]], {1 → 1, 2 → 4, 3 → 7}, {2}],
  Replace[EdgeList[CycleGraph[3]], {1 → 2, 2 → 5, 3 → 8}, {2}],
  Replace[EdgeList[CycleGraph[3]], {1 → 3, 2 → 6, 3 → 9}, {2}]]]
```



This graph is (obviously) 4-regular. If we were to remove the factor consisting of $C_3 \cup C_6$ we should be left with the last factor of $C_4 \cup C_5$. The factor $C_3 \cup C_6$ must therefore disconnect the C_4 from the C_5, meaning that it should consist simultaneously of 8 edges and 10 edges, which is impossible; therefore, this factorization cannot exist.

8.3 Decompositions and Graceful Labelings

```
(* Given a graph with a vertex labeling, apply the corresponding graceful edge labelings *)
LabelEdges[g_] :=
 SetProperty[
  g,
  EdgeLabels → (
      # → Abs[Subtract @@
         Association[PropertyValue[g, VertexLabels]] /@
          (* apply the association of vertices to vertex labels as a function to *)
          Apply[List, #] (* the both ends of the edge, by converting it to a list *) ]
     ) & /@ EdgeList[g]]
```

8.33

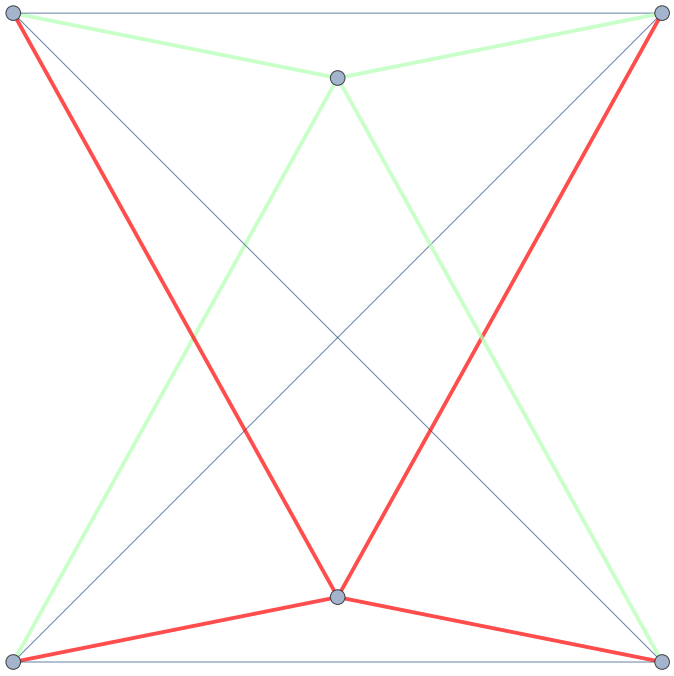
$K_{2,2,2}$ has size 12 and $K_{1,4}$ has size 4, so there would have to be 3 component graphs. $K_{2,,2,2}$ is 4-regular; three of its vertices should correspond to the hub of a $K_{1,4}$, but since each $K_{1,4}$ is incident with four other vertices, at most two can be placed inside $K_{2,2,2}$.

```
Length[EdgeList[CompleteGraph[{2, 2, 2}]]]
Length[EdgeList[CompleteGraph[{1, 4}]]]
```

12

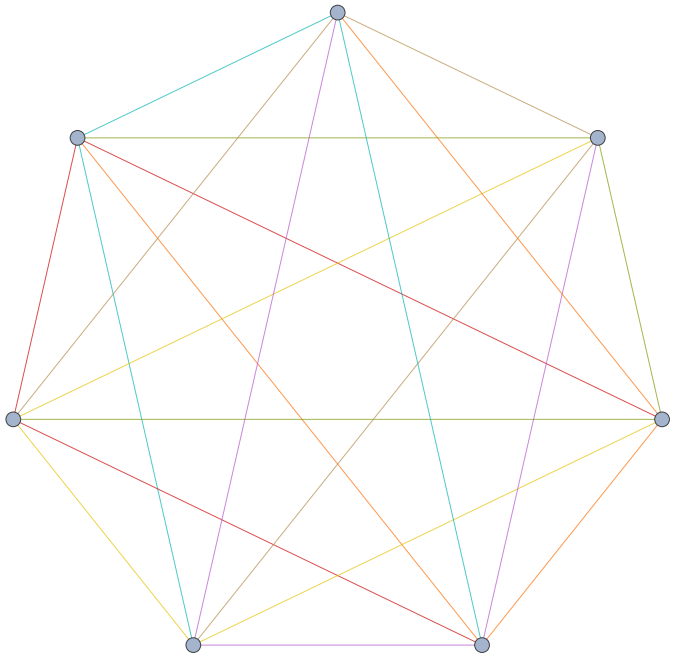
4

```
HighlightGraph[
 CompleteGraph[{2, 2, 2},
  VertexCoordinates → {{-1, -1}, {-1, +1}, {0, -0.8}, {0, +0.8}, {+1, -1}, {+1, +1}}],
 Join[
  Style[3 ⟷ #, Thick, Red] & /@ {1, 2, 5, 6},
  Style[4 ⟷ #, Thick, Lighter[Green, .7]] & /@ {1, 2, 5, 6}]
]
```



8.34

```
HighlightGraph[
 CompleteGraph[7],
 Table[
  {Mod[i - 3, 7, 1] ↔ i, i ↔ Mod[i + 1, 7, 1], Mod[i + 1, 7, 1] ↔ Mod[i + 3, 7, 1]},
  {i, 7}]]
```



8.35

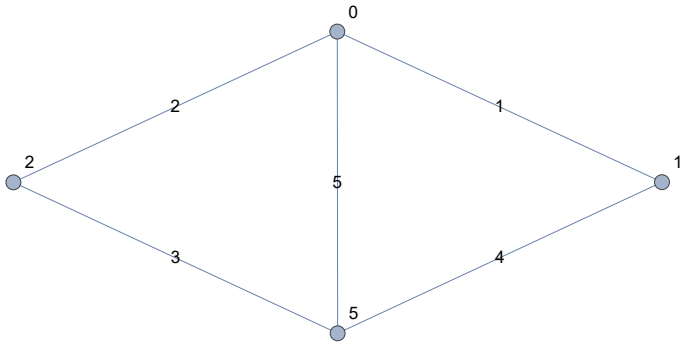
Brute force: C_5 and C_6 do not have graceful labelings; C_8 does:

```
Function[{n},
  Select[
   Join[
      {n - 1, 0, n}, (* assume WLOG cycle starts like this *)
      #] & /@ Permutations[Range[1, n - 2], {n - 3}],
   (* complete the cycle with all possible permutations *)
   DuplicateFreeQ[
     Table[Abs[#[[i]] - #[[Mod[i + 1, n, 1]]]], {i, n}] (* calculate edge labelings *)
    ] &,
   1 (* first one *)]] /@ {5, 6, 8}
```

```
{{}, {}, {{7, 0, 8, 2, 3, 6, 1, 5}}}
```

8.36

```
LabelEdges[
 Graph[
  {1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 1, 1 ↔ 3},
  VertexLabels → MapIndexed[First[#2] → #1 &, {0, 2, 5, 1}]]]
```

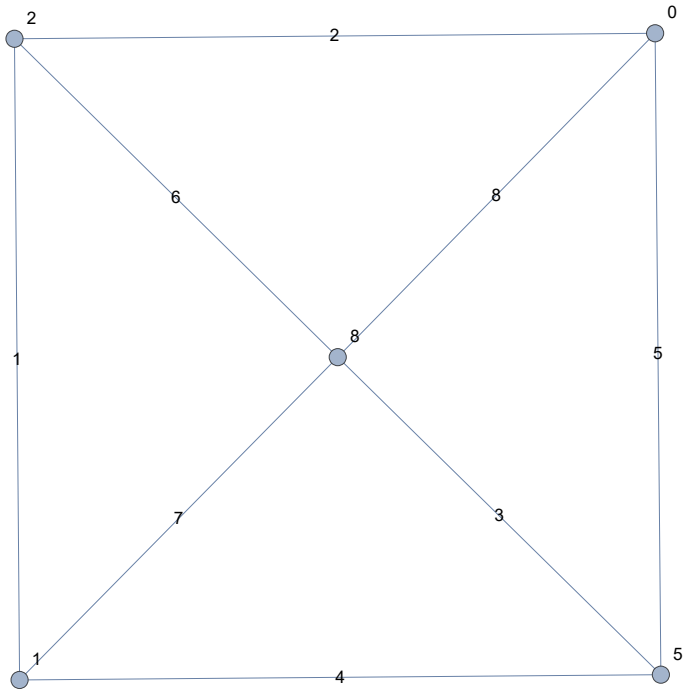


For there to exist a graceful labeling of the middle graph, in each triangle two of the edge labels have to add up to the third edge label. One of the triangles contains 6 as an edge label; therefore the other two edges of that triangle must be either 1, 5 or 2, 4. In the former case the remaining edge labels are 2, 3, 4 and in the latter 1, 3, 5; in neither case can we make two of the three edge labels add to form the third. Therefore no graceful labeling exists.

A graceful labeling of the third graph can be found by naive brute force:

```
(* find a graceful labeling of the given graph by brute force *)
GracefulLabeling[g_] :=
 First[
  Select[
   (* generate all possible vertex labelings;
   does not take into account symmetries of the graph *)
   Permutations[
    Range[0, Length[EdgeList[g]]], {Length[VertexList[g]]}],
   Function[{l},
    DuplicateFreeQ[ (* is graceful labeling *)
     Map[ (* calculate edge labelings *)
      Abs[Subtract @@ (* calculate edge labeling *)
          Map[ (* convert vertex to corresponding labeling *)
           (* construct association of vertices to corresponding labelings *)
           AssociationThread[VertexList[g] → l],
           #]] &,
       (* convert each edge to its corresponding pair of vertices *)
       (Map[Apply[List], EdgeList[g]])]]],
   1]]
```

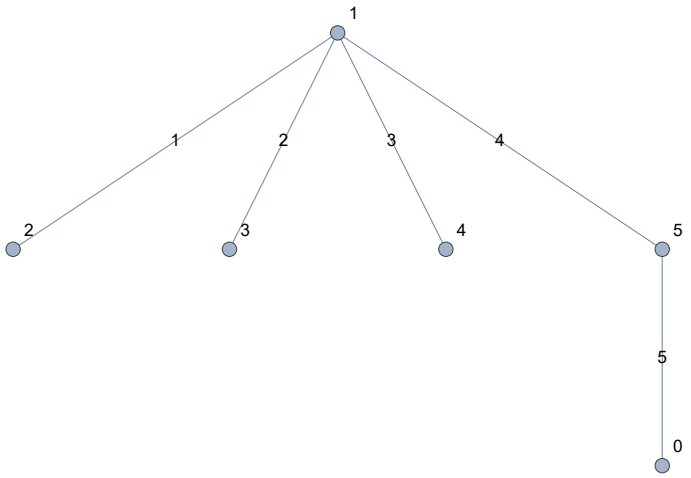
```
LabelEdges[
  Graph[
   #,
   VertexLabels → MapIndexed[
     First[#2] → #1 &,
     GracefulLabeling[#]]]
 ] &[Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 1, 1 ↔ 5, 2 ↔ 5, 3 ↔ 5, 4 ↔ 5}]]
```



8.37

Show that the tree has a graceful labeling; then K_{11} is T-decomposable by Theorem 24:

```
LabelEdges[
 Graph[
  {0 ↔ 1, 0 ↔ 2, 0 ↔ 3, 0 ↔ 4, 4 ↔ 5},
  VertexLabels → {0 → 1, 1 → 2, 2 → 3, 3 → 4, 4 → 5, 5 → 0}]]
```



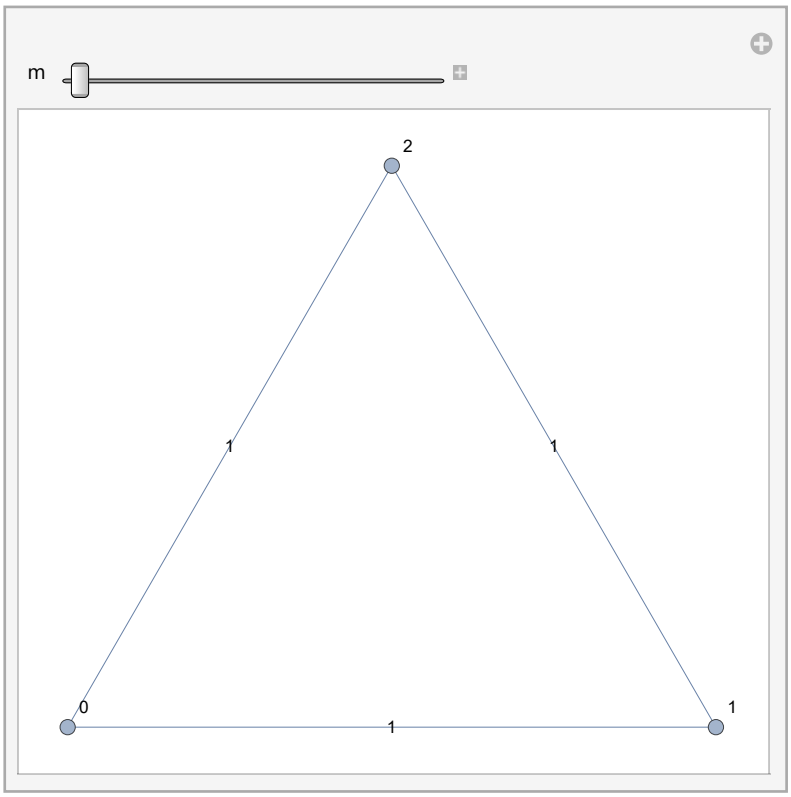
8.38a

Personally, I find the proof of Theorem 24 somewhat fuzzy and glib; for example, it isn't shown that the cyclic copies of the tree are disjoint. In any case, there doesn't appear to be any dependency within the proof that T is a tree and not just any graceful graph G; so we can immediately generalize it.

8.38b

Label/number the vertices of $K_{2\,m+3}$ as $\{0, \ldots, (2\,m+3)-1\} = \{0, \ldots, 2\,m+2\}$. Arranging the vertices (as in the Theorem) as a regular polygon, label the edges as $\{1, m+1\}$, being the distance between the adjacent vertices as measured along the edges of the polygon. Formally, $\forall\, u, v \in V(K_{2\,m+3})$, we define $d(u, v) = \min\left\{\vec{d}(u, v), \vec{d}(v, u)\right\}$, where $\vec{d}(u, v) = \min \text{mod}_{2\,m+3}\, v - u$ such that $\vec{d}(u, v) > 0$. This results in $2\,m+3$ edges each of distance $\{1, \ldots, m+1\}$.

