Appendix

Reference Guide

■ AcyclicQ

`AcyclicQ[g]` yields `True` if graph g is acyclic.

See also `FindCycle`, `TreeQ` ■ See page 285

■ AddEdge

`AddEdge[g, e]` returns a graph g with the new edge e added. e can have the form $\{a, b\}$ or the form $\{\{a, b\}, options\}$.

See also `AddVertex`, `DeleteEdge` ■ See page 193

■ AddEdges

`AddEdges[g, l]` gives graph g with the new edges in l added. l can have the form $\{a, b\}$, to add a single edge $\{a, b\}$, or the form $\{\{a, b\}, \{c, d\}, \ldots\}$, to add edges $\{a, b\}, \{c, d\}, \ldots$, or the form $\{\{\{a, b\}, x\}, \{\{c, d\}, y\}, \ldots\}$, where x and y can specify graphics information associated with $\{a, b\}$ and $\{c, d\}$, respectively.

New function ■ See also `AddEdge` ■ See page 193

■ AddVertex

`AddVertex[g]` adds one disconnected vertex to graph g. `AddVertex[g, v]` adds to g a vertex with coordinates specified by v.

See also `AddEdge`, `DeleteVertex` ■ See page 195

■ AddVertices

`AddVertices[g, n]` adds n disconnected vertices to graph g. `AddVertices[g, vList]` adds vertices in *vList* to g. *vList* contains embedding and graphics information and can have the form $\{x, y\}$ or $\{\{x_1, y_1\}, \{x_2, y_2\} \ldots\}$ or the form $\{\{\{x_1, y_1\}, g_1\}, \{\{x_2, y_2\}, g_2\}, \ldots\}$, where $\{x, y\}$, $\{x_1, y_1\}$, and $\{x_2, y_2\}$ are point coordinates and g_1 and g_2 are graphics information associated with vertices.

New function ■ See also `AddVertex` ■ See page 195

■ Algorithm

`Algorithm` is an option that informs functions such as `ShortestPath`, `VertexColoring`, and `VertexCover` about which algorithm to use.

New function ■ See also `ShortestPath`, `VertexColoring`, `VertexCover` ■ See page 313

■ AllPairsShortestPath

`AllPairsShortestPath[g]` gives a matrix, where the (i, j)th entry is the length of a shortest path in g between vertices i and j. `AllPairsShortestPath[g, Parent]` returns a three-dimensional matrix with dimensions $2 \times V[g] \times V[g]$, in which the $(1, i, j)$th entry is the length of a shortest path from i to j and the $(2, i, j)$th entry is the predecessor of j in a shortest path from i to j.

See also `ShortestPath`, `ShortestPathSpanningTree` ■ See page 330

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ AlternatingGroup

`AlternatingGroup[n]` generates the set of even size-n permutations, the alternating group on n symbols. `AlternatingGroup[l]` generates the set of even permutations of the list l.

New function ▪ See also `SymmetricGroup`, `OrbitInventory` ▪ See page 110

■ AlternatingGroupIndex

`AlternatingGroupIndex[n, x]` gives the cycle index of the alternating group of size-n permutations as a polynomial in the symbols $x[1], x[2], \ldots, x[n]$.

New function ▪ See also `SymmetricGroupIndex`, `OrbitInventory` ▪ See page 125

■ AlternatingPaths

`AlternatingPaths[g, start, ME]` returns the alternating paths in graph g with respect to the matching *ME*, starting at the vertices in the list *start*. The paths are returned in the form of a forest containing trees rooted at vertices in *start*.

New function ▪ See also `BipartiteMatching`, `BipartiteMatchingAndCover` ▪ See page 345

■ AnimateGraph

`AnimateGraph[g, l]` displays graph g with each element in the list l successively highlighted. Here l is a list containing vertices and edges of g. An optional flag, which takes on the values `All` and `One`, can be used to inform the function about whether objects highlighted earlier will continue to be highlighted or not. The default value of flag is `All`. All the options allowed by the function `Highlight` are permitted by `AnimateGraph` as well. See the usage message of `Highlight` for more details.

New function ▪ See also `Highlight`, `ShowGraph` ▪ See page 212

■ AntiSymmetricQ

`AntiSymmetricQ[g]` yields `True` if the adjacency matrix of g represents an antisymmetric binary relation.

New function ▪ See also `EquivalenceRelationQ`, `SymmetricQ` ▪ See page 352

■ Approximate

`Approximate` is a value that the option `Algorithm` can take in calls to functions such as `VertexCover`, telling it to use an approximation algorithm.

New function ▪ See also `VertexCover` ▪ See page 318

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ ApproximateVertexCover

`ApproximateVertexCover[g]` produces a vertex cover of graph g whose size is guaranteed to be within twice the optimal size.

New function ▪ See also `BipartiteMatchingAndCover`, `VertexCover` ▪ See page 318

■ ArticulationVertices

`ArticulationVertices[g]` gives a list of all articulation vertices in graph g. These are vertices whose removal will disconnect the graph.

See also `BiconnectedComponents`, `Bridges` ▪ See page 287

■ Automorphisms

`Automorphisms[g]` gives the automorphism group of the graph g.

See also `Isomorphism`, `PermutationGroupQ` ▪ See page 111

■ Backtrack

`Backtrack[s, partialQ, solutionQ]` performs a backtrack search of the state space s, expanding a partial solution so long as *partialQ* is `True` and returning the first complete solution, as identified by *solutionQ*.

See also `DistinctPermutations`, `Isomorphism`, `MaximumClique`, `MinimumVertexColoring` ▪ See page 311

■ BellB

`BellB[n]` returns the nth Bell number.

New function ▪ See also `SetPartitions`, `StirlingSecond` ▪ See page 154

■ BellmanFord

`BellmanFord[g, v]` gives a shortest-path spanning tree and associated distances from vertex v of graph g. The shortest-path spanning tree is given by a list in which element i is the predecessor of vertex i in the shortest-path spanning tree. `BellmanFord` works correctly even when the edge weights are negative, provided there are no negative cycles.

