GraphTheory

A substantial effort was put into Graph Theory for Maple 2024, including new commands for graph computation and generation.

```
> with(GraphTheory) :
```

New commands

These commands are new for Maple 2024:

AllGraphs	Condensation	FindAsteroidalTriple
IsArchimedeanGraph	IsAsteroidalTripleFree	IsDominatingSet
IsIndependentSet	MinCut	MoralGraph
RelationGraph	WienerIndex	

AllGraphs

The new AllGraphs command returns an iterator with which one can step through all graphs matching a particular set of criteria, such as vertex count or edge count. It is also possible to cause the Iterator to return only connected graphs or only graphs which are not isomorphic to a graph previously returned by this Iterator.

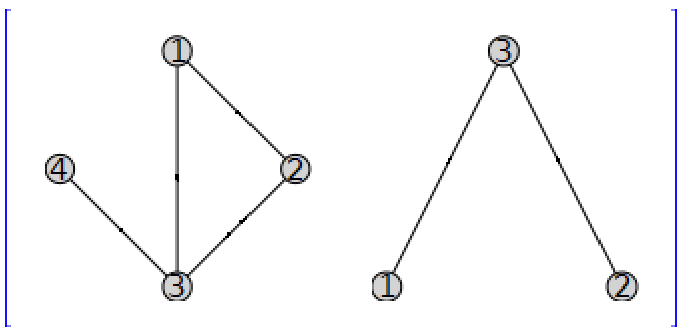
```
> iterator := AllGraphs(vertices = 3, nonisomorphic)

                         Graph Iterator
                         Vertices: 3..3
          iterator :=    Edges: 0..infinity
   connected=false, directed=false, selfloops=false, nonisomorphic=true

> iterator:-getNext( )
            Graph 1: an undirected graph with 3 vertices and 0 edge(s)
> iterator:-getNext( )
            Graph 2: an undirected graph with 3 vertices and 1 edge(s)
> iterator:-getNext( )
            Graph 3: an undirected graph with 3 vertices and 2 edge(s)
```

Condensation

The new command Condensation computes the *condensation* of a given (directed) graph.

The **condensation** of a graph *G* is a graph *C* whose vertices correspond to the strongly connected components of *G*. The condensation C will have an arc from *u* to *v* whenever there is an arc from the strongly connected component of *G* corresponding to *u* to the strongly connected component of *G* corresponding to *v*.

```
> T := Digraph( 4, {[1, 2], [1, 3], [2, 3], [3, 2], [4, 3]} )
            T := Graph 4: a directed graph with 4 vertices and 5 arc(s)
> C := Condensation(T)
            C := Graph 5: a directed graph with 3 vertices and 2 arc(s)
> DrawGraph~(⟨T|C⟩, stylesheet = [vertexpadding = 20])
```



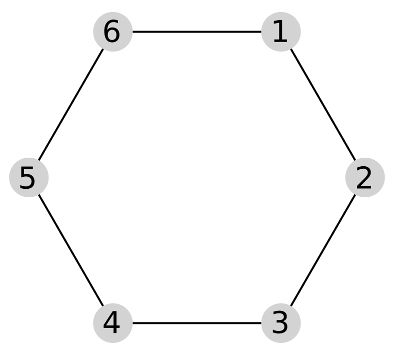
FindAsteroidalTriple and IsAsteroidalTripleFree

The new command FindAsteriodalTriple searches for an asteroidal triple in the given graph. The related new command IsAsteroidalTripleFree returns true when the given graph does not contain an asteroidal triple.

An **asteroidal triple** for a graph *G* is a triple (u, v, w) of non-adjacent vertices of *G* such that for each pair from the triple, there exists a path between them that does not intersect the third vertex or any of its neighbors.

```
> C6 := CycleGraph(6)
            C6 := Graph 6: an undirected graph with 6 vertices and 6 edge(s)
```

```
> DrawGraph(C6, size = [200, 200])
```



```
> FindAsteroidalTriple(C6)
```

$$[1, 3, 5]$$

IsArchimedeanGraph

The new command IsArchimedeanGraph tests whether a given graph is an *Archimedean graph*.

