I am using material from "Introduction to Maple's Graph Theory Package" and the Help files at MapleSoft

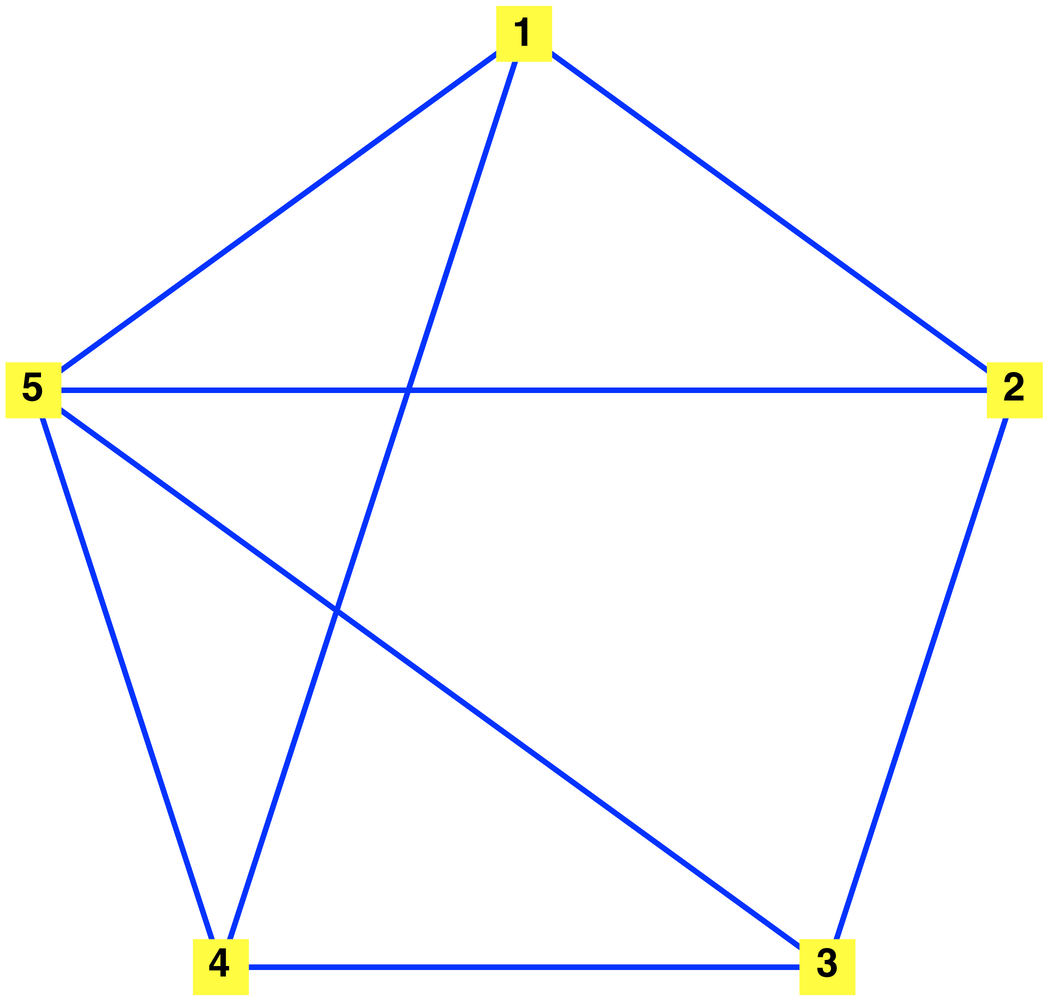
```
> restart
> with(GraphTheory)
```

```
[AcyclicPolynomial, AddArc, AddEdge, AddVertex, AdjacencyMatrix, AllPairsDistance, Arrivals,        (1)
    ArticulationPoints, BellmanFordAlgorithm, BiconnectedComponents, BipartiteMatching,
    Blocks, CartesianProduct, CharacteristicPolynomial, ChromaticIndex, ChromaticNumber,
    ChromaticPolynomial, CircularChromaticIndex, CircularChromaticNumber,
    CircularEdgeChromaticNumber, CliqueNumber, CompleteGraph, ConnectedComponents,
    Contract, ConvertGraph, CopyGraph, CycleBasis, CycleGraph, Degree, DegreeSequence,
    DelaunayTriangulation, DeleteArc, DeleteEdge, DeleteVertex, Departures, Diameter,
    Digraph, DijkstrasAlgorithm, DiscardEdgeAttribute, DiscardGraphAttribute,
    DiscardVertexAttribute, DisjointUnion, Distance, DrawGraph, DrawNetwork, DrawPlanar,
    EdgeChromaticNumber, EdgeConnectivity, Edges, Embed, ExportGraph, FlowPolynomial,
    FundamentalCycle, GetEdgeAttribute, GetEdgeWeight, GetGraphAttribute,
    GetVertexAttribute, GetVertexPositions, Girth, Graph, GraphComplement, GraphEqual,
    GraphJoin, GraphNormal, GraphPolynomial, GraphPower, GraphRank, GraphSpectrum,
    GraphUnion, GreedyColor, HasArc, HasEdge, HighlightEdges, HighlightSubgraph,
    HighlightTrail, HighlightVertex, HighlightedEdges, HighlightedVertices, ImportGraph,
    InDegree, IncidenceMatrix, IncidentEdges, IndependenceNumber, InducedSubgraph,
    IsAcyclic, IsBiconnected, IsBipartite, IsClique, IsConnected, IsCutSet, IsDirected,
    IsEdgeColorable, IsEulerian, IsForest, IsGraphicSequence, IsHamiltonian, IsIntegerGraph,
    IsIsomorphic, IsNetwork, IsPlanar, IsRegular, IsStronglyConnected, IsTournament, IsTree,
    IsTwoEdgeConnected, IsVertexColorable, IsWeighted, IsomorphicCopy, KruskalsAlgorithm,
    LaplacianMatrix, Latex, LineGraph, ListEdgeAttributes, ListGraphAttributes,
    ListVertexAttributes, MakeDirected, MakeWeighted, MaxFlow, MaximumClique,
    MaximumDegree, MaximumIndependentSet, MinimalSpanningTree, MinimumDegree,
    Mycielski, Neighborhood, Neighbors, NonIsomorphicGraphs, NumberOfEdges,
    NumberOfSpanningTrees, NumberOfVertices, OddGirth, OutDegree, PathGraph,
    PermuteVertices, PlaneDual, PrimsAlgorithm, RandomGraphs, RankPolynomial,
    RelabelVertices, ReliabilityPolynomial, SHARCorder, SeidelSpectrum, SeidelSwitch,
    SequenceGraph, SetEdgeAttribute, SetEdgeWeight, SetGraphAttribute, SetVertexAttribute,
    SetVertexPositions, ShortestPath, SpanningPolynomial, SpanningTree, SpecialGraphs,
    StronglyConnectedComponents, Subdivide, Subgraph, TensorProduct, TopologicSort, Trail,
    TravelingSalesman, TreeHeight, TuttePolynomial, TwoEdgeConnectedComponents,
    UnderlyingGraph, VertexConnectivity, Vertices, WeightMatrix]
```