```
Manipulate[
 Module[
  {g = SetProperty[ (* VertexReplace drops the VertexLabels property *)
     VertexReplace[ (* renumber vertices to be 0-base *)
      CompleteGraph[2 m + 1],
      Table[i + 1 → i, {i, 0, 2 m}]], (* list of associations renumbering vertices *)
     VertexLabels → "Name"]},
  SetProperty[ (* set edge labels *)
   g,
   EdgeLabels → Function[e,
     e → Min[
       Mod[#[Subtract @@ e], VertexCount[g]] & /@ {Plus, Minus}
      ]] /@ EdgeList[g]]],
 {m, 1, 5, 1}]
```



Let $G \subset K_{2m+3}$ as in Theorem 24. Since G is graceful, there is exactly one edge each labeled $\{1, \ldots, m\}$. The $2m + 3$ rotations

8.4 Instant Insanity

It's not clear that the graph theory contributes materially to the process of finding a solution; it merely restates the problem in different terms. The gist lies in the generation of "the seventeen 2-regular spanning pseudographs" (a manual process to which the theory doesn't contribute) and the matching of those pseudographs to the composite psudograph (again a manual and nontrivial process to which graph theory hasn't contributed anything).

```
InstantInsanity[cubes_] := Select[
  Flatten[ (* flatten outer product into a plain list *)
   Outer @@ (* outer product of rotations of all cubes *)
    Append[Prepend[(* construct argument list for Outer[List, {}, {}, {}, {}, 1] *)
      Permute[#,
          PermutationGroup[{ (* group of rotations of the cube *)
            Cycles[{{3, 6, 4, 5}}],
            Cycles[{{1, 5, 2, 6}}],
            Cycles[{{1, 4, 2, 3}}]}]
        ] & /@ cubes,
      List], 1],
   Length[cubes] - 1],
  AllTrue[
    Transpose[#[[All, 1 ;; 4]]],
    DuplicateFreeQ] &,
  1]
```

8.39

```
InstantInsanity[{{R, G, R, R, Y, B}, {Y, B, R, B, G, G}, {Y, G, R, G, B, Y}, {G, G, Y, B, B, R}}]
```

```
{{{R, G, R, R, Y, B}, {Y, B, G, G, B, R}, {G, Y, B, Y, R, G}, {B, R, Y, B, G, G}}}
```

8.40

```
InstantInsanity[{{Y, G, B, G, B, R}, {G, B, R, Y, R, G}, {Y, R, Y, G, B, B}, {G, Y, R, R, B, R}}]
```

```
{{{B, G, B, R, Y, G}, {Y, R, G, B, R, G}, {G, Y, R, Y, B, B}, {R, B, Y, G, R, R}}}
```

8.41

```
InstantInsanity[{{Y, R, G, R, G, Y}, {Y, Y, G, B, G, Y}, {Y, R, B, R, R, G}, {Y, R, G, G, B, B}}]
```

```
{{{Y, R, G, Y, R, G}, {G, Y, B, G, Y, Y}, {R, G, R, B, Y, R}, {B, B, Y, R, G, G}}}
```

8.42

```
InstantInsanity[{{G, B, G, R, R, Y}, {R, G, Y, B, R, B}, {Y, Y, G, R, B, G}, {B, R, B, Y, G, Y}}]
```

```
{{{G, B, G, R, R, Y}, {R, G, Y, B, R, B}, {Y, Y, R, G, G, B}, {B, R, B, Y, G, Y}}}
```

8.43

```
InstantInsanity[{{G, B, R, Y, Y, Y}, {G, R, Y, B, Y, Y}, {G, Y, B, R, Y, Y}, {G, B, Y, R, Y, Y}}]
```

```
{}
```

8.44

There can never be a unique solution, because in any solution the cubes can be swapped (without rotating) thereby giving other solutions; the cubes can be rotated to give other solutions; etc.

8.45

```
InstantInsanity[{{R, R, R, R, R, R}}]
```

```
{{{R, R, R, R, R, R}}}
```

```
InstantInsanity[{{R, G, R, G, R, R}, {R, G, R, G, R, R}}]
InstantInsanity[{{R, G, R, G, R, R}, {R, R, G, G, R, R}}]
```