New function ▪ See also `AllPairsShortestPath`, `ShortestPath` ▪ See page 328

■ BiconnectedComponents

`BiconnectedComponents[g]` gives a list of the biconnected components of graph g. If g is directed, the underlying undirected graph is used.

See also `ArticulationVertices`, `BiconnectedQ`, `Bridges` ▪ See page 287

■ BiconnectedQ

`BiconnectedQ[g]` yields `True` if graph g is biconnected. If g is directed, the underlying undirected graph is used.

See also `ArticulationVertices`, `BiconnectedComponents`, `Bridges` ■ See page 287

■ BinarySearch

`BinarySearch[l, k]` searches sorted list l for key k and gives the position of l containing k, if k is present in l. Otherwise, if k is absent in l, the function returns $(p+1/2)$ where k falls between the elements of l in positions p and $p+1$. `BinarySearch[l, k, f]` gives the position of k in the list obtained from l by applying f to each element in l.

See also `SelectionSort` ■ See page 5

■ BinarySubsets

`BinarySubsets[l]` gives all subsets of l ordered according to the binary string defining each subset. For any positive integer n, `BinarySubsets[n]` gives all subsets of $\{1, 2, \ldots, n\}$ ordered according to the binary string defining each subset.

See also `Subsets` ■ See page 77

■ BipartiteMatching

`BipartiteMatching[g]` gives the list of edges associated with a maximum matching in bipartite graph g. If the graph is edge-weighted, then the function returns a matching with maximum total weight.

See also `BipartiteMatchingAndCover`, `MinimumChainPartition`, `StableMarriage` ■ See page 346

■ BipartiteMatchingAndCover

`BipartiteMatchingAndCover[g]` takes a bipartite graph g and returns a matching with maximum weight along with the dual vertex cover. If the graph is not weighted, it is assumed that all edge weights are 1.

New function ■ See also `BipartiteMatching`, `VertexCover` ■ See page 349

■ BipartiteQ

`BipartiteQ[g]` yields `True` if graph g is bipartite.

See also `CompleteGraph`, `TwoColoring` ■ See page 306

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ BooleanAlgebra

BooleanAlgebra[n] gives a Hasse diagram for the Boolean algebra on n elements. The function takes two options: Type and VertexLabel, with default values Undirected and False, respectively. When Type is set to Directed, the function produces the underlying directed acyclic graph. When VertexLabel is set to True, labels are produced for the vertices.

New function ■ See also HasseDiagram, Hypercube ■ See page 358

■ Box

Box is a value that the option VertexStyle, used in ShowGraph, can be set to.

New function ■ See also GraphOptions, SetGraphOptions, ShowGraph ■ See page 302

■ BreadthFirstTraversal

BreadthFirstTraversal[g, v] performs a breadth-first traversal of graph g starting from vertex v and gives the breadth-first numbers of the vertices. BreadthFirstTraversal[g, v, Edge] returns the edges of the graph that are traversed by breadth-first traversal. BreadthFirstTraversal[g, v, Tree] returns the breadth-first search tree. BreadthFirstTraversal[g, v, Level] returns the level number of the vertices.

See also DepthFirstTraversal ■ See page 277

■ Brelaz

Brelaz is a value that the option Algorithm can take when used in the function VertexColoring.

New function ■ See also BrelazColoring, VertexColoring ■ See page 313

■ BrelazColoring

BrelazColoring[g] returns a vertex coloring in which vertices are greedily colored with the smallest available color in decreasing order of the vertex degree.

New function ■ See also ChromaticNumber, VertexColoring ■ See page 313

■ Bridges

Bridges[g] gives a list of the bridges of graph g, where each bridge is an edge whose removal disconnects the graph.

See also ArticulationVertices, BiconnectedComponents, BiconnectedQ ■ See page 287

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ ButterflyGraph

`ButterflyGraph[n]` returns the n-dimensional butterfly graph, a directed graph whose vertices are pairs (w, i), where w is a binary string of length n and i is an integer in the range 0 through n and whose edges go from vertex (w, i) to $(w', i+1)$, if w' is identical to w in all bits with the possible exception of the $(i+1)$th bit. Here bits are counted left to right. An option `VertexLabel`, with default setting `False`, is allowed. When this option is set to `True`, vertices are labeled with strings (w, i).

New function ■ See also `DeBruijnGraph`, `Hypercube` ■ See page 253

■ CageGraph

`CageGraph[k, r]` gives a smallest k-regular graph of girth r for certain small values of k and r. `CageGraph[r]` gives `CageGraph[3, r]`. For $k = 3$, r can be 3, 4, 5, 6, 7, 8, or 10. For $k = 4$ or 5, r can be 3, 4, 5, or 6.

New function ■ See also `Girth`, `RegularGraph` ■ See page 295

■ CartesianProduct

`CartesianProduct[l1, l2]` gives the Cartesian product of lists $l1$ and $l2$.

See also `GraphJoin` ■ See page 239

■ Center

`Center` is a value that options `VertexNumberPosition`, `VertexLabelPosition`, and `EdgeLabelPosition` can take on in `ShowGraph`.

New function ■ See also `GraphOptions`, `SetGraphOptions`, `ShowGraph` ■ See page 202

■ ChangeEdges

`ChangeEdges[g, e]` replaces the edges of graph g with the edges in e. e can have the form $\{\{s_1, t_1\}, \{s_2, t_2\}, \ldots\}$ or the form $\{\{\{s_1, t_1\}, gr1\}, \{\{s_2, t_2\}, gr2\}, \ldots\}$, where $\{s_1, t_1\}, \{s_2, t_2\}, \ldots$ are endpoints of edges and $gr1, gr2, \ldots$ are graphics information associated with edges.

See also `ChangeVertices` ■ See page 192

■ ChangeVertices

`ChangeVertices[g, v]` replaces the vertices of graph g with the vertices in the given list v. v can have the form $\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$ or the form $\{\{\{x_1, y_1\}, gr1\}, \{\{x_2, y_2\}, gr2\}, \ldots\}$, where $\{x_1, y_1\}, \{x_2, y_2\}, \ldots$ are coordinates of points and $gr1, gr2, \ldots$ are graphics information associated with vertices.