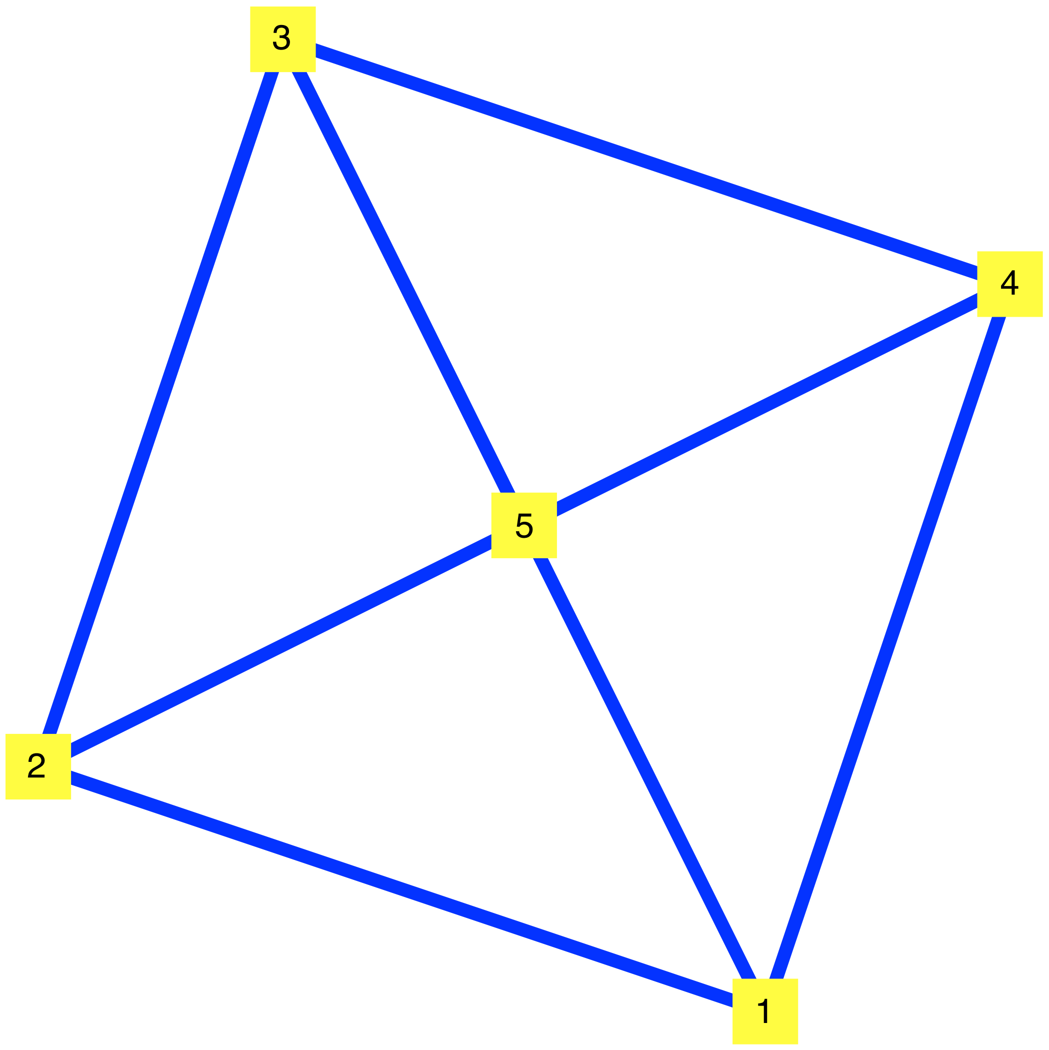
12a. We can create a graph with an adjacency list.

```
> adjacencyList12a := [[2, 4, 5], [3, 1, 5], [2, 4, 5], [1, 3, 5], [1, 2, 3, 4]]
      # note if is not symmetric it goes to a directed graph
```

```
                adjacencyList12a := [[2, 4, 5], [3, 1, 5], [2, 4, 5], [1, 3, 5], [1, 2, 3, 4]]                (2)
> G := Graph(adjacencyList12a)
              G := Graph 1: an undirected unweighted graph with 5 vertices and 8 edge(s)               (3)
> DrawGraph(G)
```