The **Archimedean graphs** are those graphs which form the skeletons of the Archimedean solids. The Archimedean solids comprise 13 convex polyhedra whose faces are regular polygons and whose vertices are all symmetric to each other, but which are not Platonic solids (polyhedra whose faces are identical).

```
> IsArchimedeanGraph( SpecialGraphs:-SnubCubeGraph( ) )
```

true

IsDominatingSet

The new command IsDominatingSet tests whether a set is a *dominating set* for a given graph.

A **dominating set** of a graph *G* is a subset *S* of the vertices of *G*, with the condition that every vertex in *G* must either be an element of *S* or the neighbor of an element of *S*.

```
> P5 := PathGraph(5)
```

P5 := Graph 7: an undirected graph with 5 vertices and 4 edge(s)

```
> S3 := {1, 3, 5}
```

$$S3 := \{1, 3, 5\}$$

```
> IsDominatingSet( P5, S3 );
```

true

MinCut

The new MinCut command works per the Max-Flow Min-Cut Theorem and uses the flow output to compute a cut-set. The EdgeConnectivity and VertexConnectivity commands have been updated to use MinCut so that they can now also return cut-sets.

```
> MG := Graph({{1, 2}, {1, 3}, {2, 3}, {2, 4}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {4, 7}})
```

MG := Graph 8: an undirected graph with 7 vertices and 9 edge(s)

```
> (mf, flowM) := MaxFlow(MG, 2, 7); i
```

$$mf, flowM := 2, \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$i$$

The MinCut command will call MaxFlow by default but can reuse the flow matrix if MaxFlow has been called already.

```
> MinCut(MG, 2, 7, flows = flowM);
```