See also `ChangeEdges` ■ See page 192

■ ChromaticNumber

ChromaticNumber[g] gives the chromatic number of the graph, which is the fewest number of colors necessary to color the graph.

See also ChromaticPolynomial, MinimumVertexColoring ■ See page 312

■ ChromaticPolynomial

ChromaticPolynomial[g, z] gives the chromatic polynomial $P(z)$ of graph g, which counts the number of ways to color g with, at most, z colors.

See also ChromaticNumber, MinimumVertexColoring ■ See page 308

■ ChvatalGraph

ChvatalGraph returns a smallest triangle-free, 4-regular, 4-chromatic graph.

New function ■ See also FiniteGraphs, MinimumVertexColoring ■ See page 23

■ CirculantGraph

CirculantGraph[n, l] constructs a circulant graph on n vertices, meaning that the ith vertex is adjacent to the $(i+j)$th and $(i-j)$th vertices for each j in list l. CirculantGraph[n, l], where l is an integer, returns the graph with n vertices in which each i is adjacent to $(i+l)$ and $(i-l)$.

See also CompleteGraph, Cycle ■ See page 245

■ CircularEmbedding

CircularEmbedding[n] constructs a list of n points equally spaced on a circle. CircularEmbedding[g] embeds the vertices of g equally spaced on a circle.

New function ■ See also CircularVertices ■ See page 213

■ CircularVertices

CircularVertices[n] constructs a list of n points equally spaced on a circle. CircularVertices[g] embeds the vertices of g equally spaced on a circle. This function is obsolete; use CircularEmbedding instead.

See also ChangeVertices, CompleteGraph, Cycle ■ See page 19

■ CliqueQ

CliqueQ[g, c] yields True if the list of vertices c defines a clique in graph g.

See also MaximumClique, Turan ■ See page 316

■ CoarserSetPartitionQ

`CoarserSetPartitionQ[a, b]` yields `True` if set partition b is coarser than set partition a, that is, every block in a is contained in some block in b.

New function ■ See also `SetPartitionQ`, `ToCanonicalSetPartition` ■ See page 360

■ CodeToLabeledTree

`CodeToLabeledTree[l]` constructs the unique labeled tree on n vertices from the Prüfer code l, which consists of a list of $n-2$ integers between 1 and n.

See also `LabeledTreeToCode`, `RandomTree` ■ See page 259

■ Cofactor

`Cofactor[m, i, j]` calculates the (i,j)th cofactor of matrix m.

See also `NumberOfSpanningTrees` ■ See page 339

■ CompleteBinaryTree

`CompleteBinaryTree[n]` returns a complete binary tree on n vertices.

New function ■ See also `CompleteKaryTree`, `RandomTree` ■ See page 261

■ CompleteGraph

`CompleteGraph[n]` creates a complete graph on n vertices. An option `Type` that takes on the values `Directed` or `Undirected` is allowed. The default setting for this option is `Type -> Undirected`. `CompleteGraph[a, b, c,...]` creates a complete k-partite graph of the prescribed shape. The use of `CompleteGraph` to create a complete k-partite graph is obsolete; use `CompleteKPartiteGraph` instead.

New function ■ See also `CirculantGraph`, `CompleteKPartiteGraph` ■ See page 244

■ CompleteKaryTree

`CompleteKaryTree[n, k]` returns a complete k-ary tree on n vertices.

New function ■ See also `CodeToLabeledTree`, `CompleteBinaryTree` ■ See page 261

■ CompleteKPartiteGraph

`CompleteKPartiteGraph[a, b, c, ...]` creates a complete k-partite graph of the prescribed shape, provided the k arguments $a, b, c, \ldots$ are positive integers. An option `Type` that takes on the values `Directed` or `Undirected` is allowed. The default setting for this option is `Type -> Undirected`.

New function ■ See also `CompleteGraph`, `GraphJoin`, `Turan` ■ See page 247

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ CompleteQ

`CompleteQ[g]` yields `True` if graph g is complete. This means that between any pair of vertices there is an undirected edge or two directed edges going in opposite directions.

See also `CompleteGraph`, `EmptyQ` ■ See page 198

■ Compositions

`Compositions[n, k]` gives a list of all compositions of integer n into k parts.

See also `NextComposition`, `RandomComposition` ■ See page 147

■ ConnectedComponents

`ConnectedComponents[g]` gives the vertices of graph g partitioned into connected components.

See also `BiconnectedComponents`, `ConnectedQ`, `StronglyConnectedComponents`, `WeaklyConnectedComponents` ■ See page 283

■ ConnectedQ

`ConnectedQ[g]` yields `True` if undirected graph g is connected. If g is directed, the function returns `True` if the underlying undirected graph is connected. `ConnectedQ[g, Strong]` and `ConnectedQ[g, Weak]` yield `True` if the directed graph g is strongly or weakly connected, respectively.

See also `ConnectedComponents`, `StronglyConnectedComponents`, `WeaklyConnectedComponents` ■ See page 283

■ ConstructTableau

`ConstructTableau[p]` performs the bumping algorithm repeatedly on each element of permutation p, resulting in a distinct Young tableau.

See also `DeleteFromTableau`, `InsertIntoTableau` ■ See page 164

■ Contract

`Contract[g, x, y]` gives the graph resulting from contracting the pair of vertices $\{x, y\}$ of graph g.

See also `ChromaticPolynomial`, `InduceSubgraph` ■ See page 231

■ CostOfPath

`CostOfPath[g, p]` sums up the weights of the edges in graph g defined by the path p.

See also `TravelingSalesman` ■ See page 303

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ CoxeterGraph

CoxeterGraph gives a non-Hamiltonian graph with a high degree of symmetry such that there is a graph automorphism taking any path of length 3 to any other.