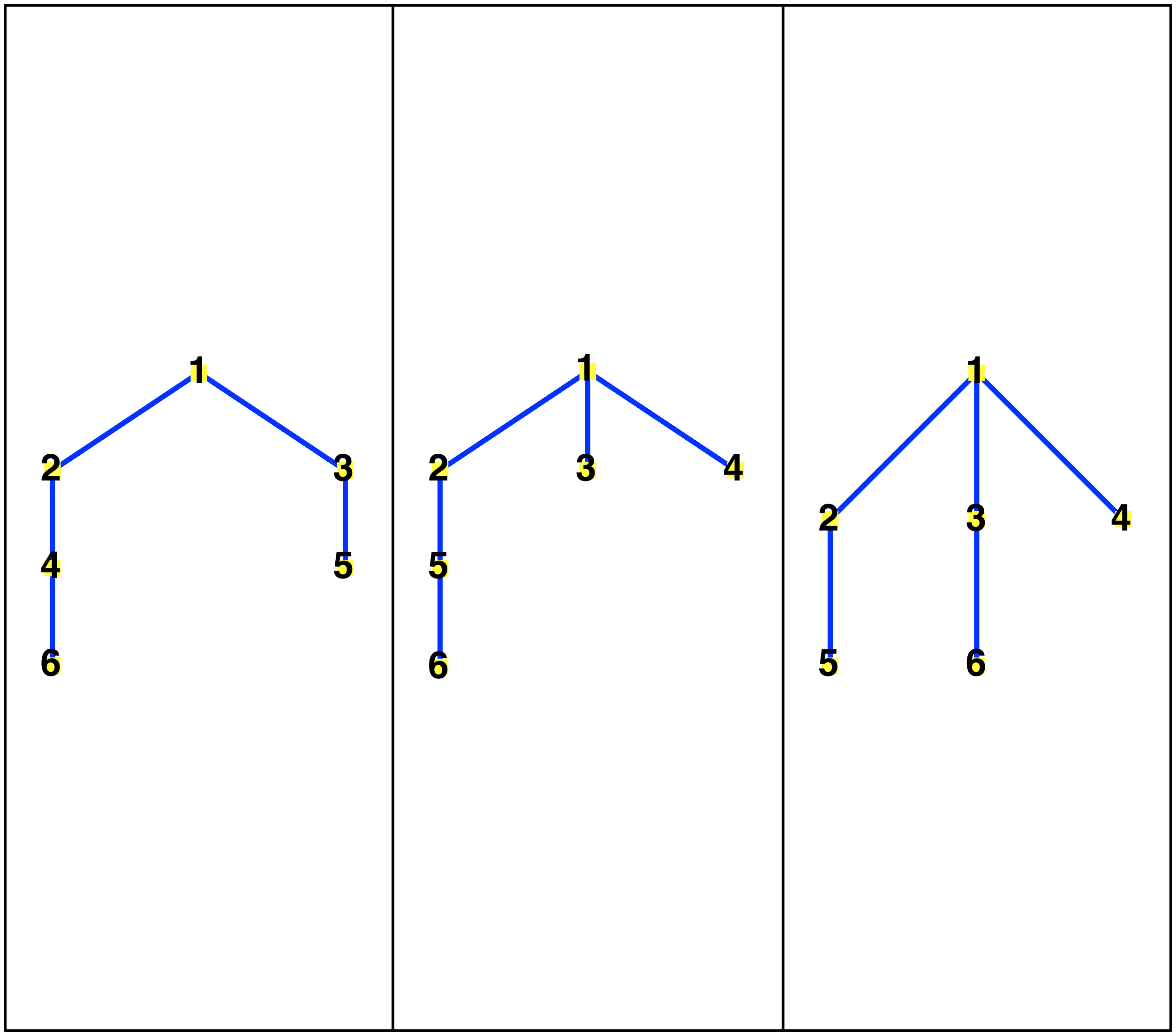
```
> DrawGraph(G, style = spring)
```

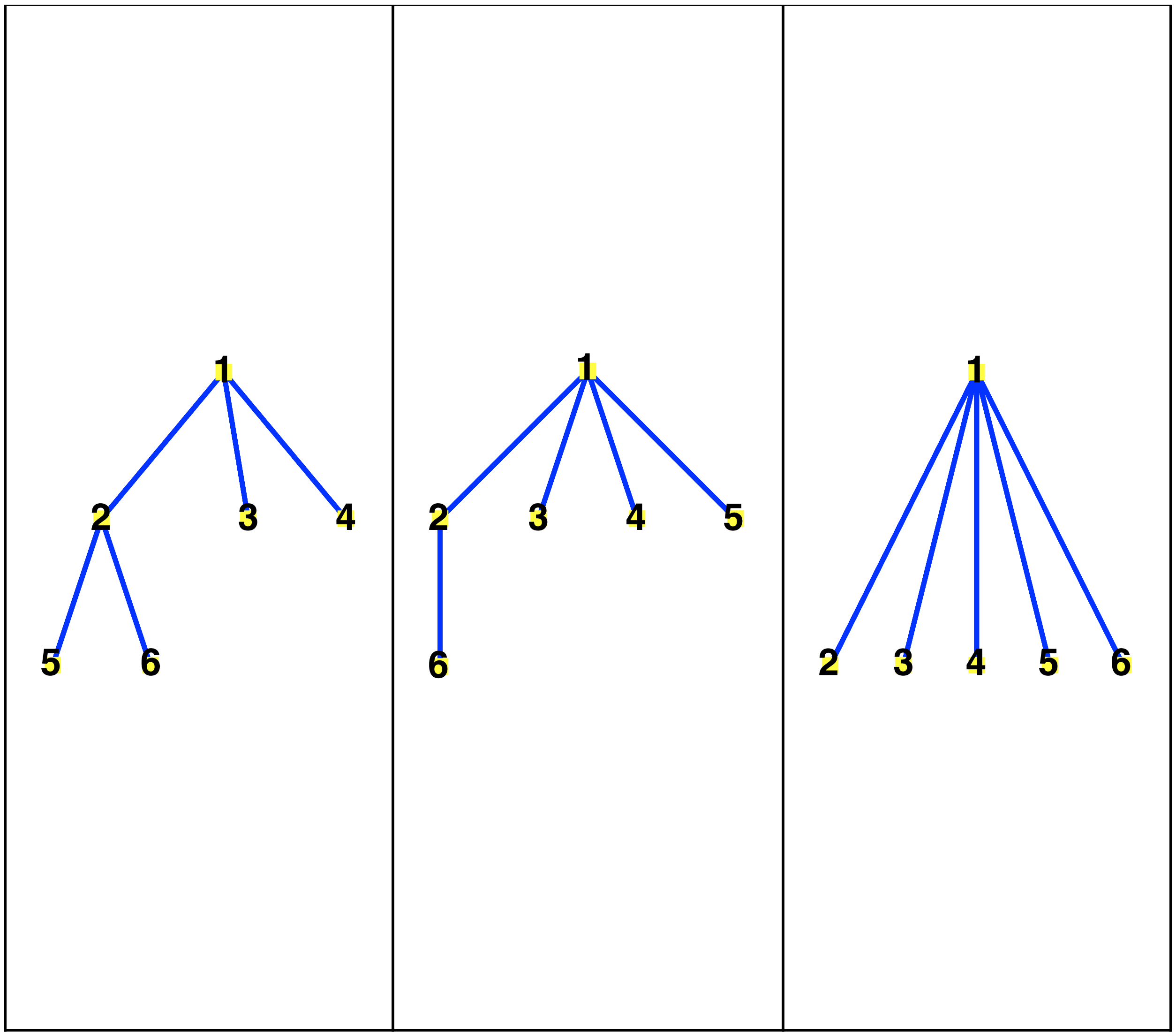


Note that these are the same graph but are drawn differently. These two graphs are isomorphic. There is a one-one onto f from the vertices of the first to the vertices of the second so that {a,b} is an edge iff {f(a),f(b)} is an edge. It is not always simple to decide if two graphs are isomorphic. In fact it is not known how hard it is. We will in general look for invariants like Degree to disprove graphs are not isomorphic. A related concept is the neighbors of vertices. For directed graphs there are arrivals and departures and outdegree and indegree.

```
> Degree(G, 4)#InDegree;OutDegree
                                   3                                        (4)
> Neighbors(G)#Arrivals,Departures
              [[2, 4, 5], [1, 3, 5], [2, 4, 5], [1, 3, 5], [1, 2, 3, 4]]    (5)
> T := NonIsomorphicGraphs(6, 5, restrictto = connected, output = graphs, outputform = graph)
T := Graph 10: an undirected unweighted graph with 6 vertices and 5 edge(s),   (6)
     Graph 11: an undirected unweighted graph with 6 vertices and 5 edge(s),
     Graph 12: an undirected unweighted graph with 6 vertices and 5 edge(s),
     Graph 13: an undirected unweighted graph with 6 vertices and 5 edge(s),
     Graph 14: an undirected unweighted graph with 6 vertices and 5 edge(s),
     Graph 15: an undirected unweighted graph with 6 vertices and 5 edge(s)
```

```
> DrawGraph([T])
```

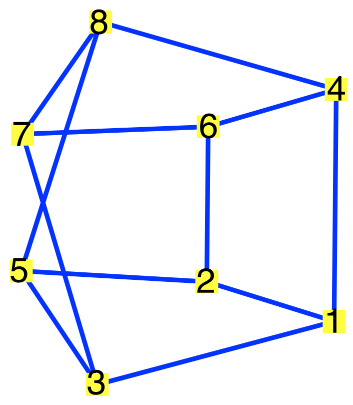
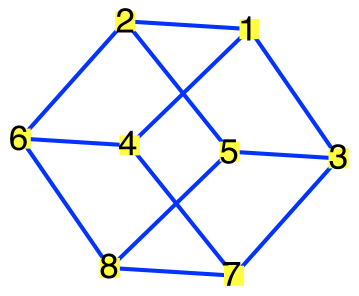
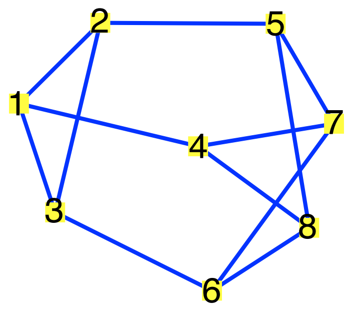