$$\{\{3, 7\}, \{4, 7\}\}$$

```
> EdgeConnectivity(MG, output = cutset);
```

$$\{\{3, 5\}\}$$

```
> VertexConnectivity(MG, output = cutset);
```

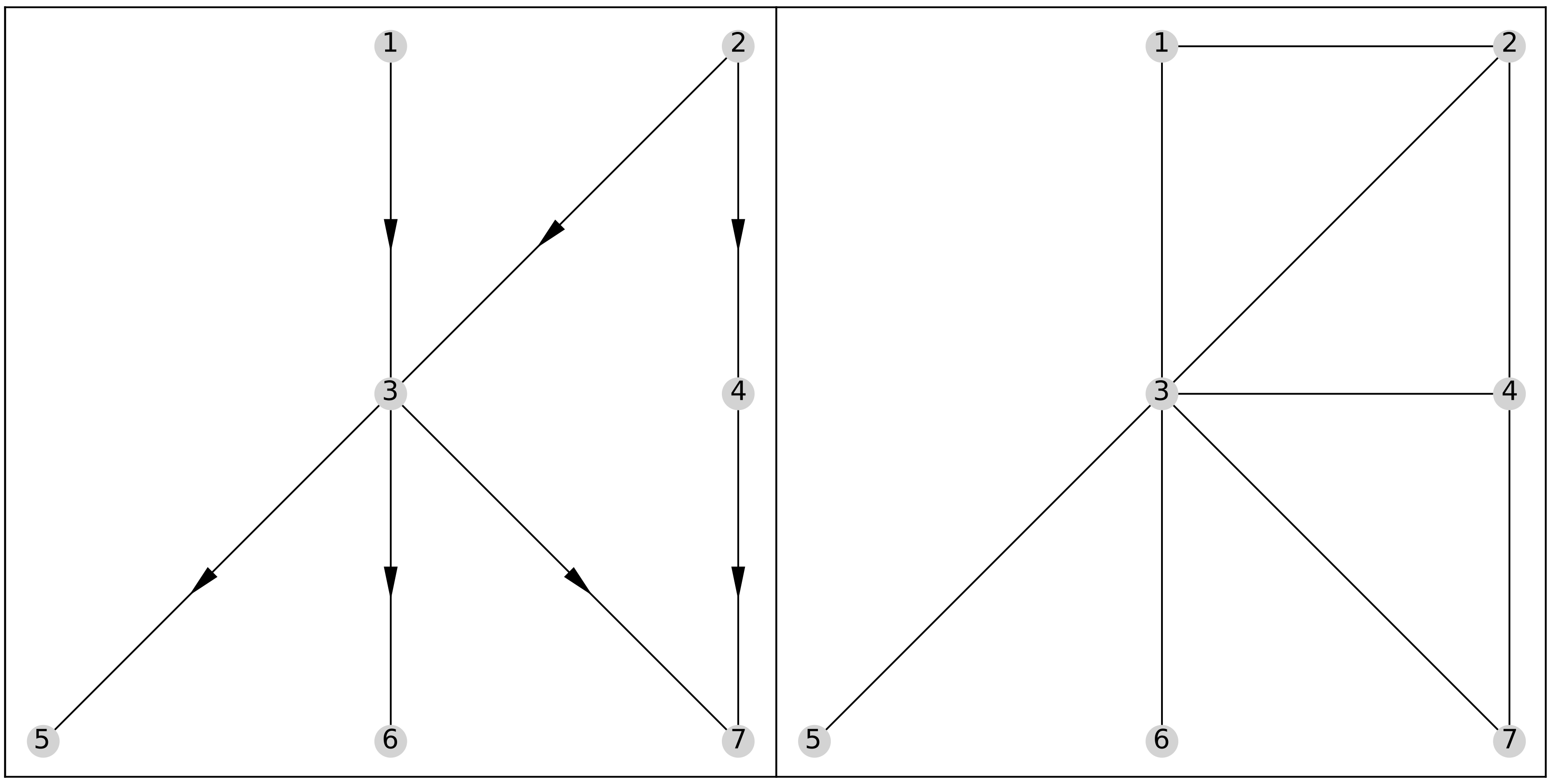
$$\{3\}$$

MoralGraph

The **moral graph** of a directed graph *G* is a graph *M* with the same vertices as *G* but with all directed edges made undirected, and with the property that whenever two directed edges in G share a destination vertex, then their source vertices are connected in *M*.

The new command MoralGraph constructs the moral graph given a directed graph.

```
> DG := Graph( 7, {[1, 3], [2, 3], [2, 4], [3, 5], [3, 6], [3, 7], [4, 7]} , vertexpositions = [[1, 2], [2, 2], [1, 1], [2, 1], [0, 0], [1, 0], [2, 0]])
                DG := Graph 9: a directed graph with 7 vertices and 7 arc(s)
> NumberOfEdges(DG)
                                            7
> MG := MoralGraph( DG )
                MG := Graph 10: an undirected graph with 7 vertices and 9 edge(s)
> NumberOfEdges(MG)
                                            9
> SetVertexPositions(MG, GetVertexPositions(DG))
> plots:-display(DrawGraph~(⟨DG|MG⟩))
```