New function ■ See also FiniteGraphs ■ See page 23

■ CubeConnectedCycle

CubeConnectedCycle[d] returns the graph obtained by replacing each vertex in a d-dimensional hypercube by a cycle of length d. Cube-connected cycles share many properties with hypercubes but have the additional desirable property that for $d > 1$ every vertex has degree 3.

New function ■ See also ButterflyGraph, Hypercube ■ See page 23

■ CubicalGraph

CubicalGraph returns the graph corresponding to the cube, a Platonic solid.

New function ■ See also DodecahedralGraph, FiniteGraphs ■ See page 370

■ Cut

Cut is a tag that can be used in a call to NetworkFlow to tell it to return the minimum cut.

New function ■ See also EdgeConnectivity, NetworkFlow ■ See page 292

■ CycleIndex

CycleIndex[pg, x] returns the polynomial in $x[1], x[2], \ldots, x[index[g]]$ that is the cycle index of the permutation group pg. Here, $index[pg]$ refers to the length of each permutation in pg.

New function ■ See also DistinctPermutations, OrbitInventory ■ See page 122

■ Cycle

Cycle[n] constructs the cycle on n vertices, the 2-regular connected graph. An option Type that takes on values Directed or Undirected is allowed. The default setting is Type -> Undirected.

See also AcyclicQ, RegularGraph ■ See page 248

■ Cycles

Cycles is an optional argument for the function Involutions.

New function ■ See also DistinctPermutations, Involutions ■ See page 6

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ CycleStructure

CycleStructure[p, x] returns the monomial in $x[1], x[2], \ldots, x[Length[p]]$ that is the cycle structure of the permutation p.

New function ▪ See also CycleIndex, Orbits ▪ See page 122

■ Cyclic

Cyclic is an argument to the Polya-theoretic functions ListNecklaces, NumberOfNecklace, and NecklacePolynomial, which count or enumerate distinct necklaces. Cyclic refers to the cyclic group acting on necklaces to make equivalent necklaces that can be obtained from each other by rotation.

New function ▪ See also ListNecklaces, NecklacePolynomial, NumberOfNecklaces ▪ See page 6

■ CyclicGroup

CyclicGroup[n] returns the cyclic group of permutations on n symbols.

New function ▪ See also OrbitInventory, SymmetricGroup ▪ See page 110

■ CyclicGroupIndex

CyclicGroupIndex[n, x] returns the cycle index of the cyclic group on n symbols, expressed as a polynomial in $x[1], x[2], \ldots, x[n]$.

New function ▪ See also OrbitInventory, SymmetricGroupIndex ▪ See page 124

■ DeBruijnGraph

DeBruijnGraph[m, n] constructs the n-dimensional De Bruijn graph with m symbols for integers $m > 0$ and $n > 1$. DeBruijnGraph[alph, n] constructs the n-dimensional De Bruijn graph with symbols from *alph*. Here *alph* is nonempty and $n > 1$ is an integer. In the latter form, the function accepts an option VertexLabel, with default value False, which can be set to True if users want to associate strings on *alph* to the vertices as labels.

New function ▪ See also DeBruijnSequence, Hypercube ▪ See page 257

■ DeBruijnSequence

DeBruijnSequence[a, n] returns a De Bruijn sequence on the alphabet a, a shortest sequence in which every string of length n on alphabet a occurs as a contiguous subsequence.

See also DeBruijnGraph, EulerianCycle, Strings ▪ See page 299

■ DegreeSequence

DegreeSequence[g] gives the sorted degree sequence of graph g.

See also Degrees, GraphicQ, RealizeDegreeSequence ▪ See page 266

■ Degrees

`Degrees[g]` returns the degrees of vertex $1, 2, 3, \ldots$ in that order.

New function ■ See also `DegreeSequence`, `Vertices` ■ See page 266

■ DegreesOf2Neighborhood

`DegreesOf2Neighborhood[g, v]` returns the sorted list of degrees of vertices of graph g within a distance of 2 from v.

New function ■ See also `Isomorphism` ■ See page 365

■ DeleteCycle

`DeleteCycle[g, c]` deletes a simple cycle c from graph g. c is specified as a sequence of vertices in which the first and last vertices are identical. g can be directed or undirected. If g does not contain c, it is returned unchanged; otherwise, g is returned with c deleted.

See also `ExtractCycles`, `FindCycle` ■ See page 298

■ DeleteEdge

`DeleteEdge[g, e]` gives graph g minus e. If g is undirected, then e is treated as an undirected edge; otherwise, it is treated as a directed edge. If there are multiple edges between the specified vertices, only one edge is deleted. `DeleteEdge[g, e, All]` will delete all edges between the specified pair of vertices. Using the tag `Directed` as a third argument in `DeleteEdge` is now obsolete.

See also `AddEdge`, `DeleteVertex` ■ See page 194

■ DeleteEdges

`DeleteEdges[g, l]` gives graph g minus the list of edges l. If g is undirected, then the edges in l are treated as undirected edges; otherwise, they are treated as directed edges. If there are multiple edges that qualify, then only one edge is deleted. `DeleteEdges[g, l, All]` will delete all edges that qualify. If only one edge is to be deleted, then l can have the form $\{s, t\}$; otherwise, it has the form $\{\{s_1, t_1\}, \{s_2, t_2\}, \ldots\}$.

New function ■ See also `AddEdges`, `DeleteVertex` ■ See page 194

■ DeleteFromTableau

`DeleteFromTableau[t, r]` deletes the last element of row r from Young tableaux t.

See also `ConstructTableau`, `InsertIntoTableau` ■ See page 165

■ DeleteVertex

`DeleteVertex[g, v]` deletes a single vertex v from graph g. Here v is a vertex number.

See also `AddVertex`, `DeleteEdge` ■ See page 196

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ DeleteVertices

`DeleteVertices[g, vList]` deletes vertices in *vList* from graph g. *vList* has the form $\{i, j, \ldots\}$, where $i, j, \ldots$ are vertex numbers.