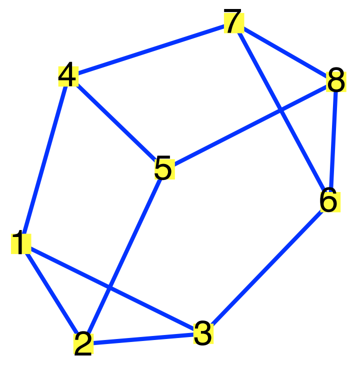


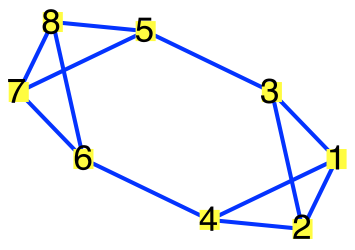
```
> R := NonIsomorphicGraphs(8, 12, restrictto = [connected, regular], output = graphs,
      outputform = graph)
```

```
R := Graph 16: an undirected unweighted graph with 8 vertices and 12 edge(s),          (7)
    Graph 17: an undirected unweighted graph with 8 vertices and 12 edge(s),
    Graph 18: an undirected unweighted graph with 8 vertices and 12 edge(s),
    Graph 19: an undirected unweighted graph with 8 vertices and 12 edge(s),
    Graph 20: an undirected unweighted graph with 8 vertices and 12 edge(s)
```

```
> DrawGraph([R], style = spring)
```

 8 4 7 6 5 2 1 3	 2 1 6 4 5 3 8 7	 2 5 1 7 4 3 8 6





We shall see that the sum of all the degrees gives twice the number of edges. Above was a list and drawing of nonisomorphic trees of size 6. Not so simple even for trees.Want something harder look at the more complicated example following usig list R of graphs even though each vertices had degree 3 (why)

We can also use an adjancency matrix. It is symmetric unless the graph is a digraph (directed graph)

```
> AdjacencyMatrix(G)
```

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \qquad \textbf{(8)}$$

12b.

```
> adjMatrix := Matrix([[0,1,1,1,1],[1,0,1,1,1],[1,1,0,0,1],[1,1,0,0,1],[1,1,1,1,0]])
```

> $adjMatrix := \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$

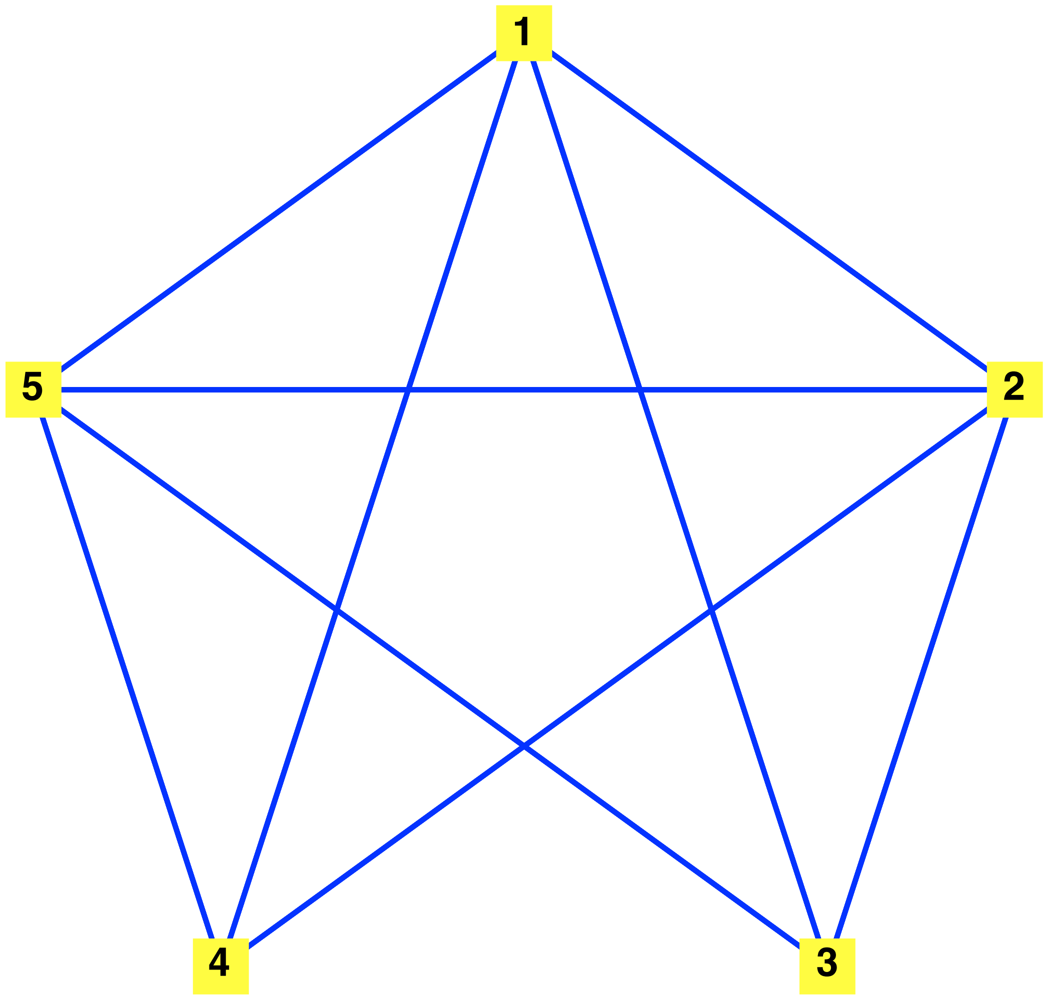
# note if not symmetric gives a directed graph. Use the Matrix and fill in values

$$adjMatrix := \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \qquad (9)$$

> *Gb* := *Graph*(*adjMatrix*)

Gb := *Graph 2: an undirected unweighted graph with 5 vertices and 9 edge(s)* **(10)**

> *DrawGraph*(*Gb*)