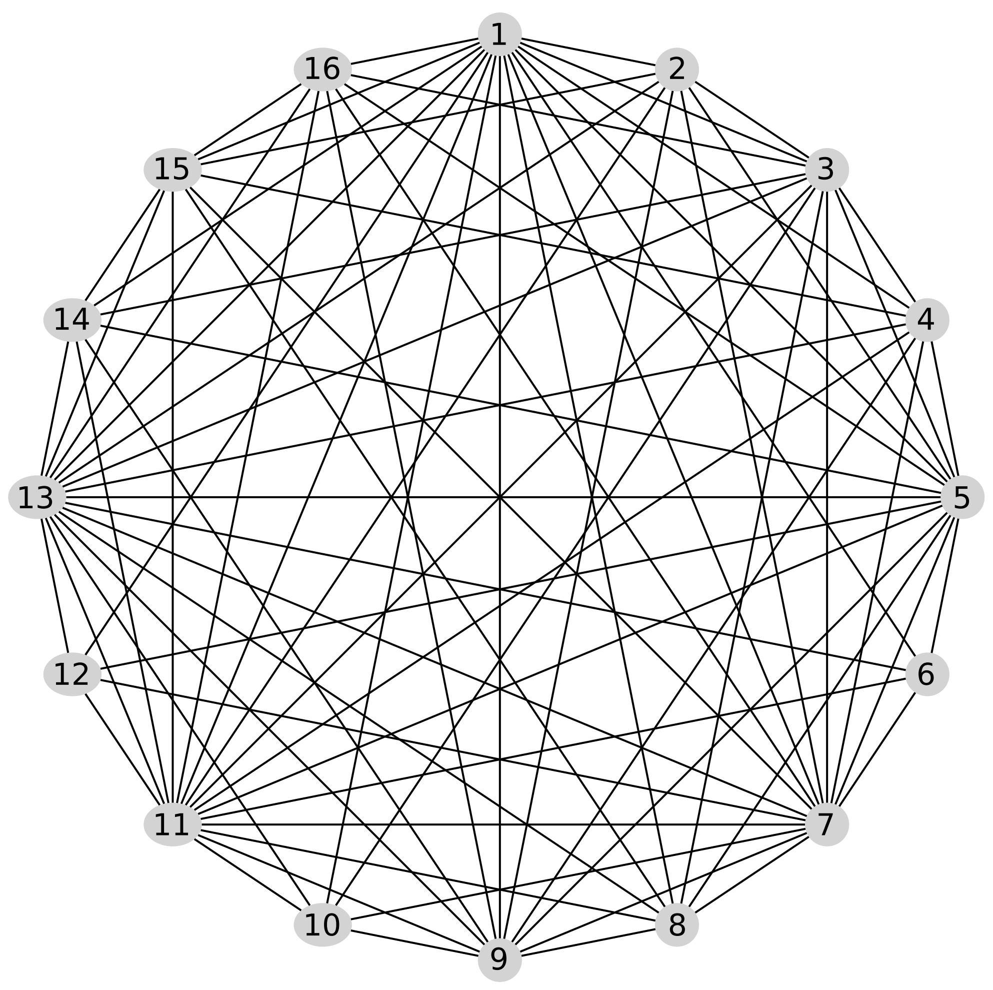


RelationGraph

The RelationGraph command constructs a graph on a given vertex list in which an edge is present in the graph whenever a particular Boolean predicate on its two vertices is true.

```
> RG := RelationGraph( [seq(1 ..16)], NumberTheory:-AreCoprime )
                RG := Graph 11: an undirected graph with 16 vertices and 79 edge(s)
```

```
> DrawGraph(RG)
```



In this example, the neighborhood of the element 5 will be all those vertices whose values are coprime with 5:

```
> Neighborhood( RG, 5 )
                    [1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 16]
```

WienerIndex

The WienerIndex command computes the Wiener index on a given graph. The **Wiener index** of a graph is the sum of the lengths of the shortest paths between all pairs of vertices in the graph.

```
> G := CycleGraph(20)
          G := Graph 12: an undirected graph with 20 vertices and 20 edge(s)
> WienerIndex(G)
                                     40
```

