Instructor:

BRIJ MOHAN
Hansraj College,
University of Delhi, Delhi.

GraphTheory Updates in Maple 2025

A new package to work in Graph Theory

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

New commands

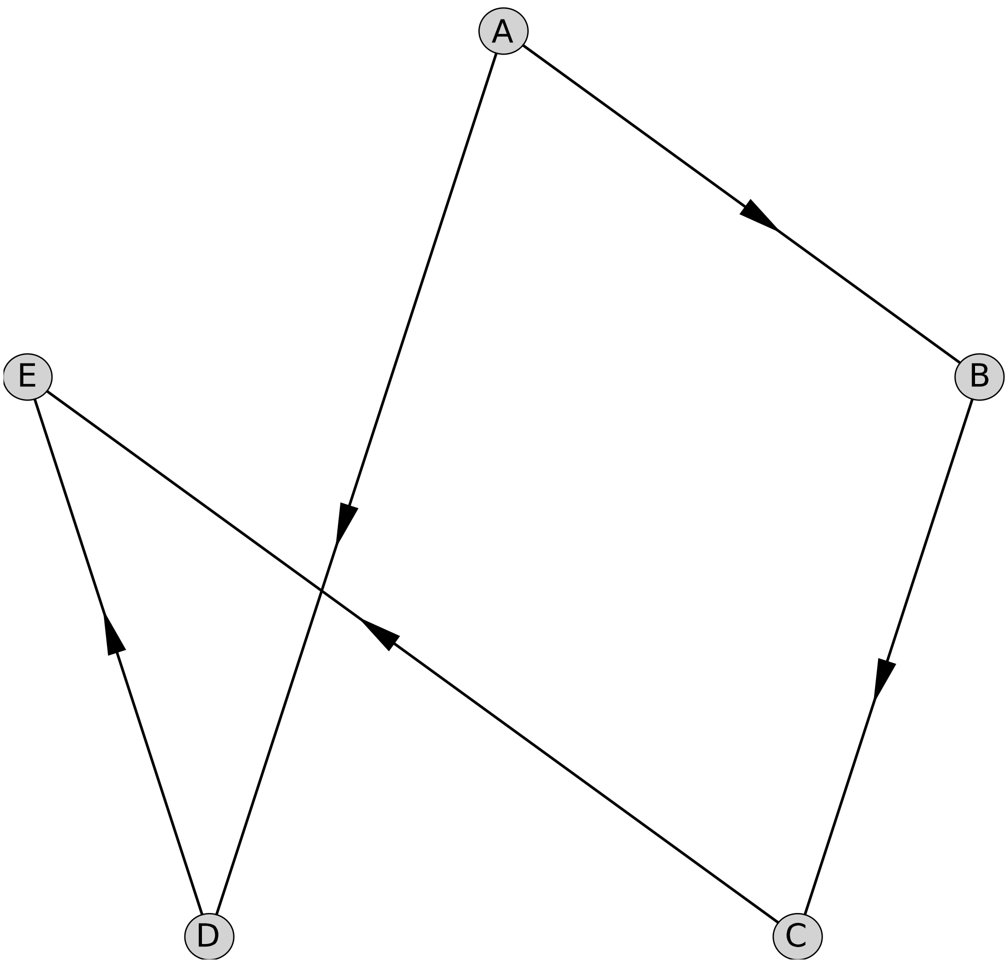
AllSimplePaths

The new AllSimplePaths command returns an iterator with which one can step through all paths from a given vertex to another vertex in a directed graph.

```
> G1 := Graph({["A", "B"], ["A", "D"], ["B", "C"], ["C",
  "E"], ["D", "E"]});
```

G1 := *Graph 1: a directed graph with 5 vertices and 5 arcs* **(2.1.1)**

```
> DrawGraph(G1);
```




```
> iterator := AllSimplePaths( G1, "A", "E" );
```