```
{{{R, G, R, G, R, R}, {G, R, G, R, R, R}}}
```

```
{}
```

8.6 Bi-Graceful Graphs

8.46

Since a graceful edge labeling is surjective there is an edge with label m; therefore there is a vertex with label 0. Change the label of that vertex to $2m$; then incident edges' e labelings change to $2m - f'e$, which are unique and result in a non-graceful labeling.

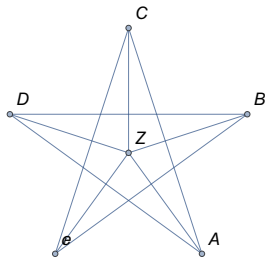
8.47

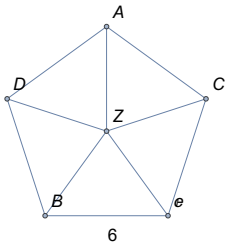
9 Planarity

9.1 Planar Graphs

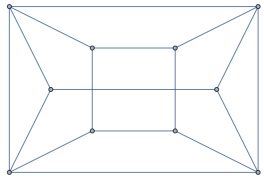
- 9.1

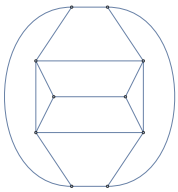
```
GraphicsGrid[
 MapThread[
  Function[{g, vc0, vc1}, {
    (* draw the graph with the first set of vertex coordinates *)
    Graph[g, VertexCoordinates → vc0],
    (* draw the graph with the second (supposedly planar) set of coordinates *)
    Labeled[ (* label with the number of planar regions *)
     Graph[g, VertexCoordinates → vc1],
     2 - VertexCount[g] + EdgeCount[g],
     ImageMargins → 10]
   }],
  Transpose[
   {{ (* transpose function invocation oriented arguments to function slot orientation *)
     Graph[(* 9.1a *)
      {A, B, C, D, E, Z},
      {A ↔ D, A ↔ C, B ↔ E, B ↔ D, C ↔ E, Z ↔ A, Z ↔ B, Z ↔ C, Z ↔ D, Z ↔ E},
      VertexLabels → "Name"],
     Append[CirclePoints[5], {0, 0}],
     Permute[Append[CirclePoints[5], {0, 0}], Cycles[{{1, 5, 2, 3}}]]
    }, {
     Graph[ (* 9.1b *)
      Range[10],
      {1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 1, 5 ↔ 6, 6 ↔ 7, 7 ↔ 8,
       8 ↔ 5, 1 ↔ 5, 2 ↔ 6, 3 ↔ 7, 4 ↔ 8, 1 ↔ 9, 4 ↔ 9, 2 ↔ 10, 3 ↔ 10, 9 ↔ 10},
      EdgeShapeFunction → Function[
        {pts, e},
        Module[
         {controlPts},
         controlPts = pts /. {a_, b_} :> {a, {6 a[[1]], a[[2]]}, {6 b[[1]], b[[2]]}, b};
         If[
          pts[[1]][[2]] == -pts[[2]][[2]] && Abs[pts[[1]][[1]] ] == 1 && Abs[pts[[1]][[2]] ] == 5,
          BezierCurve[controlPts],
          Line[pts]]]],
     {{-3, -2}, {+3, -2}, {+3, +2},
      {-3, +2}, {-1, -1}, {+1, -1}, {+1, +1}, {-1, +1}, {-2, 0}, {+2, 0}},
     {{-3, -2}, {+3, -2}, {+3, +2}, {-3, +2}, {-1, -5}, {+1, -5},
      {+1, +5}, {-1, +5}, {-2, 0}, {+2, 0}}
    }, {
     Graph[ (* 9.1c *)
      Range[6],
      {1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 5 ↔ 6, 6 ↔ 1, 2 ↔ 6, 2 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4}],
     {{-2, 0}, {-1, +1}, {+1, +1}, {+2, 0}, {+1, -1}, {-1, -1}},
     {{-2, 0}, {-1, +1}, {+1, 0}, {+2, 0}, {+1, -1}, {-1, 0}}
    }}
  ]]]
```



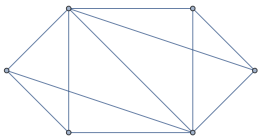


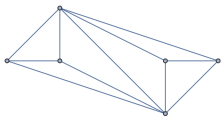
6





9





7

9.2

The Euler Identity must hold:

```
Reduce[{n - m + r == 2, n == 12, m == 1/2 k n, r == 8}]
```

```
r == 8 && n == 12 && m == 18 && k == 3
```

9.3

Using Theorem 2:

```
Reduce[{m ≤ 3 n - 6, n == 7, m == Total[{3, 4, 4, 4, 5, 6, 6}]}]
```

```
False
```

```
Reduce[{m ≤ 3 n - 6, n == 12, m == Total[{4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7}]}]
```

```
False
```

9.4a

For $n \geq 3$, Theorem 2 must hold, with $m = \frac{1}{2} n(n-1)$ for complete graphs:

```
Reduce[{m ≤ 3 n - 6, m == 1/2 n (n - 1)}]
```

$3 \le n \le 4 \&\& m == \frac{1}{2}\left(-n + n^2\right)$

K_3 and K_4 are readily verified as planar; for the cases of $n < 3$, K_1 and K_2 are also obviously planar.

9.4b

Theorem 2 doesn't give a meaningful bound:

```
Reduce[{m ≤ 3 n - 6, n == r + s, m == r s}, Reals]
```

$\left(\left(s < 3 \&\& r \ge \frac{-6 + 3 s}{-3 + s}\right) || s == 3 || \left(s > 3 \&\& r \le \frac{-6 + 3 s}{-3 + s}\right)\right) \&\& n == r + s \&\& m == r s$

Note that any $K_{r,s}$: $r, s \ge 3$ however contains a subdivision of $K_{3,3}$ and is therefore nonplanar. All the other bipartite graphs ($K_{2,3}$, $K_{2,2}$, $K_{1,3}$, $K_{1,2}$, and $K_{1,1}$) are easily verified to be planar.

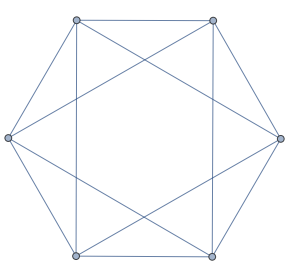
9.5a

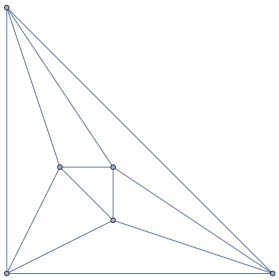
Such a graph must have at least 6 vertices:

```
Reduce[{m ≤ 3 n - 6, m == 1/2  4 n}]
```

$n \ge 6 \&\& m == 2 n$

```
GraphicsRow[
 #[
    Graph[{1 ↔ 2, 1 ↔ 4, 1 ↔ 5, 1 ↔ 6, 2 ↔ 3, 2 ↔ 4, 2 ↔ 6, 3 ↔ 4, 3 ↔ 5, 3 ↔ 6, 4 ↔ 5, 5 ↔ 6}]
   ] & /@
  {Show, Graph[#, GraphLayout → "PlanarEmbedding"] &,
   PlanarGraphQ, AllTrue[VertexDegree[#], EqualTo[4]] &},
 ImageSize →
  Full]
```





True

True

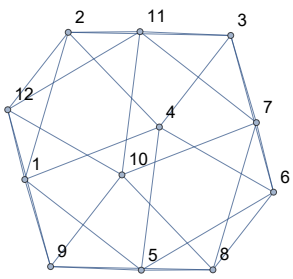
9.5b

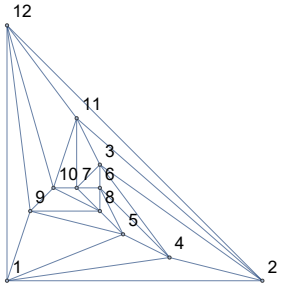
Such a graph must have at least 12 vertices:

```
Reduce[{m ≤ 3 n - 6, m == 1/2  5 n}]
```

$n \geq 12 \&\& m == \frac{5\,n}{2}$

```
GraphicsRow[
 #[
    Graph[
     Flatten[
      MapThread[
       Table[i + 1 ⟷ Mod[i + #2, 12] + 1, {i, #1, 11, #3}] &,
       Transpose[{{0, 1, 1}, {1, 2, 2}, {0, 3, 4}, {2, 3, 4}, {0, 4, 4}, {2, 4, 4}}]]],
     VertexLabels → "Name"]] & /@
  {Show, Graph[#, GraphLayout → "PlanarEmbedding"] &,
   PlanarGraphQ, AllTrue[VertexDegree[#], EqualTo[5]] &},
 ImageSize →
  Full]
```





True True

9.5c

Because by Corollary 3, each planar graph must contain a vertex of degree less than 6.

9.6

a. True

b. False

c. False: consider a maximal nonplanar graph

d. False: might still contain a *subdivision* of K_5 or $K_{3,3}$

e. False: consider $K_{3,3}$: $m \leq 3\,n - 6 \Leftarrow 3 \cdot 3 \leq 3 \cdot 6 - 6 \Leftarrow 9 \leq 12$ but is nonplanar

f. False: add a single triangle to $K_{3,3}$, is nonplanar

9.7

a. K_4

b. Does not exist: even K_4 is planar

c. K_5 with a single subdivided edge is nonplanar but does not contain K_5 (or $K_{3,3}$) as a subgraph

d. Since K_5 has $\frac{1}{2} \cdot 5 \cdot 4 = 10$ edges and is nonplanar, and there is no other graph with the same size and order, this is impossible

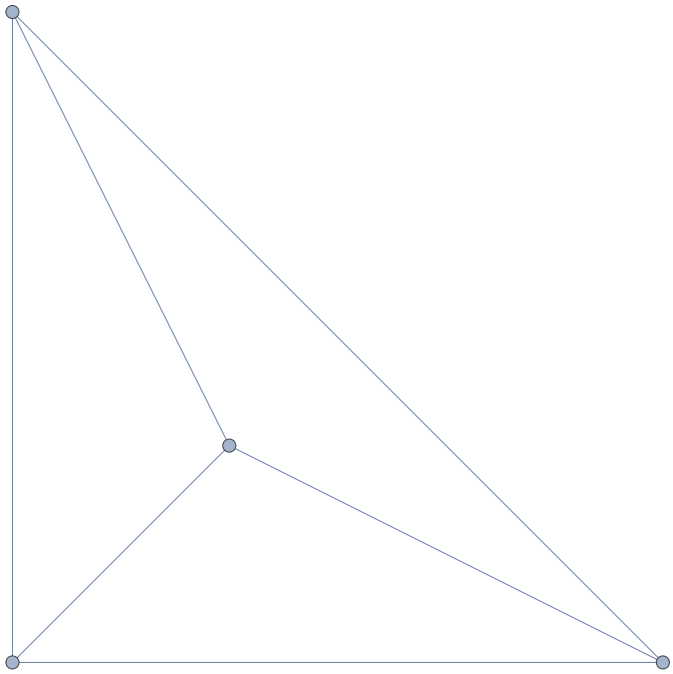
e. K_3 has order 3 and size $3 \cdot 3 - 6 = 3$, and is planar

f. $K_{3,3}$ has order 6 and size 9 and is nonplanar. Add any three edges to it; the resulting graph is still nonplanar.

9.8

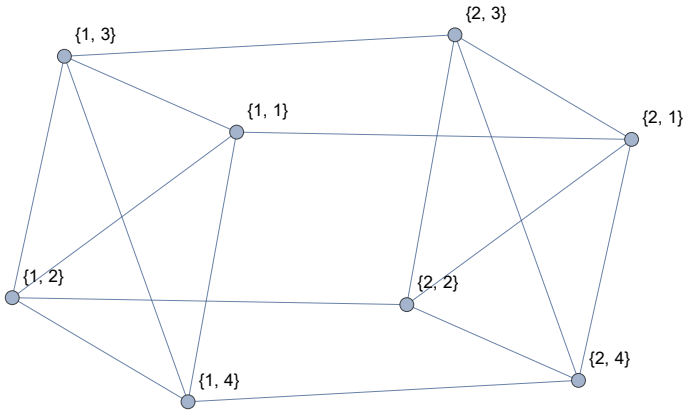
K_4 can be drawn planar, but one of the four vertices will always be enclosed in the interior:

```
CompleteGraph[4, GraphLayout → "PlanarEmbedding"]
```



That means that in the Cartesian product, that vertex cannot be joined to the 'other' K_4 without crossing an edge.

```
Labeled[#, "Planar: " <> ToString[PlanarGraphQ[#]]] & @
  GraphProduct[CompleteGraph[2], CompleteGraph[4], VertexLabels → "Name"]
```

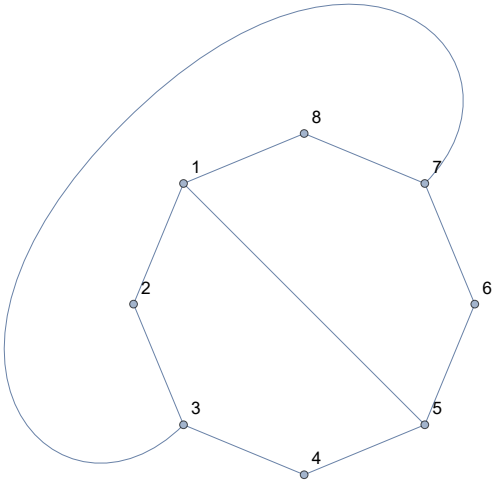



```
Planar: False
```

9.9

Consider the below subgraph. The edges 2⟷6 and 4⟷8 can only be drawn by crossing the cycle, the inner edge, or the outer edge; thus nonplanar.