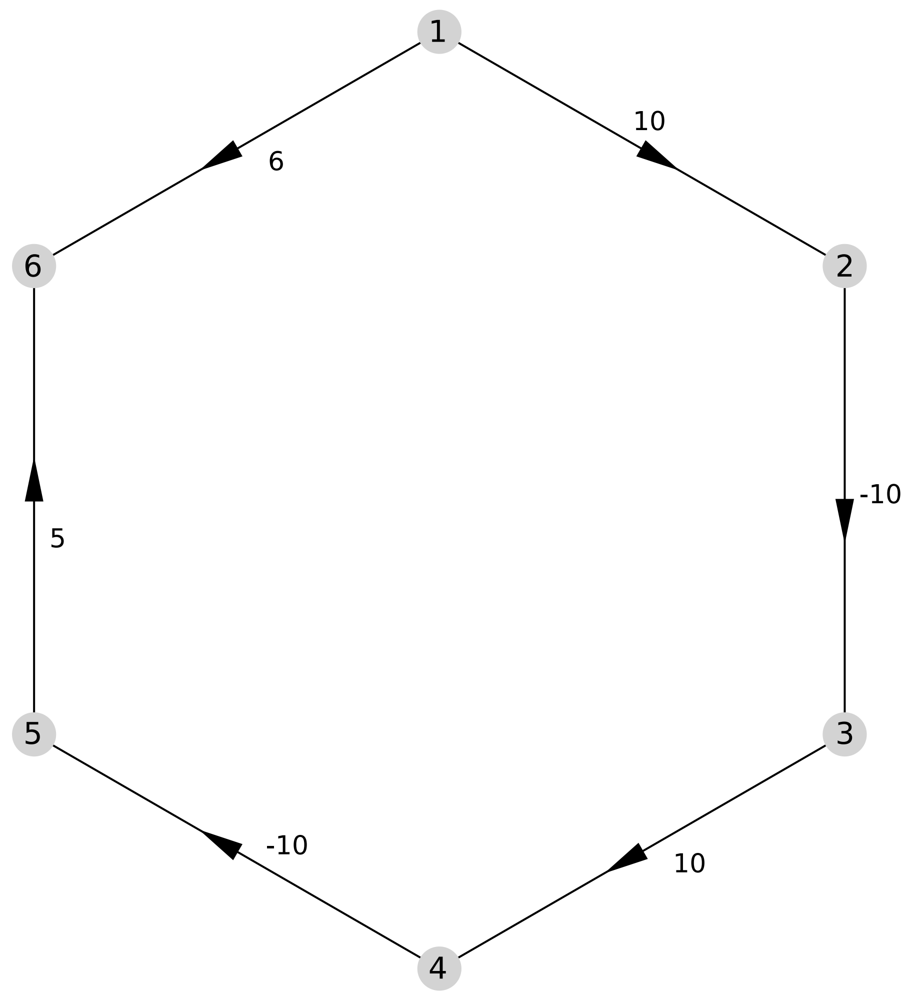
New functionality for existing commands

Distance and ShortestPath

The Distance and ShortestPath commands now use the edge weights of a weighted matrix. They both have a new option **ignoreweights** to compute the distance or shortest path in the underlying unweighted graph.

```
> W6 := Graph( {[[1,2], 10], [[1,6], 6], [[2,3], −10], [[3,4], 10], [[4,5], −10], [[5,6], 5]} );
          W6 := Graph 13: a directed weighted graph with 6 vertices and 6 arc(s)
```

```
> DrawGraph(W6, layout = circle);
```



```
> ShortestPath(W6, 1, 6);
                              [1, 2, 3, 4, 5, 6]
> Distance(W6, 1, 6);
                                      5
> ShortestPath(W6, 1, 6, 'ignoreweights');
                                    [1, 6]
> Distance(W6, 1, 6, 'ignoreweights');
                                      1
```

▼ MaxFlow

The MaxFlow command now works on all graphs. It now treats an unweighted graph as a graph with all edge weights equal to 1, or to the edge multiplicity for multigraphs.

▼ Additions to SpecialGraphs

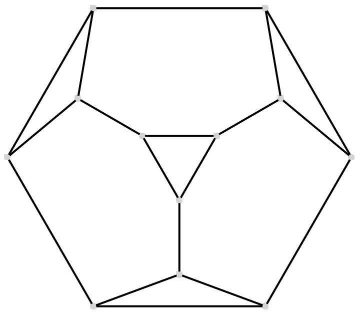
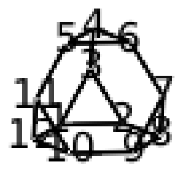
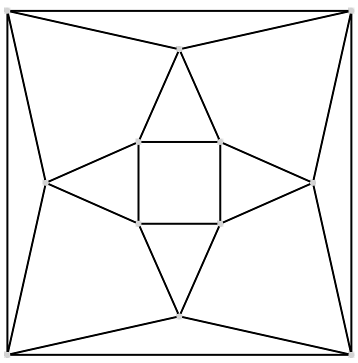
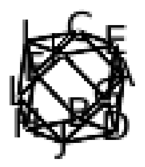
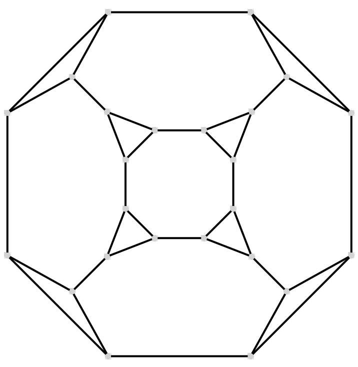
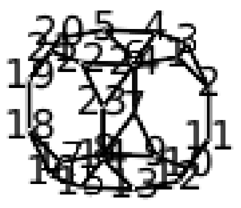
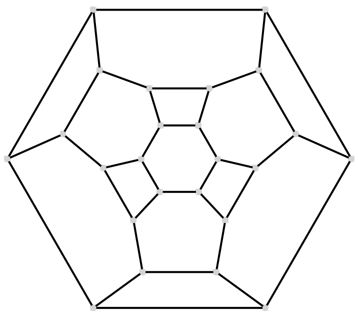
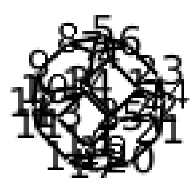
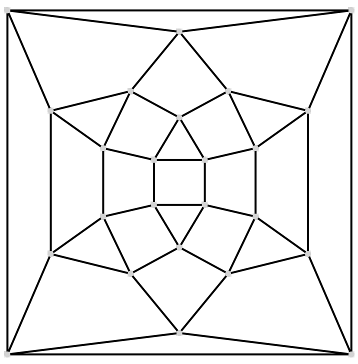
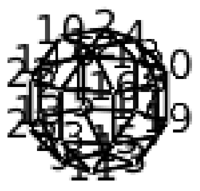
The SpecialGraphs subpackage now includes commands for all the Archimedean graphs, as well as the Möbius ladder graph and the Wagner graph.