> *Edges*(*Gb*)

```
{{1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 3}, {2, 4}, {2, 5}, {3, 5}, {4, 5}}                (11)
```

```
> IncidenceMatrix(Gb)
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (12)$$

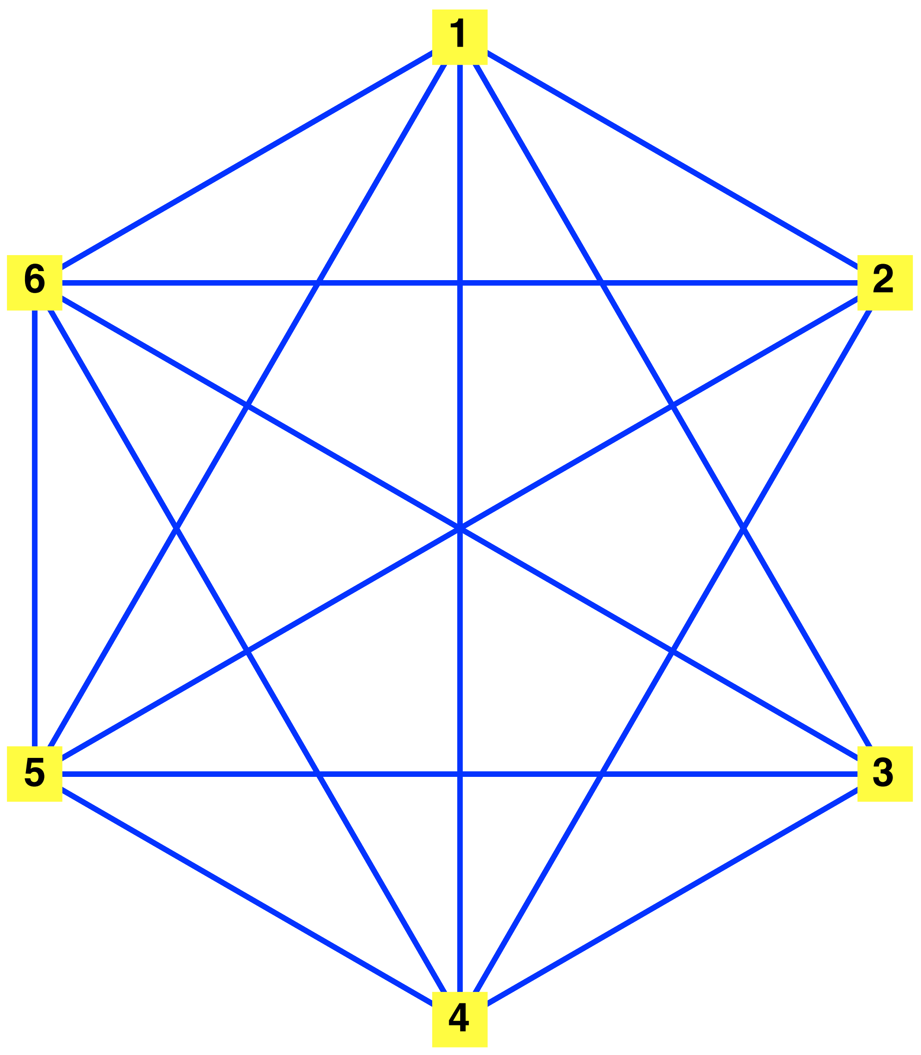
If we put symmetric weights in matrix we get a weighted graph. We can also use an IncidenceMatrix to create a graph. Rows are vertices and columns Edges. Note we have to order the edges and that is why I used the command Edges(Gb)

12c

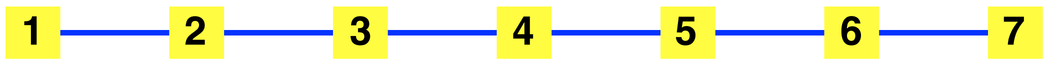
```
> edgeSet := {{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 4}, {2, 5}, {2, 6}, {3, 4}, {3, 5}, {3,
      6}, {4, 5}, {4, 6}, {5, 6}, {1, 2}}# note allows repition if forget
edgeSet := {{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 4}, {2, 5}, {2, 6}, {3, 4}, {3, 5}, {3,     (13)
    6}, {4, 5}, {4, 6}, {5, 6}}
```

```
> Gc := Graph(edgeSet)
        Gc := Graph 3: an undirected unweighted graph with 6 vertices and 14 edge(s)     (14)
```

```
> DrawGraph(Gc)
```



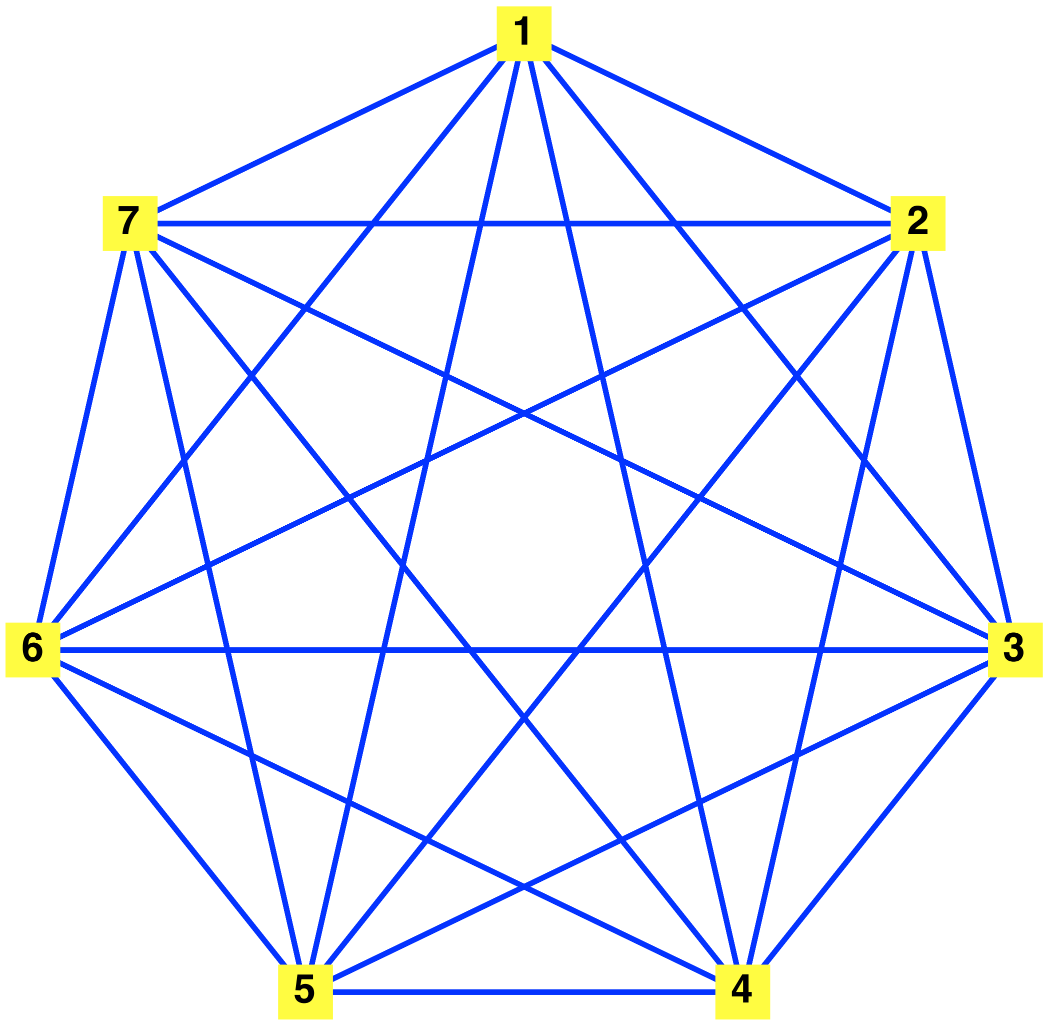
```
> with(SpecialGraphs)
[AntiPrismGraph, CageGraph, ClebschGraph, CompleteBinaryTree, CompleteKaryTree,          (15)
    CoxeterGraph, DesarguesGraph, DodecahedronGraph, DoubleStarSnark, DyckGraph,
    FlowerSnark, FosterGraph, GeneralizedBlanusaSnark, GeneralizedHexagonGraph,
    GeneralizedPetersenGraph, GoldbergSnark, GridGraph, GrinbergGraph, GrotzschGraph,
    HeawoodGraph, HerschelGraph, HoffmanSingletonGraph, HypercubeGraph,
    IcosahedronGraph, KneserGraph, LCFGraph, LeviGraph, McGeeGraph,
    MobiusKantorGraph, OctahedronGraph, OddGraph, PappusGraph, PayleyGraph,
    PetersenGraph, PrismGraph, RobertsonGraph, ShrikhandeGraph, SoccerBallGraph,
    StarGraph, SzekeresSnark, TetrahedronGraph, ThetaGraph, TorusGridGraph,
    Tutte8CageGraph, WebGraph, WheelGraph]
> P := PathGraph(7)
        P := Graph 4: an undirected unweighted graph with 7 vertices and 6 edge(s)      (16)
> DrawGraph(P)
```