New function ■ See also `AddVertices`, `DeleteEdges` ■ See page 196

■ DepthFirstTraversal

`DepthFirstTraversal[g, v]` performs a depth-first traversal of graph g starting from vertex v and gives a list of vertices in the order in which they were encountered. `DepthFirstTraversal[g, v, Edge]` returns the edges of the graph that are traversed by the depth-first traversal in the order in which they are traversed. `DepthFirstTraversal[g, v, Tree]` returns the depth-first tree of the graph.

See also `BreadthFirstTraversal` ■ See page 280

■ DerangementQ

`DerangementQ[p]` tests whether permutation p is a derangement, that is, a permutation without a fixed point.

See also `Derangements`, `NumberOfDerangements` ■ See page 107

■ Derangements

`Derangements[p]` constructs all derangements of permutation p.

See also `DerangementQ`, `NumberOfDerangements` ■ See page 107

■ Diameter

`Diameter[g]` gives the diameter of graph g, the maximum length among all pairs of vertices in g of a shortest path between each pair.

See also `Eccentricity`, `Radius` ■ See page 332

■ Dihedral

`Dihedral` is an argument to the Polya-theoretic functions `ListNecklaces`, `NumberOfNecklace`, and `NecklacePolynomial`, which count or enumerate distinct necklaces. `Dihedral` refers to the dihedral group acting on necklaces to make equivalent necklaces that can be obtained from each other by a rotation or a flip.

New function ■ See also `ListNecklaces`, `NecklacePolynomial`, `NumberOfNecklaces` ■ See page 6

■ DihedralGroup

`DihedralGroup[n]` returns the dihedral group on n symbols. Note that the order of this group is $2n$.

New function ■ See also `OrbitInventory`, `SymmetricGroup` ■ See page 110

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ DihedralGroupIndex

DihedralGroupIndex[n, x] returns the cycle index of the dihedral group on n symbols, expressed as a polynomial in $x[1], x[2], \ldots, x[n]$.

New function ■ See also OrbitInventory, SymmetricGroupIndex ■ See page 125

■ Dijkstra

Dijkstra[g, v] gives a shortest-path spanning tree and associated distances from vertex v of graph g. The shortest-path spanning tree is given by a list in which element i is the predecessor of vertex i in the shortest-path spanning tree. Dijkstra does not work correctly when the edge weights are negative; BellmanFord should be used in this case.

See also ShortestPath, ShortestPathSpanningTree ■ See page 324

■ DilateVertices

DilateVertices[v, d] multiplies each coordinate of each vertex position in list v by d, thus dilating the embedding. DilateVertices[g, d] dilates the embedding of graph g by the factor d.

See also NormalizeVertices, RotateVertices, TranslateVertices ■ See page 219

■ Directed

Directed is an option value for Type.

New function ■ See also MakeGraph, ToOrderedPairs ■ See page 188

■ Disk

Disk is a value taken by the VertexStyle option in ShowGraph.

New function ■ See also ShowGraph, VertexStyle ■ See page 202

■ Distances

Distances[g, v] returns the distances in nondecreasing order from vertex v to all vertices in g, treating g as an unweighted graph."

New function ■ See also BreadthFirstTraversal, ShortestPath ■ See page 365

■ DistinctPermutations

DistinctPermutations[l] gives all permutations of the multiset described by list l.

See also LexicographicPermutations, MinimumChangePermutations ■ See page 5

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ Distribution

Distribution[l, set] lists the frequency of each element of a set in list l.

See also RankedEmbedding ■ See page 61

■ DodecahedralGraph

DodecahedralGraph returns the graph corresponding to the dodecahedron, a Platonic solid.

New function ■ See also FiniteGraphs, OctahedralGraph ■ See page 370

■ DominatingIntegerPartitionQ

DominatingIntegerPartitionQ[a, b] yields True if integer partition a dominates integer partition b; that is, the sum of a size-t prefix of a is no smaller than the sum of a size-t prefix of b for every t.

New function ■ See also PartitionQ, Partitions ■ See page 359

■ DominationLattice

DominationLattice[n] returns a Hasse diagram of the partially ordered set on integer partitions of n in which $p < q$ if q dominates p. The function takes two options: Type and VertexLabel, with default values Undirected and False, respectively. When Type is set to Directed, the function produces the underlying directed acyclic graph. When VertexLabel is set to True, labels are produced for the vertices.

New function ■ See also HasseDiagram, Partitions ■ See page 360

■ DurfeeSquare

DurfeeSquare[p] gives the number of rows involved in the Durfee square of partition p, the side of the largest-sized square contained within the Ferrers diagram of p.

See also FerrersDiagram, TransposePartition ■ See page 8

■ Eccentricity

Eccentricity[g] gives the eccentricity of each vertex v of graph g, the maximum length among all shortest paths from v.

See also AllPairsShortestPath, Diameter, GraphCenter ■ See page 332

■ Edge

Edge is an optional argument to inform certain functions to work with edges instead of vertices.

New function ■ See also BreadthFirstTraversal, DepthFirstTraversal, NetworkFlow ■ See page 278

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ EdgeChromaticNumber

EdgeChromaticNumber[g] gives the fewest number of colors necessary to color each edge of graph g, so that no two edges incident on the same vertex have the same color.

See also ChromaticNumber, EdgeColoring ■ See page 314

■ EdgeColor

EdgeColor is an option that allows the user to associate colors with edges. Black is the default color. EdgeColor can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 204

■ EdgeColoring

EdgeColoring[g] uses Brelaz's heuristic to find a good, but not necessarily minimal, edge coloring of graph *g*.

See also LineGraph, VertexColoring ■ See page 314

■ EdgeConnectivity

EdgeConnectivity[g] gives the minimum number of edges whose deletion from graph *g* disconnects it. EdgeConnectivity[g, Cut] gives a set of edges of minimum size whose deletion disconnects the graph.

See also NetworkFlow, VertexConnectivity ■ See page 289

■ EdgeDirection

EdgeDirection is an option that takes on values True or False, allowing the user to specify whether the graph is directed or not. EdgeDirection can be set as part of the graph data structure or in ShowGraph.