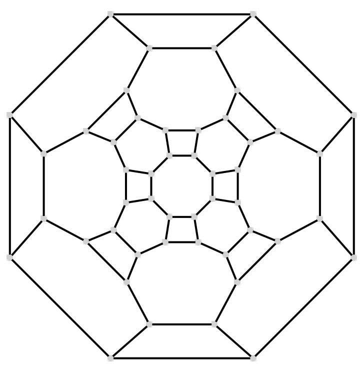
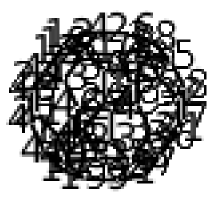
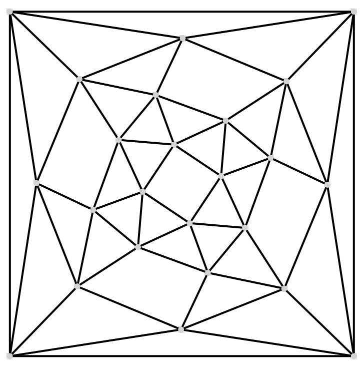
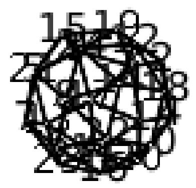
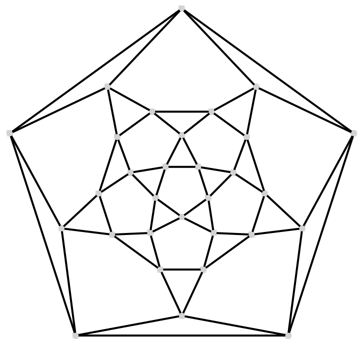
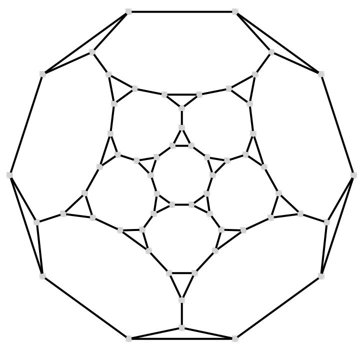
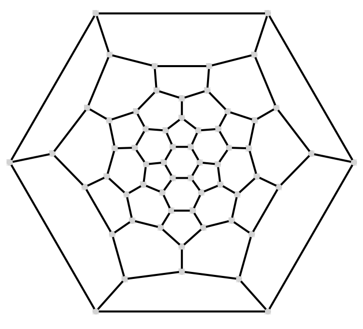
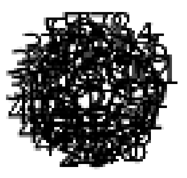
▼ Archimedean Graphs

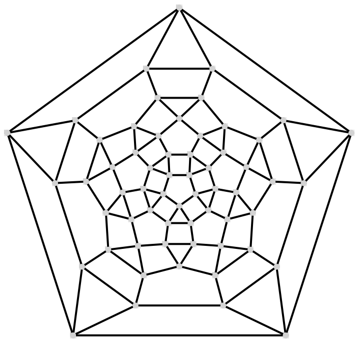
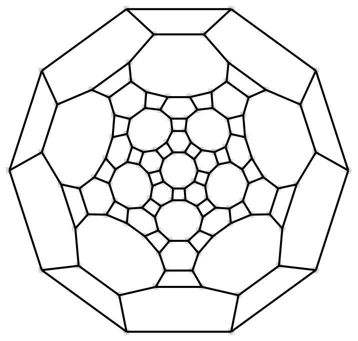
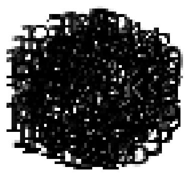
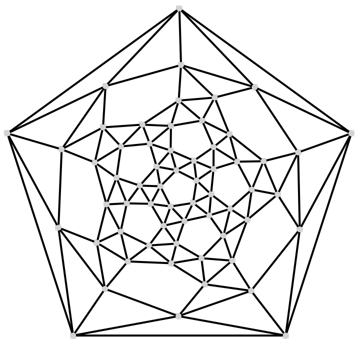
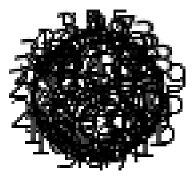
The **Archimedean graphs** are those graphs which form the skeletons of the Archimedean solids. The Archimedean solids comprise 13 convex polyhedra whose faces are regular polygons and whose vertices are all symmetric to each other, but which are not Platonic solids (polyhedra whose faces are identical). They were first enumerated by Archimedes.