```
> #DeleteEdge(P, {3, 4})
> #DrawGraph(P)
> #P1 := AddEdge(P, {1, 7}, inplace = false)
> #DrawGraph(P1)
> #DrawGraph(P)
> #SP := SpanningTree(P1)
> #DrawGraph(SP)
> #SPC := Contract(SP, {7, 1})
> #DrawGraph(SPC)
> #SPCI := InducedSubgraph(SPC, {1, 2, 3, 6})
> #DrawGraph(SPCI)
```

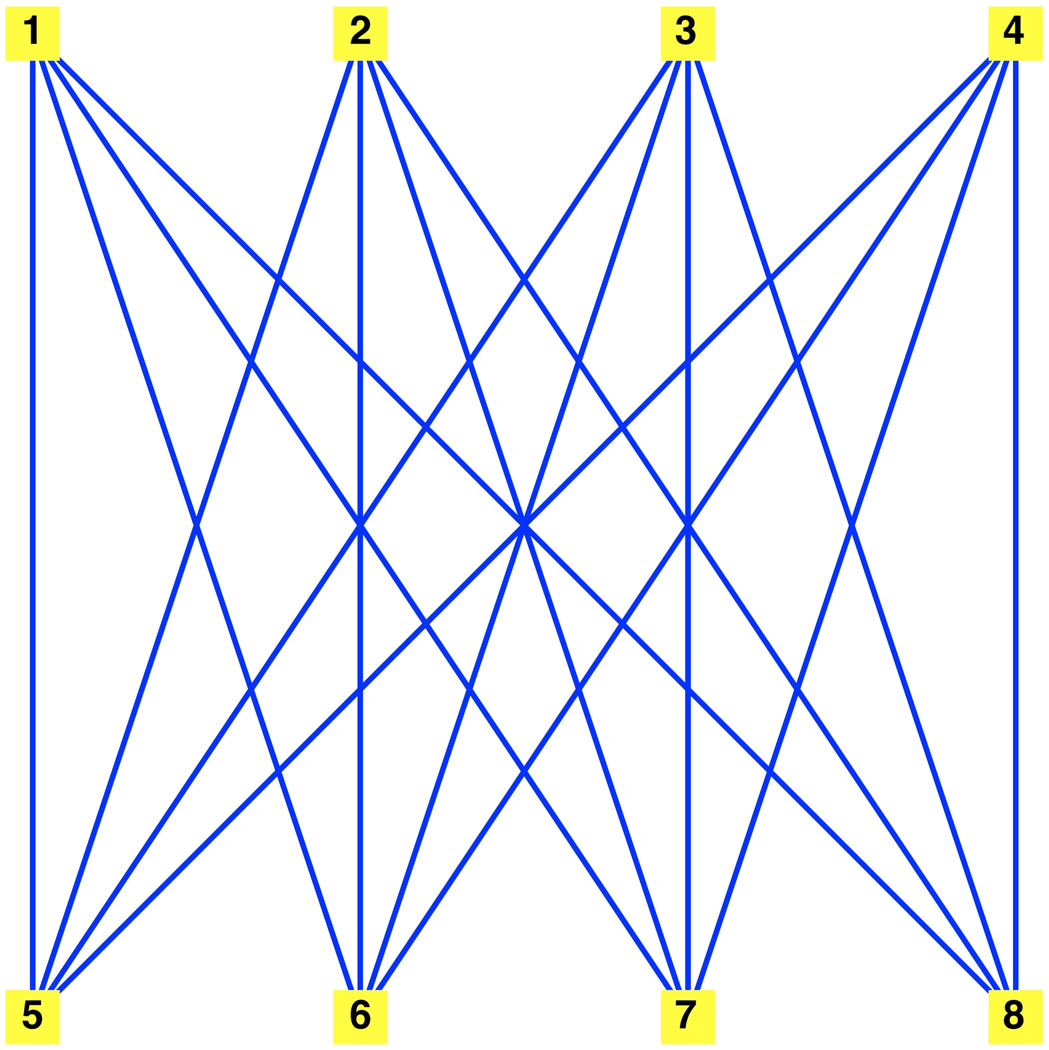
14a