New function ■ See also SetGraphOptions, ShowGraph ■ See page 204

■ EdgeLabel

EdgeLabel is an option that can take on values True or False, allowing the user to associate labels to edges. By default, there are no edge labels. The EdgeLabel option can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 204

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ EdgeLabelColor

EdgeLabelColor is an option that allows the user to associate different colors to edge labels. Black is the default color. EdgeLabelColor can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 204

■ EdgeLabelPosition

EdgeLabelPosition is an option that allows the user to place an edge label in a certain position relative to the midpoint of the edge. LowerLeft is the default value of this option. EdgeLabelPosition can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 204

■ Edges

Edges[g] gives the list of edges in *g*. Edges[g, All] gives the edges of *g* along with the graphics options associated with each edge. Edges[g, EdgeWeight] returns the list of edges in *g* along with their edge weights.

See also M, Vertices ■ See page 182

■ EdgeStyle

EdgeStyle is an option that allows the user to associate different sizes and shapes to edges. A line segment is the default edge. EdgeStyle can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 204

■ EdgeWeight

EdgeWeight is an option that allows the user to associate weights with edges; 1 is the default weight. EdgeWeight can be set as part of the graph data structure.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 197

■ Element

Element is now obsolete in *Combinatorica*, though the function call Element[a, p] still gives the *p*th element of the nested list *a*, where *p* is a list of indices.

■ EmptyGraph

EmptyGraph[n] generates an empty graph on *n* vertices. An option Type that can take on values Directed or Undirected is provided. The default setting is Type -> Undirected.

See also EmptyQ, CompleteGraph ■ See page 239

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ EmptyQ

EmptyQ[g] yields True if graph g contains no edges.

See also CompleteQ, EmptyGraph ■ See page 198

■ EncroachingListSet

EncroachingListSet[p] constructs the encroaching list set associated with permutation p.

See also Tableaux ■ See page 5

■ EquivalenceClasses

EquivalenceClasses[r] identifies the equivalence classes among the elements of matrix r.

See also EquivalenceRelationQ ■ See page 114

■ EquivalenceRelationQ

EquivalenceRelationQ[r] yields True if the matrix r defines an equivalence relation. EquivalenceRelationQ[g] tests whether the adjacency matrix of graph g defines an equivalence relation.

See also EquivalenceClasses ■ See page 114

■ Equivalences

Equivalences[g, h] lists the vertex equivalence classes between graphs g and h defined by their vertex degrees. Equivalences[g] lists the vertex equivalences for graph g defined by the vertex degrees. Equivalences[g, h, f1, f2, ...] and Equivalences[g, f1, f2, ...] can also be used, where $f1, f2, \ldots$ are functions that compute other vertex invariants. It is expected that for each function fi, the call fi[g, v] returns the corresponding invariant at vertex v in graph g. The functions $f1, f2, \ldots$ are evaluated in order, and the evaluation stops either when all functions have been evaluated or when an empty equivalence class is found. Three vertex invariants, DegreesOf2Neighborhood, NumberOf2Paths, and Distances, are *Combinatorica* functions and can be used to refine the equivalences.

See also Isomorphism, NumberOf2Paths ■ See page 364

■ Euclidean

Euclidean is an option for SetEdgeWeights.

New function ■ See also Distances, GraphOptions, SetEdgeWeights ■ See page 196

■ Eulerian

Eulerian[n, k] gives the number of permutations of length n with k runs.

See also Runs, StirlingFirst ■ See page 75

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ EulerianCycle

EulerianCycle[g] finds an Eulerian cycle of g, if one exists.

See also DeBruijnSequence, EulerianQ, HamiltonianCycle ■ See page 298

■ EulerianQ

EulerianQ[g] yields True if graph g is Eulerian, meaning that there exists a tour that includes each edge exactly once.

See also EulerianCycle, HamiltonianQ ■ See page 297

■ ExactRandomGraph

ExactRandomGraph[n, e] constructs a random labeled graph with exactly e edges and n vertices.

See also NthPair, RandomGraph, RealizeDegreeSequence ■ See page 264

■ ExpandGraph

ExpandGraph[g, n] expands graph g to n vertices by adding disconnected vertices. This is obsolete; use AddVertices[g, n] instead.

See also AddVertex, InduceSubgraph ■ See page 38

■ ExtractCycles

ExtractCycles[g] gives a maximal list of edge-disjoint cycles in graph g.

See also DeleteCycle ■ See page 294

■ FerrersDiagram

FerrersDiagram[p] draws a Ferrers diagram of integer partition p.

See also Partitions, TransposePartition ■ See page 143

■ FindCycle

FindCycle[g] finds a list of vertices that define a cycle in graph g.

See also AcyclicQ, DeleteCycle, ExtractCycles ■ See page 294

■ FindSet

FindSet[n, s] gives the root of the set containing n in the union-find data structure s.

See also InitializeUnionFind, MinimumSpanningTree, UnionSet ■ See page 336

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ FiniteGraphs

FiniteGraphs produces a convenient list of all the interesting, finite, parameterless graphs built into *Combinatorica*.

New function ■ See also CageGraph ■ See page 2

■ FirstLexicographicTableau

FirstLexicographicTableau[p] constructs the first Young tableau with shape described by partition p.

See also Tableaux ■ See page 167

■ FolkmanGraph

FolkmanGraph returns a smallest graph that is edge-transitive but not vertex-transitive.

New function ■ See also Automorphisms, FiniteGraphs ■ See page 116

■ FranklinGraph

FranklinGraph returns a 12-vertex graph that represents a 6-chromatic map on the Klein bottle. It is the sole counterexample to Heawood's map coloring conjecture.

New function ■ See also ChromaticNumber, FiniteGraphs ■ See page 23

■ FromAdjacencyLists

FromAdjacencyLists[l] constructs an edge list representation for a graph from the given adjacency lists l, using a circular embedding. FromAdjacencyLists[l, v] uses v as the embedding for the resulting graph. An option called Type that takes on the values Directed or Undirected can be used to affect the type of graph produced. The default value of Type is Undirected.