(2.1.2)

$$iterator := \left[\quad \textit{Path Iterator} \quad\right] \tag{2.1.2}$$

```
> iterator:-getNext();
```

$$["A", "D", "E"] \tag{2.1.3}$$

```
> iterator:-getNext();
```

$$["A", "B", "C", "E"] \tag{2.1.4}$$

```
> iterator:-hasNext();
```

$$\mathit{false} \tag{2.1.5}$$

Ancestors and Descendants

The new Ancestors command returns a list of ancestors of the given vertex in the given directed graph. The related new command Descendants returns a list of descendants of the given vertex.

```
> Ancestors( G1, "A" );
```

$$[\,] \tag{2.2.1}$$

```
> Ancestors( G1, "E" );
```

$$["A", "B", "C", "D"] \tag{2.2.2}$$

```
> Descendants( G1, "A" );
```

$$["B", "C", "D", "E"] \tag{2.2.3}$$

FindCycle

The new FindCycle command finds a cycle, if one exists in the given graph.

```
> FindCycle(G1);
```

$$[\,] \tag{2.3.1}$$

```
> FindCycle( Graph( {["A", "B"], ["B", "C"], ["C", "A"]} ) )
  ;
```

$$["C", "A", "B", "C"] \tag{2.3.2}$$

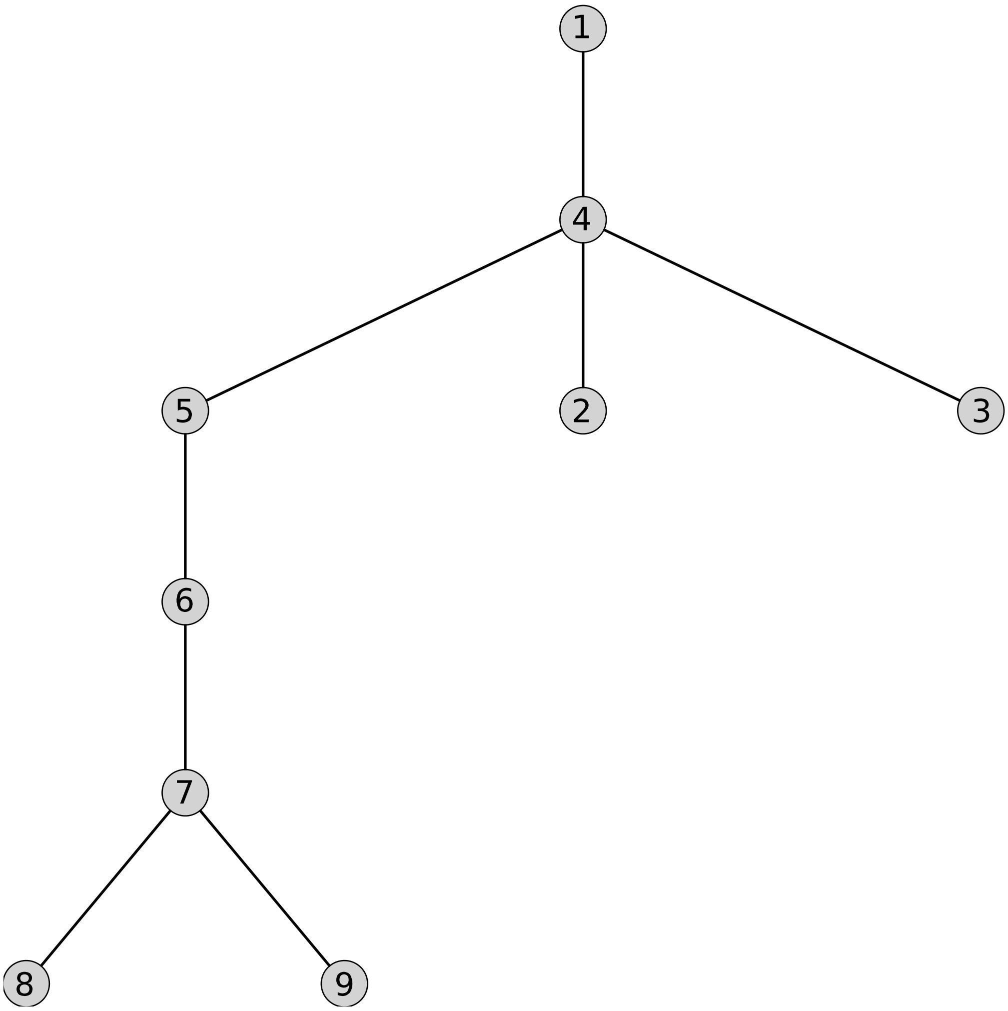
IsCaterpillarTree and IsLobsterTree

The new IsCaterpillarTree command tests whether the graph is a caterpillar tree, a tree for which there is some path such that every vertex is either on the path or the neighbor of a vertex on the path.

```
> CT := Graph({{1,4},{2,4},{3,4},{4,5},{5,6},{6,7},{7,8},{7,
  9}});
```

$$CT := \textit{Graph 2: an undirected graph with 9 vertices and 8 edges} \tag{2.4.1}$$

```
> DrawGraph(CT);
```




```
> IsCaterpillarTree(CT);
```