```
> K7 := CompleteGraph(7)
        K7 := Graph 5: an undirected unweighted graph with 7 vertices and 21 edge(s)        (17)
> DrawGraph(K7)
```

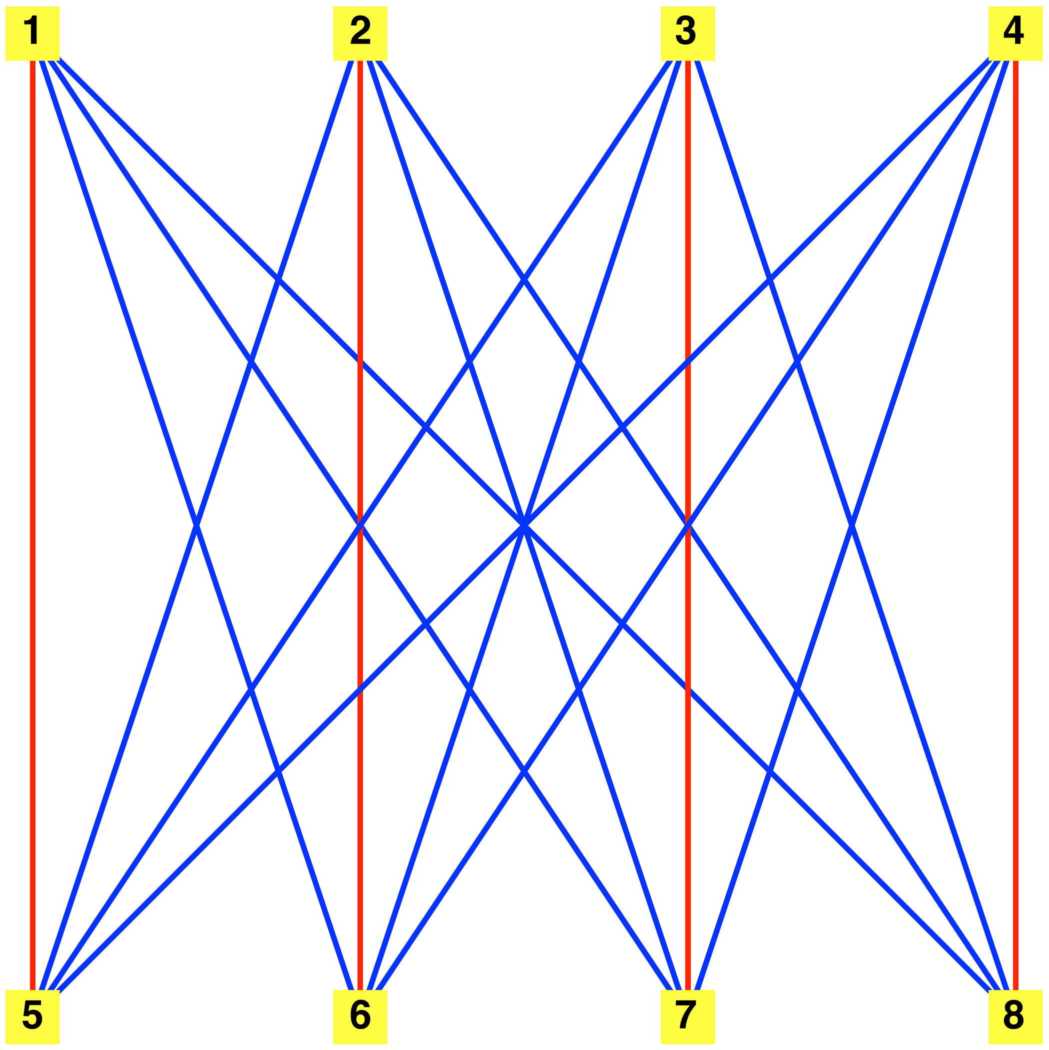


This is how we create a complete bipartite graph.

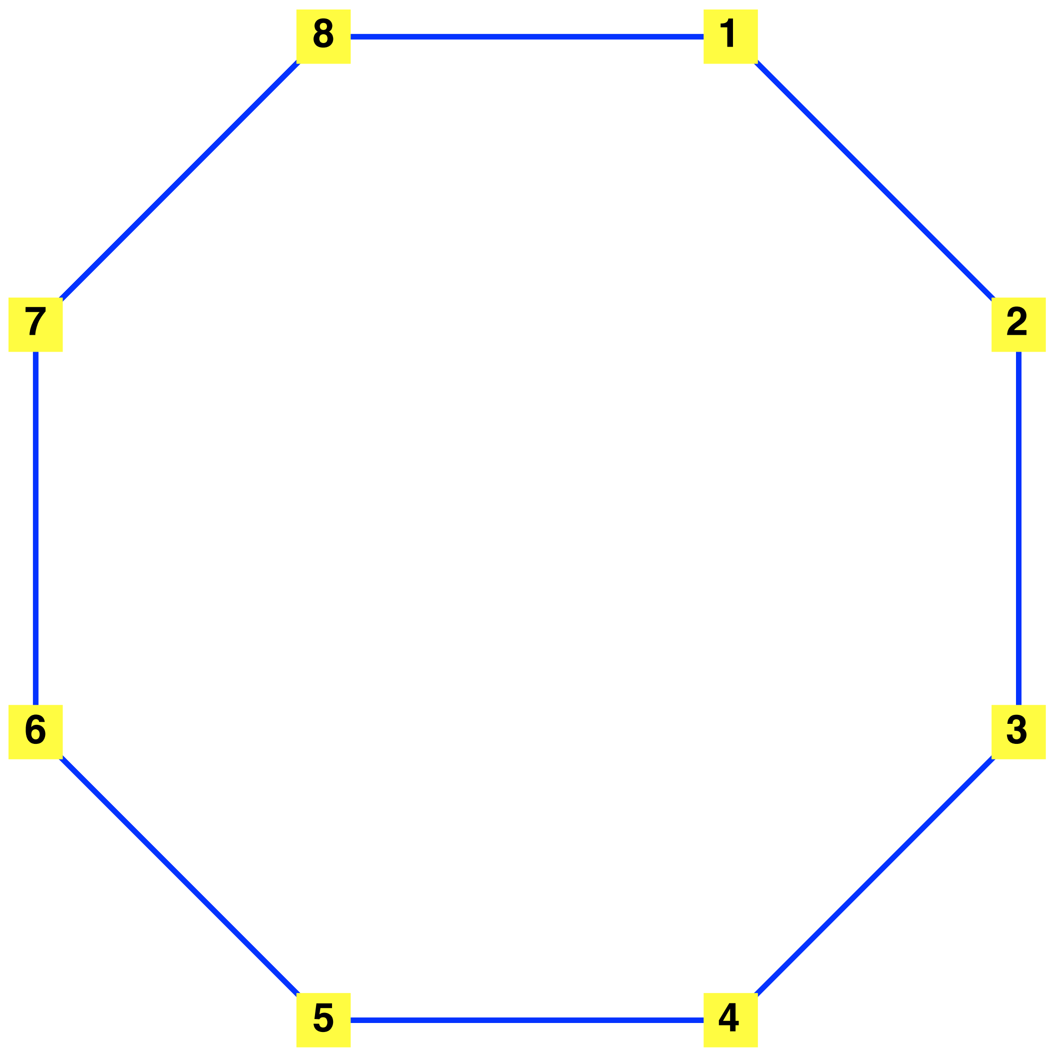
```
> K44 := CompleteGraph(4, 4)
        K44 := Graph 6: an undirected unweighted graph with 8 vertices and 16 edge(s)        (18)
> DrawGraph(K44)
```



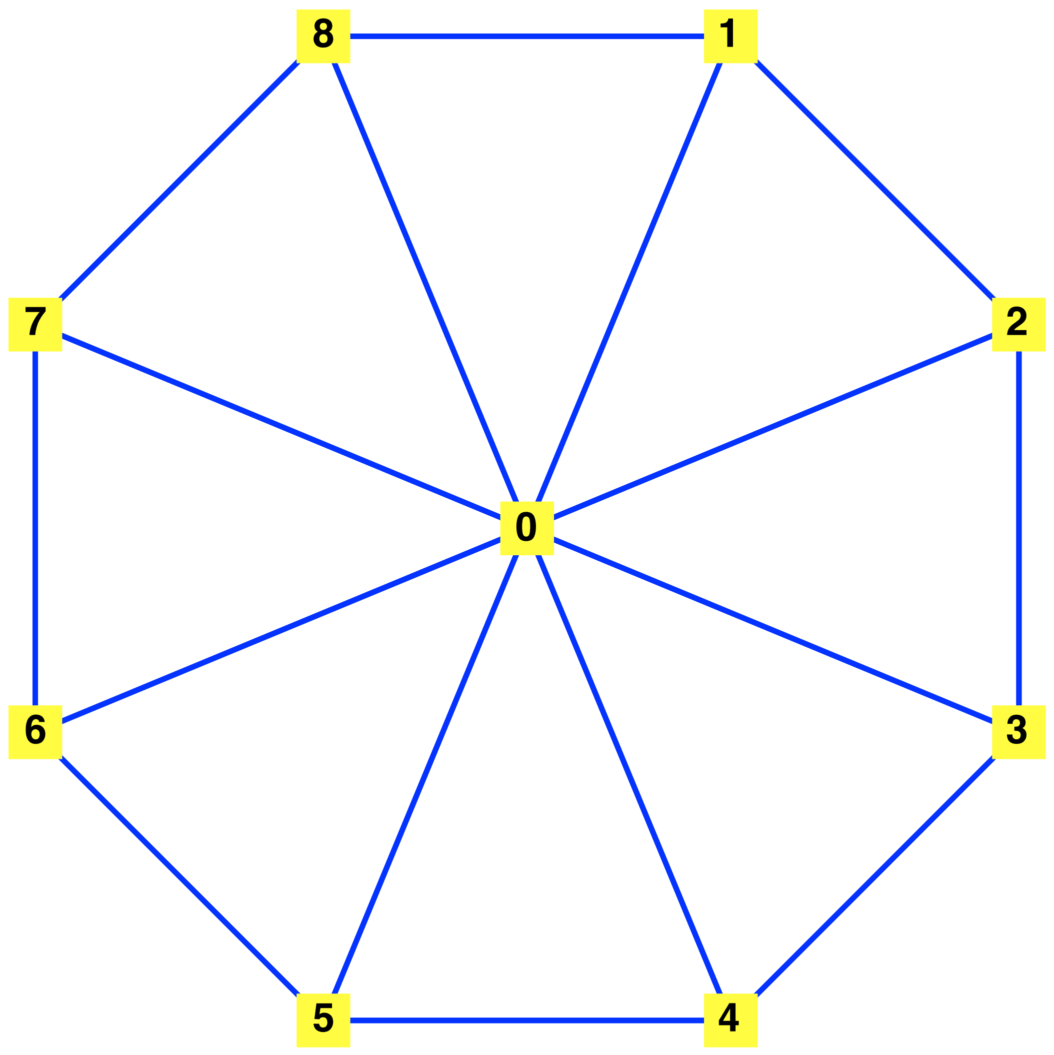
```
> B := BipartiteMatching(K44)
                B := 4, {{1, 5}, {2, 6}, {3, 7}, {4, 8}}                (19)
> HighlightEdges(K44, B[2])
> DrawGraph(K44)
```



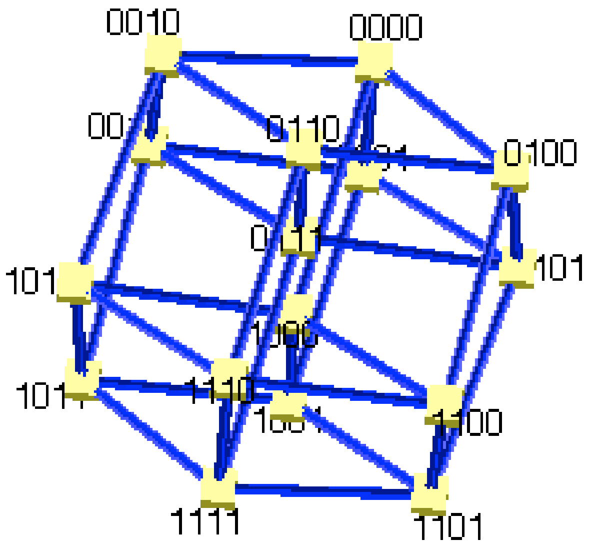
```
> C8 := CycleGraph(8)
            C8 := Graph 7: an undirected unweighted graph with 8 vertices and 8 edge(s)          (20)