The new command IsArchimedeanGraph tests whether a given graph is an Archimedean graph.

The following new commands generate each of the Archimedean graphs.

Archimedean solid	Degree	Vertices	Edges	Command	2-D	3-D
truncated tetrahedon	3	12	18	> *G* ≔ *SpecialGraphs*:-*TruncatedTetrahedronGraph*( ); *G* ≔ *Graph 14: an undirected graph with 12 vertices and 18 edge(s)*	> *DrawPlanar*(*G*, *showlabels* = *false*) 	> *DrawGraph*(*G*, *dimension* = 3) 
cuboctahedron	4	12	24	> *letters* ≔ [*seq*("ABCDEFGHIJKL")] *letters* ≔ ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L"] > *G* ≔ *SpecialGraphs*:-*CuboctahedronGraph*(*letters*) *G* ≔ *Graph 15: an undirected graph with 12 vertices and 24 edge(s)*	> *DrawPlanar*(*G*, *showlabels* = *false*) 	> *DrawGraph*(*G*, *dimension* = 3) 
truncated cube	3	24	36	> *G* ≔ *SpecialGraphs*:-*TruncatedCubeGraph*( ) *G* ≔ *Graph 16: an undirected graph with 24 vertices and 36 edge(s)*	> *DrawPlanar*(*G*, *showlabels* = *false*) 	> *DrawGraph*(*G*, *dimension* = 3) 
truncated octahedron	3	24	36	> *G* ≔ *SpecialGraphs*:-*TruncatedOctahedronGraph*( ) *G* ≔ *Graph 17: an undirected graph with 24 vertices and 36 edge(s)*	> *DrawPlanar*(*G*, *showlabels* = *false*) 	> *DrawGraph*(*G*, *dimension* = 3) 
small rhombicuboctahedron	4	24	48	> *G* ≔ *SpecialGraphs*:-*SmallRhombicuboctahedron*\ *Graph*( ) *G* ≔ *Graph 18: an undirected graph with 24 vertices and 48 edge(s)*	> *DrawPlanar*(*G*, *showlabels* = *false*) 	> *DrawGraph*(*G*, *dimension* = 3) 

great rhombicuboctahedron (also called truncated cuboctahedron)	3	48	72	> $G := SpecialGraphs\text{:-}GreatRhombicuboctahedronGraph(\ )$ $G :=$ *Graph 19: an undirected graph with 48 vertices and 72 edge(s)*	> $DrawPlanar(G, showlabels = false)$ 	> $DrawGraph(G, dimension = 3)$ 
snub cube	5	24	60	> $G := SpecialGraphs\text{:-}SnubCubeGraph(\ )$ $G :=$ *Graph 20: an undirected graph with 24 vertices and 60 edge(s)*	> $DrawPlanar(G, showlabels = false)$ 	> $DrawGraph(G, dimension = 3)$ 
Icosidodecahedron	4	30	60	> $G := SpecialGraphs\text{:-}IcosidodecahedronGraph(\ )$ $G :=$ *Graph 21: an undirected graph with 30 vertices and 60 edge(s)*	> $DrawPlanar(G, showlabels = false)$ 	> $DrawGraph(G, dimension = 3)$
truncated dodecahedron	3	60	90	> $G := SpecialGraphs\text{:-}TruncatedDodecahedronGraph(\ )$ $G :=$ *Graph 22: an undirected graph with 60 vertices and 90 edge(s)*	> $DrawPlanar(G, showlabels = false)$ 	> $DrawGraph(G, dimension = 3)$
truncated icosahedron	3	60	90	> $G := SpecialGraphs\text{:-}TruncatedIcosahedronGraph(\ )$ $G :=$ *Graph 23: an undirected graph with 60 vertices and 90 edge(s)*	> $DrawPlanar(G, showlabels = false)$ 	> $DrawGraph(G, dimension = 3)$ 