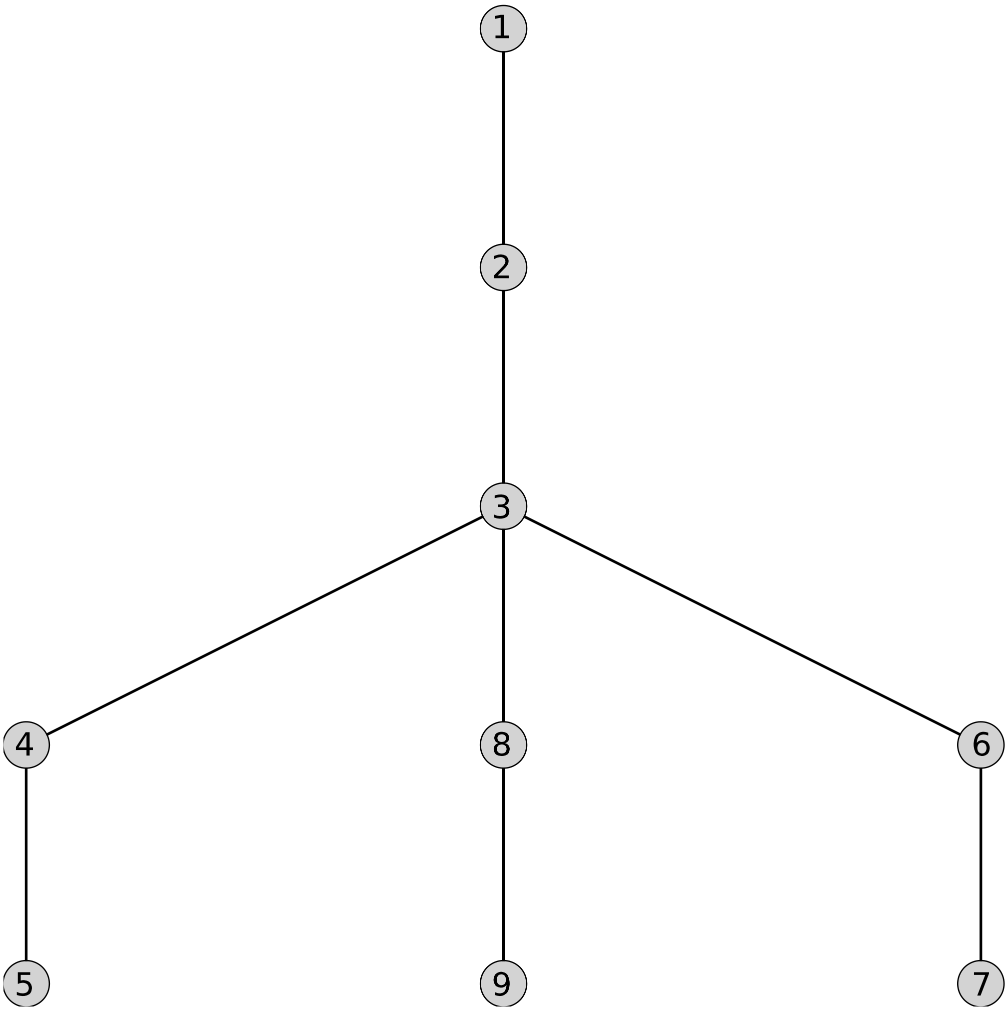
true **(2.4.2)**

The new IsLobsterTree command tests whether the graph is lobster tree, a tree such that the result of removing all leaf vertices is a caterpillar tree.

```
> LT := Graph({{1,2},{2,3},{3,4},{4,5},{3,6},{6,7},{3,8},{8,
  9}});
```