> DrawGraph(C8)
```



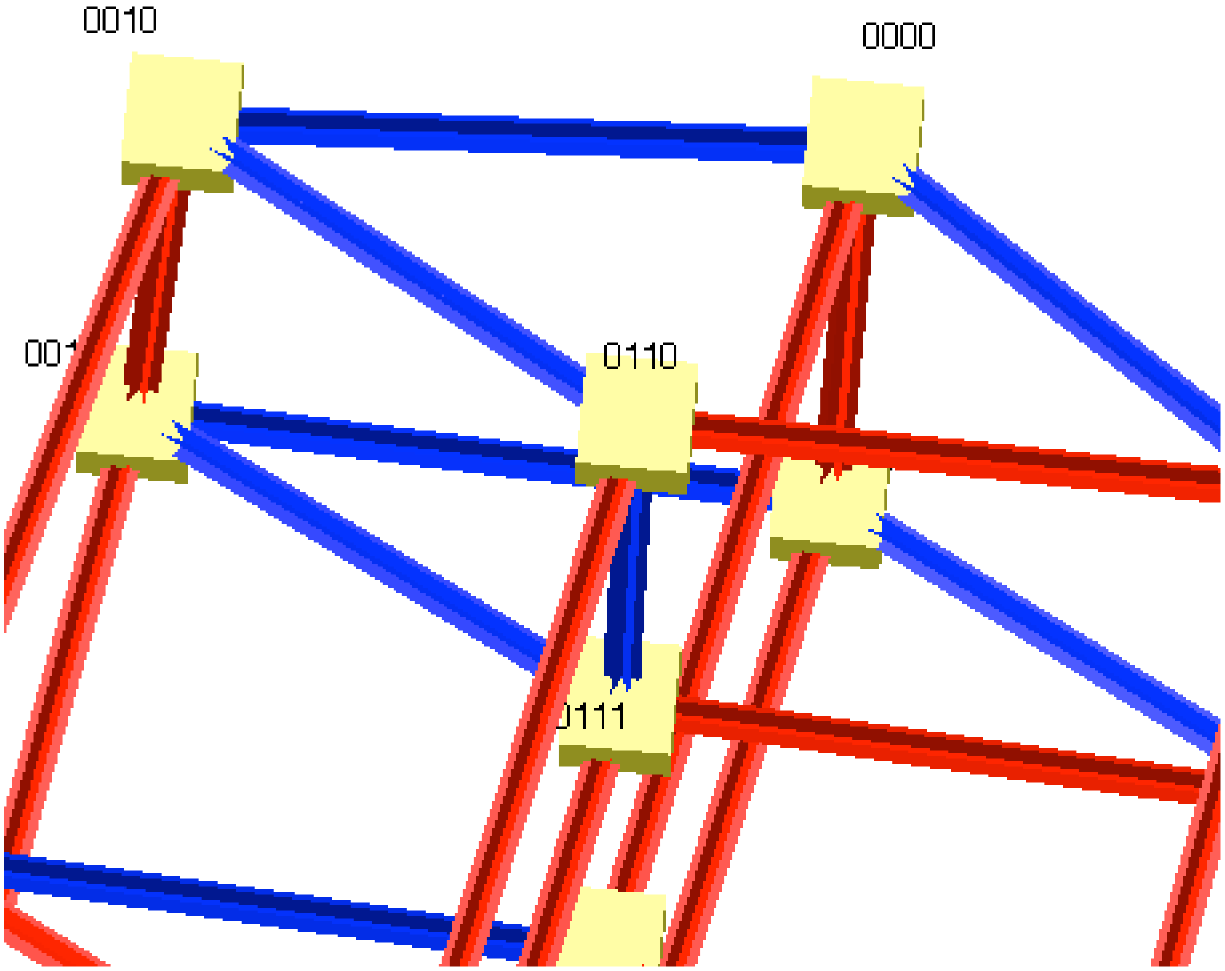
```
> W8 := WheelGraph(8)
        W8 := Graph 8: an undirected unweighted graph with 9 vertices and 16 edge(s)        (21)
> DrawGraph(W8)
```



```
> 
> Q4 := HypercubeGraph(4)
        Q4 := Graph 9: an undirected unweighted graph with 16 vertices and 32 edge(s)          (22)
> DrawGraph(Q4, style = spring, dimension = 3)
```



```
> IsHamiltonian(Q4,'Cir')
                                   true                                        (23)
> Cir
["0000", "1000", "1100", "0100", "0110", "1110", "1010", "0010", "0011", "1011", "1111",   (24)
    "0111", "0101", "1101", "1001", "0001", "0000"]
> HighlightTrail(Q4, Cir, red)
> DrawGraph(Q4, style = spring, dimension = 3)
```


0010
0000
001
0110
0111