```
EdgeAdd[
 CycleGraph[8,
  VertexLabels → "Name",
  EdgeShapeFunction → Function[
    {p, e},
    If[
     List @@ e == {3, 7}, (* can't compare UndirectedEdge? *)
     Module[{T = TranslationTransform[RotationTransform[π/2][3/2 p[[2]]]]},
      BezierCurve[{
        p[[1]], 2 p[[1]],
        T[2 p[[1]]],
        T[{0, 0}],
        T[2 p[[2]]],
        2 p[[2]], p[[2]]}]],
     Line[p]
    ]]],
 {1 ⟷ 5, 3 ⟷ 7}]
```



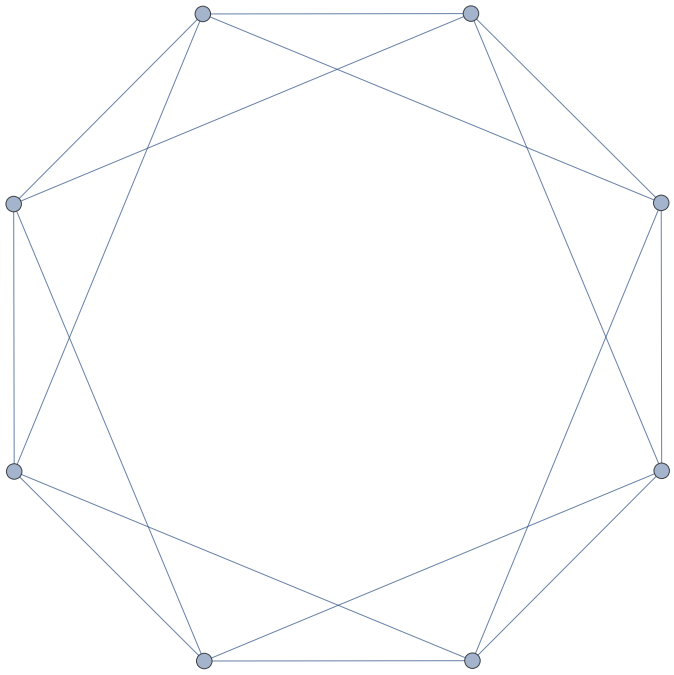
9.10

The graph has a subgraph containing a subdivision of K_5 by removing b, a, v, x, z, s and so is nonplanar.

9.11

The edges z⟷t, t⟷v, v⟷x, and x⟷z can be drawn on the outside without crossing any other edges, resulting in a planar drawing.

```
Labeled[#, "Planar: " <> ToString[PlanarGraphQ[#]]] & @
 Graph[{y ⟷ z, z ⟷ s, s ⟷ t, t ⟷ u, u ⟷ v, v ⟷ w,
   w ⟷ x, x ⟷ y, y ⟷ s, s ⟷ u, u ⟷ w, w ⟷ y, z ⟷ t, t ⟷ v, v ⟷ x, x ⟷ z}]
```

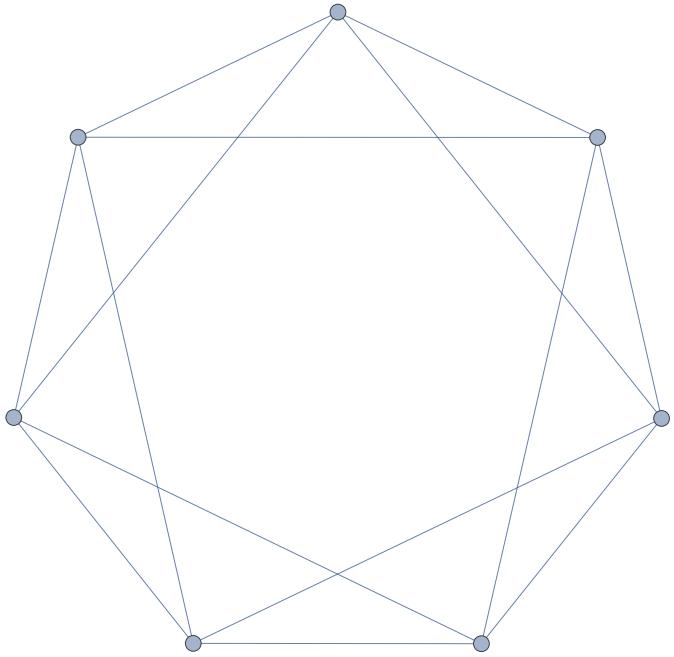


```
Planar: True
```

9.12

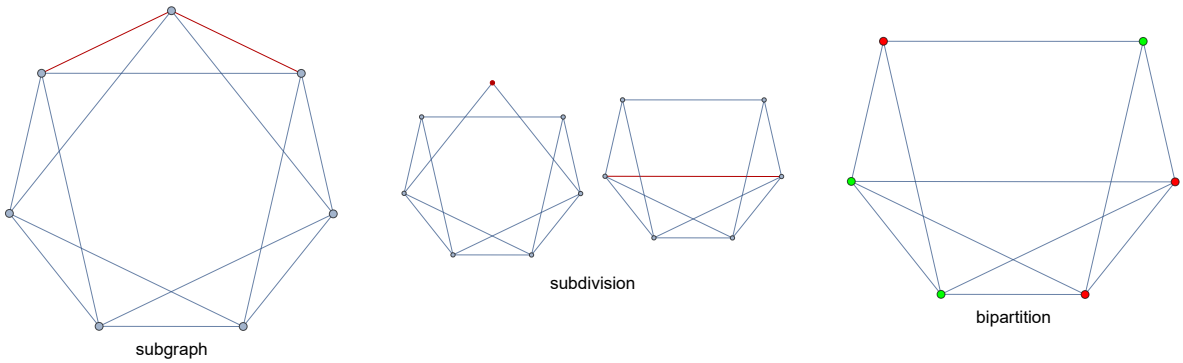
The graph seems intuitively nonplanar: the inside cycle can't be 'untangled' from the outer one:

```
Labeled[#, "Planar: " <> ToString[PlanarGraphQ[#]]] & @
 Graph[
  {y ↔ z, z ↔ t, t ↔ u, u ↔ v, v ↔ w, w ↔ x, x ↔ y, y ↔ t, t ↔ v, v ↔ x, x ↔ z, z ↔ u, u ↔ w, w ↔ y}]
```



By Kuratowski it should contain a subgraph that is a subdivision of K_5 or $K_{3,3}$. Note that the graph has 7 vertices of degree 4; to reduce it to K_5 we would have to remove two vertices by undoing subdivision leaving vertices with degree 3, which could thus not result in K_5. The graph must therefore contain a subdivision of $K_{3,3}$:

```
Module[
 {g0, g1, g2},
 g0 = Graph[{y ↔ z, z ↔ t, t ↔ u, u ↔ v, v ↔ w,
    w ↔ x, x ↔ y, y ↔ t, t ↔ v, v ↔ x, x ↔ z, z ↔ u, u ↔ w, w ↔ y}];
 g1 = Graph[EdgeDelete[g0, {x ↔ w, w ↔ v}], VertexCoordinates → GraphEmbedding[g0]];
 g2 = EdgeAdd[VertexDelete[g1, w], {u ↔ y}];
 GraphicsRow[{
   Labeled[HighlightGraph[g0, {x ↔ w, w ↔ v}], "subgraph"],
   Labeled[
    GraphicsRow[{
      HighlightGraph[g1, {w}],
      HighlightGraph[g2, {u ↔ y}]},
     ImageSize → Full],
    "subdivision"],
   Labeled[Graph[g2,
     VertexStyle → {x → Red, t → Red, u → Red, y → Green, z → Green, v → Green}], "bipartition"]},
  ImageSize → Full
 ]]
```




```
PlanarGraphQ[
 EdgeDelete[
  CompleteGraph[7, VertexLabels → "Name"],
  EdgeList[CycleGraph[7]]]]
```

```
False
```

9.13a

Following the proof of Theorem 2: since all cycles are of minimum length 4, $M \geq 4r$. With $M \leq 2m$ we have $4r \leq 2m \Rightarrow 2r \leq m$. Substituting into the Euler Identity $2 = n - m + r \leq n - m + \frac{1}{2}m = n - \frac{1}{2}m \Rightarrow \frac{1}{2}m \leq n - 2 \Rightarrow m \leq 2n - 4$.

9.13b

Since $K_{3,3}$ is bipartite it contains no triangles. So $m \leq 2n - 4 \Rightarrow 3 \cdot 3 \leq 2 \cdot 6 - 4 \Rightarrow 9 \leq 8$, which is a contradiction.

9.13c

Suppose all vertices are of degree at least 4; then $m \geq \frac{1}{2}\ 4\,n = 2\,n$. But to be planar, $m \leq 2\,n - 4$. (In any case, we already knew that a planar bipartite graph must have a vertex of degree less than 3.)

9.14a

Following the proof of Theorem 2, since all cycles are of minimum length 5, $M \geq 5\,r$. With $M \leq 2\,m$ we have $5\,r \leq 2\,m \Rightarrow r \leq \frac{2}{5}\,m$. Substituting into the Euler Identity $2 = n - m + r \leq n - m + \frac{2}{5}\,m = n - \frac{3}{5}\,m$.

9.14b

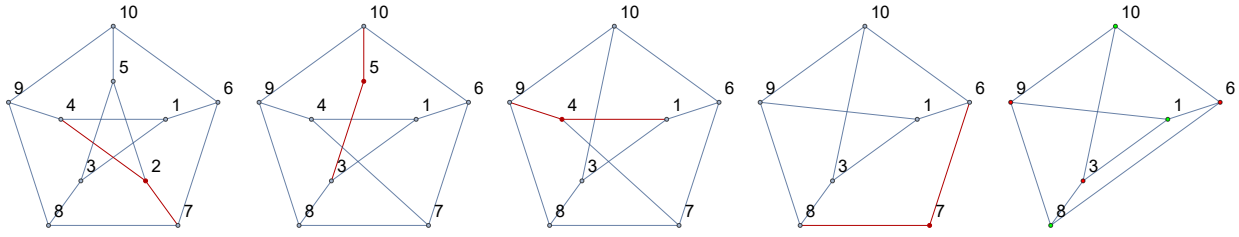
```
Reduce[{m ≤ 5/3 (n - 2), n == 10, m == 15}]
```

```
False
```

9.14c

Contains a subdivision of $K_{3,3}$:

```
Module[{g0, g1, g2, g3, g4},
 g0 = PetersenGraph[VertexLabels → "Name"];
 g1 = EdgeAdd[VertexDelete[g0, 2], 4 ↔ 7];
 g2 = EdgeAdd[VertexDelete[g1, 5], 3 ↔ 10];
 g3 = EdgeAdd[VertexDelete[g2, 4], 9 ↔ 1];
 g4 = EdgeAdd[VertexDelete[g3, 7], 8 ↔ 6];
 GraphicsRow[{
   HighlightGraph[g0, {2, 4 ↔ 2, 2 ↔ 7}],
   HighlightGraph[g1, {5, 3 ↔ 5, 5 ↔ 10}],
   HighlightGraph[g2, {4, 9 ↔ 4, 4 ↔ 1}],
   HighlightGraph[g3, {7, 8 ↔ 7, 7 ↔ 6}],
   Graph[g4, VertexStyle → {9 → Red, 3 → Red, 6 → Red, 8 → Green, 1 → Green, 10 → Green}]
  },
  ImageSize → Full]]
```



9.14d

9.15

Suppose the graph is planar but has no vertex of degree less than 5, then $n \geq 12$; so if a graph is planar with $n < 12$, then it must have a vertex of degree less than 5.

```
Reduce[{m ≤ 3 n - 6, m ≥ 1/2  5 n}]
```

```
(n == 12 && m == 30) || (n > 12 && 5 n/2 ≤ m ≤ -6 + 3 n)
```

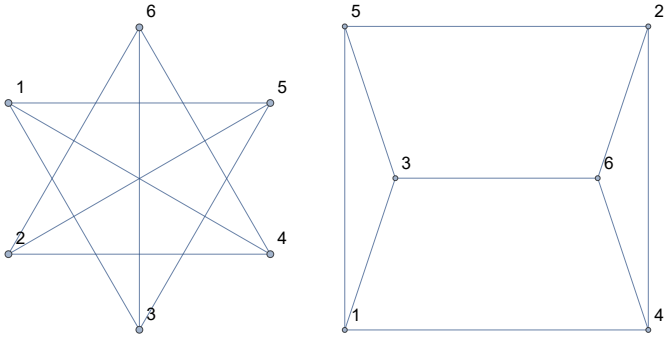
9.16

Consider the subdivision of K_5 with a single added vertex on some edge; this is clearly nonplanar, and the removal of any single edge results in a maximally planar graph. Removing an edge incident to the subdivision and removing any other edge therefore results in two nonisomorphic maximally planar graphs.

9.17

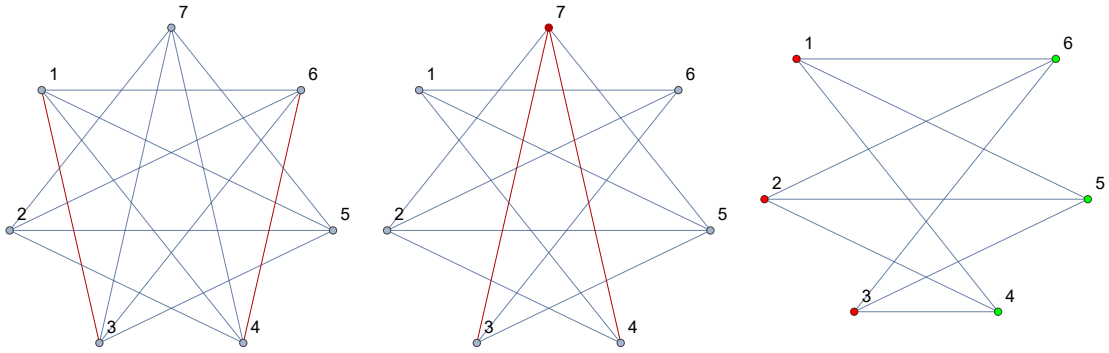
$\overline{C_6}$ is planar:

```
Module[
 {g = GraphComplement[CycleGraph[6], VertexLabels → "Name"]},
 GraphicsRow[{
   g,
   Graph[g, VertexCoordinates → {{-3, -3}, {+3, +3}, {-2, 0}, {+3, -3}, {-3, +3}, {+2, 0}}]},
  ImageSize → Medium]]
```



$\overline{C_7}$ contains a subgraph that is a subdivision of $K_{3,3}$:

```
Module[
 {g0, g1, g2},
 g0 = GraphComplement[CycleGraph[7], VertexLabels → "Name"];
 g1 = Graph[EdgeDelete[g0, {1 ↔ 3, 4 ↔ 6}]];
 g2 = Graph[EdgeAdd[VertexDelete[g1, 7], 3 ↔ 4],
   VertexCoordinates → Drop[GraphEmbedding[g1], {7}]];
 GraphicsRow[{
   HighlightGraph[g0, {1 ↔ 3, 4 ↔ 6}],
   HighlightGraph[g1, {7, 3 ↔ 7, 7 ↔ 4}],
   Graph[g2, VertexStyle → {1 → Red, 2 → Red, 3 → Red, 4 → Green, 5 → Green, 6 → Green}]},
  ImageSize → Large]]
```



$\overline{C_8}$ has too many edges to be planar:

9.18

For G to be maximally planar it must be one edge away from $K_{3,3}$: that is, 5 vertices must have degree 3 and the remainder could have at most degree 2; and for it to be planar, the inequality of Theorem 2 must hold as well:

In[10]:=
```
Reduce[{m < 3 n - 6, m ≥ 5 * 3 + (n - 5) * 2}]
```

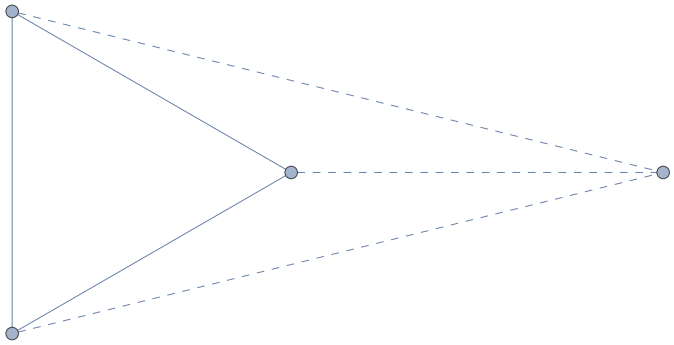
Out[10]= $n > 11 \&\& 5 + 2\,n \leq m < -6 + 3\,n$

9.19

We know that for a planar graph $m \leq 3\,n - 6$. Construct a graph for which $m = 3\,n - 6$ by induction: for $n = 3$, K_3 satisfies the equality. Add a vertex and connect it to all three vertices on the outside of the graph; this encloses one of those vertices, but exposes the added vertex; resulting in a new graph with three exterior vertices, one added vertex and three added edges and thus also satisfies the equality. Adding one edge to this graph violates the inequality and therefore results in a nonplanar graph; thus any graph constructed this way is maximally planar.

In[27]:=
```
Graph[{G1 ⟷ G2, G2 ⟷ G3, G3 ⟷ G1, v ⟷ G1, v ⟷ G2, v ⟷ G3},
 VertexCoordinates → Append[CirclePoints[{1, 0}, 3], {3, 0}],
 EdgeStyle → Table[v ⟷ Gi → Dashed, {i, 3}]]
```

Out[27]=



In[32]:=
```
Reduce[{m == 3 n - 6, n - m + r == 2, n == 100}]
```

Out[32]=
```
r == 196 && n == 100 && m == 294
```

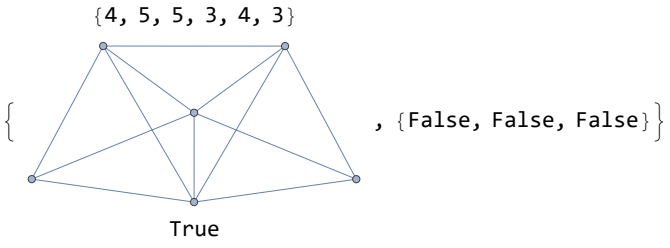
9.20

In[35]:=
```
Reduce[{m == 3 n - 6, m == 1/2 * 1/3 (3 + 4 + 5) n}]
```

Out[35]=
```
n == 6 && m == 12
```

In[52]:=
```
Module[
 {g = Graph[{1 ⟷ 2, 1 ⟷ 5, 1 ⟷ 6, 1 ⟷ 3, 2 ⟷ 3, 2 ⟷ 4, 2 ⟷ 5, 2 ⟷ 6, 3 ⟷ 4, 3 ⟷ 5, 4 ⟷ 5, 5 ⟷ 6}]},
 {Labeled[g, {VertexDegree[g], PlanarGraphQ[g]}, {Top, Bottom}],
  PlanarGraphQ[EdgeAdd[g, #]] & /@ EdgeList[GraphComplement[g]]}]
```

Out[52]=



9.21

(Following the proof in the solutions.) Suppose G is maximal planar with a vertex v, $\deg v = 2$; so $m = 3n - 6$ (while I understand that there is *a* maximal graph with that size, I don't know that *every* maximal planar graph has that size). Consider $G' = G - v$; then:

In[64]:=
```
Reduce[{m' ≤ 3 n' - 6 /. {m' → m - 2, n' → n - 1}, m == 3 n - 6}]
```

Out[64]=
```
False
```

9.22

In[65]:=
```
Reduce[{m == 3 n - 6, m == 1/2 * 4 n}]
```

Out[65]= n == 6 && m == 12

9.2 Embedding Graphs on Surfaces

```
ParametricPlot3D[
 {(2 + Cos[ν]) Cos[μ], (2 + Cos[ν]) Sin[μ], Sin[ν]},
 {ν, 0, 2 π}, {μ, 0, 2 π}]
```