LT := Graph 3: an undirected graph with 9 vertices and 8 edges **(2.4.3)**

```
> DrawGraph(LT);
```




```
> IsLobsterTree(LT);
```

true **(2.4.4)**

```
> IsCaterpillarTree(LT);
```

false **(2.4.5)**

IsPlatonicGraph

The new IsPlatonicGraph command tests whether the graph is Platonic. The Platonic graphs consist of those graphs whose skeletons are the Platonic solids (polyhedra whose faces are identical).

```
> IsPlatonicGraph( SpecialGraphs:-CubeGraph() );
```

true **(2.5.1)**

LongestPath

The new LongestPath command computes the longest path within a given (directed) graph.

```
> LongestPath(G1);
```

["A", "B", "C", "E"] **(2.6.1)**

LowestCommonAncestors

The new LowestCommonAncestors command computes the set of lowest common ancestors in a given directed graph.

```
> LowestCommonAncestors( G1, "C", "D" );
```

{"A"} **(2.7.1)**

ModularityMatrix

The new ModularityMatrix command computes the modularity matrix of the graph G.

```
> ModularityMatrix(G1);
```

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ -\frac{2}{5} & -\frac{1}{5} & \frac{4}{5} & -\frac{1}{5} & 0 \\ -\frac{2}{5} & -\frac{1}{5} & -\frac{1}{5} & -\frac{1}{5} & 1 \\ -\frac{2}{5} & -\frac{1}{5} & -\frac{1}{5} & -\frac{1}{5} & 1 \\ -\frac{4}{5} & -\frac{2}{5} & -\frac{2}{5} & -\frac{2}{5} & 0 \end{bmatrix} \tag{2.8.1}$$

ResistanceDistance

The new ResistanceDistance command computes the resistance distance of the graph G.

```
> ResistanceDistance(SpecialGraphs:-CubeGraph());
```

(2.9.1)

$$\begin{bmatrix} 0 & \frac{7}{12} & \frac{7}{12} & \frac{3}{4} & \frac{7}{12} & \frac{3}{4} & \frac{3}{4} \cdots \\ \frac{7}{12} & 0 & \frac{3}{4} & \frac{7}{12} & \frac{3}{4} & \frac{7}{12} & \frac{5}{6} \cdots \\ \frac{7}{12} & \frac{3}{4} & 0 & \frac{7}{12} & \frac{3}{4} & \frac{5}{6} & \frac{7}{12} \cdots \\ \frac{3}{4} & \frac{7}{12} & \frac{7}{12} & 0 & \frac{5}{6} & \frac{3}{4} & \frac{3}{4} \cdots \\ \frac{7}{12} & \frac{3}{4} & \frac{3}{4} & \frac{5}{6} & 0 & \frac{7}{12} & \frac{7}{12} \cdots \\ \frac{3}{4} & \frac{7}{12} & \frac{5}{6} & \frac{3}{4} & \frac{7}{12} & 0 & \frac{3}{4} \cdots \\ \frac{3}{4} & \frac{5}{6} & \frac{7}{12} & \frac{3}{4} & \frac{7}{12} & \frac{3}{4} & 0 \cdots \\ \frac{5}{6} & \frac{3}{4} & \frac{3}{4} & \frac{7}{12} & \frac{3}{4} & \frac{7}{12} & \frac{7}{12} \cdots \end{bmatrix} \quad \textbf{(2.9.1)}$$

ShortestAncestralPath and ShortestDescendantPath

The new ShortestAncestralPath constructs the shortest ancestral path between two nodes in the given directed graph.

```
> ShortestAncestralPath( G1, "C", "D") ;
```

$$[["A", "B", "C"], ["A", "D"]] \quad \textbf{(2.10.1)}$$

You can similarly find the shortest descendent path.