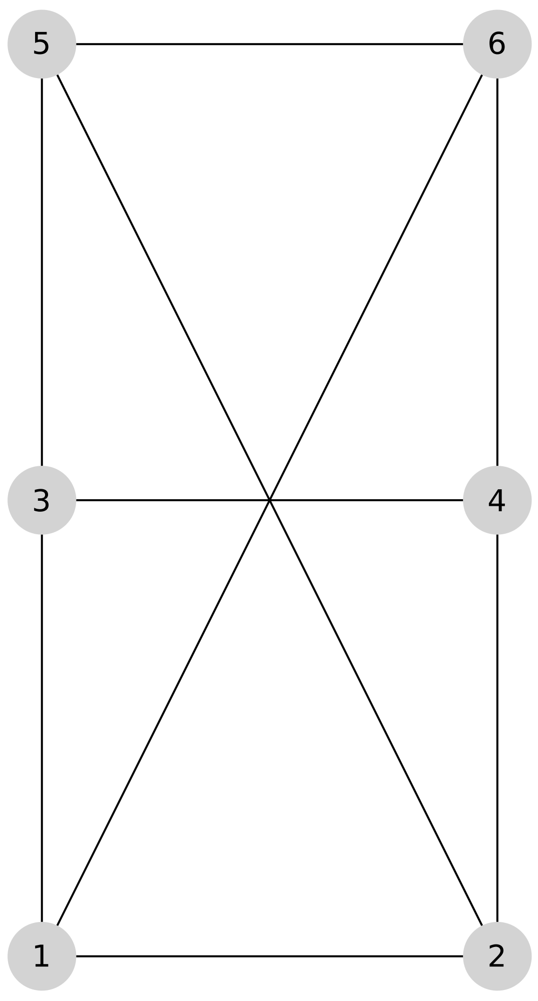
small rhombicosidodecahedron	4	60	120	> G := *SpecialGraphs*:-*SmallRhombicosidodecahedronGraph*( ) *G* := *Graph 24: an undirected graph with 60 vertices and 120 edge(s)*	> *DrawPlanar*(*G*, *showlabels* = *false*) 	> *DrawGraph*(*G*, *dimension* = 3)
great rhombicosidodecahedron (also called truncated icosidodecahedron)	3	120	180	> G := *SpecialGraphs*:-*GreatRhombicosidodecahedronGraph*( ) *G* := *Graph 25: an undirected graph with 120 vertices and 180 edge(s)*	> *DrawPlanar*(*G*, *showlabels* = *false*) 	> *DrawGraph*(*G*, *dimension* = 3) 
snub dodecahedron	5	60	150	> G := *SpecialGraphs*:-*SnubDodecahedronGraph*( ) *G* := *Graph 26: an undirected graph with 60 vertices and 150 edge(s)*	> *DrawPlanar*(*G*, *showlabels* = *false*) 	> *DrawGraph*(*G*, *dimension* = 3) 

Möbius ladder graph and Wagner graph

The **Möbius ladder graph** on $2n$ vertices may be understood visually as the ladder graph on the same vertices, with the addition of two edges: one connecting the left-hand side of the "op rung to the right-hand side of the bottom rung, and one connecting the right-hand side of the top rung to the left-hand side of the bottom rung.

```
> WG : ML3 := SpecialGraphs:-MoebiusLadderGraph(3)
                 ML3 := Graph 27: an undirected graph with 6 vertices and 9 edge(s)
```

```
> DrawGraph(ML3, stylesheet = [vertexpadding = 8])
```



The Wagner graph is a 3-regular graph which is a particular instance of an Möbius ladder graph on 8 vertices.

```
> WG := SpecialGraphs:-WagnerGraph( )
                 WG := Graph 28: an undirected graph with 8 vertices and 12 edge(s)
> DrawGraph(WG, stylesheet = [vertexpadding = 8])
```