See also ToAdjacencyLists, ToOrderedPairs ■ See page 188

■ FromAdjacencyMatrix

FromAdjacencyMatrix[m] constructs a graph from a given adjacency matrix m, using a circular embedding. FromAdjacencyMatrix[m, v] uses v as the embedding for the resulting graph. An option Type that takes on the values Directed or Undirected can be used to affect the type of graph produced. The default value of Type is Undirected. FromAdjacencyMatrix[m, EdgeWeight] interprets the entries in m as edge weights, with infinity representing missing edges, and from this constructs a weighted graph using a circular embedding. FromAdjacencyMatrix[m, v, EdgeWeight] uses v as the embedding for the resulting graph. The option Type can be used along with the EdgeWeight tag.

New function ■ See also IncidenceMatrix, ToAdjacencyMatrix ■ See page 190

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ FromCycles

FromCycles[c1, c2, ...] gives the permutation that has the given cycle structure.

See also HideCycles, RevealCycles, ToCycles ■ See page 95

■ FromInversionVector

FromInversionVector[v] reconstructs the unique permutation with inversion vector v.

See also Inversions, ToInversionVector ■ See page 69

■ FromOrderedPairs

FromOrderedPairs[l] constructs an edge list representation from a list of ordered pairs l, using a circular embedding. FromOrderedPairs[l, v] uses v as the embedding for the resulting graph. The option Type that takes on values Undirected or Directed can be used to affect the kind of graph produced. The default value of Type is Directed. Type -> Undirected results in the underlying undirected graph.

See also FromUnorderedPairs, ToOrderedPairs, ToUnorderedPairs ■ See page 185

■ FromUnorderedPairs

FromUnorderedPairs[l] constructs an edge list representation from a list of unordered pairs l, using a circular embedding. FromUnorderedPairs[l, v] uses v as the embedding for the resulting graph. The option Type that takes on values Undirected or Directed can be used to affect the kind of graph produced.

See also FromOrderedPairs, ToOrderedPairs, ToUnorderedPairs ■ See page 185

■ FruchtGraph

FruchtGraph returns the smallest 3-regular graph whose automorphism group consists of only the identity.

New function ■ See also Automorphisms, FiniteGraphs ■ See page 112

■ FunctionalGraph

FunctionalGraph[f, v] takes a set v and a function f from v to v and constructs a directed graph with vertex set v and edges $(x,f(x))$ for each x in v. FunctionalGraph[f, v], where f is a list of functions, constructs a graph with vertex set v and edge set $(x,f_i(x))$ for every f_i in f. An option called Type that takes on the values Directed and Undirected is allowed. Type -> Directed is the default, while Type -> Undirected returns the corresponding underlying undirected graph.

See also IntervalGraph, MakeGraph ■ See page 271

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ GeneralizedPetersenGraph

GeneralizedPetersenGraph[n, k] returns the generalized Petersen graph, for integers $n > 1$ and $k > 0$, which is the graph with vertices $\{u_1, u_2, \cdots, u_n\}$ and $\{v_1, v_2, \cdots, v_n\}$ and edges $\{u_i, u_{i+1}\}, \{v_i, v_{i+k}\}$, and $\{u_i, v_i\}$. The Petersen graph is identical to the generalized Petersen graph with $n = 5$ and $k = 2$.

New function ■ See also FiniteGraphs, PetersenGraph ■ See page 215

■ GetEdgeLabels

GetEdgeLabels[g] returns the list of labels of the edges of g. GetEdgeLabels[g, es] returns the list of labels in graph g of the edges in es.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 197

■ GetEdgeWeights

GetEdgeWeights[g] returns the list of weights of the edges of g. GetEdgeWeights[g, es] returns the list of weights in graph g of the edges in es.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 197

■ GetVertexLabels

GetVertexLabels[g] returns the list of labels of vertices of g. GetVertexLabels[g, vs] returns the list of labels in graph g of the vertices specified in list vs.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 197

■ GetVertexWeights

GetVertexWeights[g] returns the list of weights of vertices of g. GetVertexWeights[g, vs] returns the list of weights in graph g of the vertices in vs.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 197

■ Girth

Girth[g] gives the length of the shortest cycle in a simple graph g.

See also FindCycle, ShortestPath ■ See page 296

■ Graph

Graph[e, v, opts] represents a graph object where e is the list of edges annotated with graphics options, v is a list of vertices annotated with graphics options, and *opts* is a set of global graph options. e has the form $\{\{\{i1, j1\}, opts1\}, \{\{i2, j2\}, opts2\}, \ldots\}$, where $\{i1, j1\}, \{i2, j2\}, \ldots$ are edges of the graph and $opts1, opts2, \ldots$ are options that respectively apply to these edges. v has the form $\{\{\{x1, y1\}, opts1\}, \{\{x2, y2\}, opts2\}, \ldots\}$, where $\{x1, y1\}, \{x2, y2\}, \ldots$ respectively denote the coordinates in the plane of vertices $1, 2, \ldots$ and $opts1, opts2, \ldots$ are options that respectively apply to these vertices. Permitted edge options are EdgeWeight, EdgeColor, EdgeStyle, EdgeLabel, EdgeLabelColor, and EdgeLabelPosition. Permitted vertex options are VertexWeight, VertexColor, VertexStyle, VertexNumber, VertexNumberColor, VertexNumberPosition, VertexLabel, VertexLabelColor, and VertexLabelPosition. The third item in a Graph object is *opts*, a sequence of zero or more global options that apply to all vertices or all edges or to the graph as a whole. All of the edge options and vertex options can also be used as global options. If a global option and a local edge option or vertex option differ, then the local edge or vertex option is used for that particular edge or vertex. In addition to these options, the following two options also can be specified as part of the global options: LoopPosition and EdgeDirection. Furthermore, all the options of the *Mathematica* function Plot can be used as global options in a Graph object. These can be used to specify how the graph looks when it is drawn. Also, all options of the graphics primitive Arrow also can be specified as part of the global graph options. These can be used to affect the look of arrows that represent directed edges. See the usage message of individual options to find out more about values that these options can take on. Whether a graph is undirected or directed is given by the option EdgeDirection. This has the default value False. For undirected graphs, the edges $\{i1, j1\}, \{i2, j2\}, \ldots$ have to satisfy $i1 \leq j1, i2 \leq j2, \ldots$, and for directed graphs, the edges $\{i1, j1\}, \{i2, j2\}, \ldots$ are treated as ordered pairs, each specifying the direction of the edge as well.