New functionality for existing commands

IsReachable and Reachable

The IsReachable and Reachable commands now have a new option distance to constrain the distance within a given vertex.

```
> IsReachable( G1, "A", "E", distance = 1 );
```

false **(3.1.1)**

```
> Reachable( G1, "A", distance = 1 );
```

["A", "B", "D"] **(3.1.2)**

ShortestPath

The ShortestPath command accepts an option avoidvertices to constrain the search space for a shortest path to avoid some specified set of vertices.

```
> ShortestPath( G1, "A", "E" );
```

["A", "D", "E"] **(3.2.1)**

```
> ShortestPath( G1, "A", "E", avoidvertices = {"D"} );
```

["A", "B", "C", "E"] **(3.2.2)**

Additions to SpecialGraphs

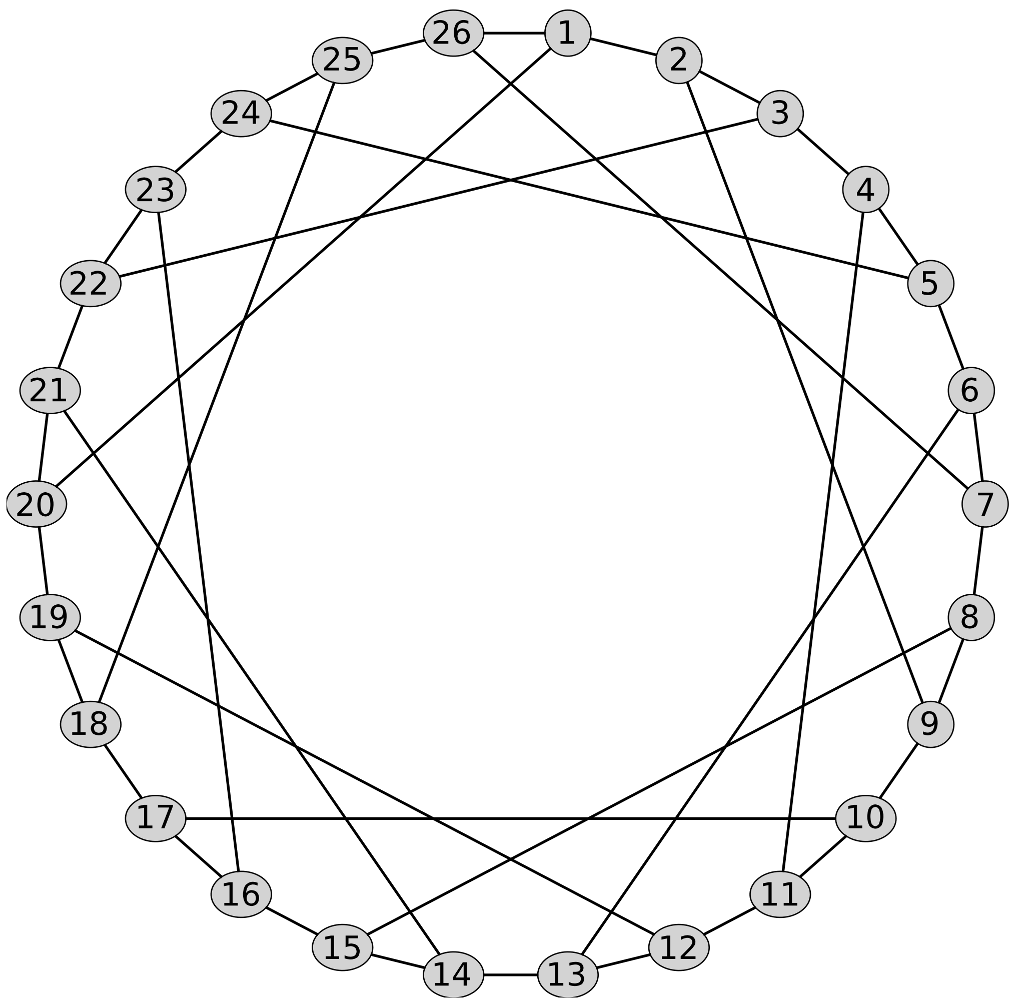
The SpecialGraphs subpackage now includes commands for the F26a graph and Hanoi graph.

- The F26a graph may be understood visually

```
> FG := SpecialGraphs:-F26AGraph();
```

FG := *Graph 4: an undirected graph with 26 vertices and 39 edges* **(4.1)**

```
> DrawGraph(FG);
```



- The Hanoi graph is a graph whose edges correspond to allowed moves of the tower of Hanoi problem.

```
> HG4 := SpecialGraphs:-HanoiGraph(4);
```

HG4 ≔ *Graph 5: an undirected graph with 81 vertices and 120 edges* **(4.2)**