See also FromAdjacencyLists, FromAdjacencyMatrix ■ See page 179

■ GraphCenter

GraphCenter[g] gives a list of the vertices of graph g with minimum eccentricity.

See also AllPairsShortestPath, Eccentricity ■ See page 332

■ GraphComplement

GraphComplement[g] gives the complement of graph g.

See also SelfComplementaryQ ■ See page 238

■ GraphDifference

GraphDifference[g, h] constructs the graph resulting from subtracting the edges of graph h from the edges of graph g.

See also GraphProduct, GraphSum ■ See page 238

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ GraphicQ

`GraphicQ[s]` yields `True` if the list of integers s is a graphic sequence and thus represents a degree sequence of some graph.

See also `DegreeSequence`, `RealizeDegreeSequence` ■ See page 266

■ GraphIntersection

`GraphIntersection[g1, g2, ...]` constructs the graph defined by the edges that are in all the graphs $g1, g2, \ldots$.

See also `GraphJoin`, `GraphUnion` ■ See page 237

■ GraphJoin

`GraphJoin[g1, g2, ...]` constructs the join of graphs $g1, g2, \ldots$. This is the graph obtained by adding all possible edges between different graphs to the graph union of $g1, g2, \ldots$

See also `GraphProduct`, `GraphUnion` ■ See page 239

■ GraphOptions

`GraphOptions[g]` returns the display options associated with g. `GraphOptions[g, v]` returns the display options associated with vertex v in g. `GraphOptions[g, u, v]` returns the display options associated with edge $\{u, v\}$ in g.

New function ■ See also `SetGraphOptions` ■ See page 181

■ GraphPolynomial

`GraphPolynomial[n, x]` returns a polynomial in x in which the coefficient of x^m is the number of nonisomorphic graphs with n vertices and m edges. `GraphPolynomial[n, x, Directed]` returns a polynomial in x in which the coefficient of x^m is the number of nonisomorphic directed graphs with n vertices and m edges.

New function ■ See also `ListGraphs`, `NumberOfGraphs` ■ See page 129

■ GraphPower

`GraphPower[g, k]` gives the kth power of graph g. This is the graph whose vertex set is identical to the vertex set of g and that contains an edge between vertices i and j if g contains a path between i and j of length, at most, k.

See also `ShortestPath` ■ See page 333

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ GraphProduct

`GraphProduct[g1, g2, ...]` constructs the product of graphs $g1, g2, \ldots$.

See also `GraphDifference`, `GraphSum` ■ See page 240

■ GraphSum

`GraphSum[g1, g2, ...]` constructs the graph resulting from joining the edge lists of graphs $g1, g2, \ldots$

See also `GraphDifference`, `GraphProduct` ■ See page 238

■ GraphUnion

`GraphUnion[g1, g2, ...]` constructs the union of graphs $g1, g2, \ldots$. `GraphUnion[n, g]` constructs n copies of graph g, for any nonnegative integer n.

See also `GraphIntersection`, `GraphJoin` ■ See page 235

■ GrayCode

`GrayCode[l]` constructs a binary reflected Gray code on set l. `GrayCode` is obsolete, so use `GrayCodeSubsets` instead.

See also `GrayCodeSubsets` ■ See page 38

■ GrayCodeKSubsets

`GrayCodeKSubsets[l, k]` generates k-subsets of l in Gray code order.

New function ■ See also `GrayCodeSubsets`, `KSubsets` ■ See page 86

■ GrayCodeSubsets

`GrayCodeSubsets[l]` constructs a binary reflected Gray code on set l.

New function ■ See also `GrayCodeSubsets`, `Subsets` ■ See page 79

■ GrayGraph

`GrayGraph` returns a 3-regular, 54-vertex graph that is edge-transitive but not vertex-transitive; it is the smallest known such example.

New function ■ See also `Automorphisms`, `FiniteGraphs` ■ See page 23

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ Greedy

Greedy is a value that the option Algorithm can take in calls to functions such as VertexCover, telling the function to use a greedy algorithm.

New function ■ See also GreedyVertexCover ■ See page 31

■ GreedyVertexCover

GreedyVertexCover[g] returns a vertex cover of graph g constructed using the greedy algorithm. This is a natural heuristic for constructing a vertex cover, but it can produce poor vertex covers.

New function ■ See also BipartiteMatchingAndCover, VertexCover ■ See page 317

■ GridGraph

GridGraph[n, m] constructs an $n \times m$ grid graph, the product of paths on n and m vertices. GridGraph[p, q, r] constructs a $p \times q \times r$ grid graph, the product of GridGraph[p, q] and a path of length r.

See also GraphProduct, Path ■ See page 250

■ GrotztschGraph

GrotztschGraph returns the smallest triangle-free graph with chromatic number 4. This is identical to MycielskiGraph[4].

New function ■ See also MycielskiGraph ■ See page 23

■ HamiltonianCycle

HamiltonianCycle[g] finds a Hamiltonian cycle in graph g, if one exists. HamiltonianCycle[g, All] gives all Hamiltonian cycles of graph g.

See also EulerianCycle, HamiltonianQ, HamiltonianPath ■ See page 300

■ HamiltonianPath

HamiltonianPath[g] finds a Hamiltonian path in graph g, if one exists. HamiltonianPath[g, All] gives all Hamiltonian paths of graph g.

See also EulerianCycle, HamiltonianPath, HamiltonianQ ■ See page 302

■ HamiltonianQ

HamiltonianQ[g] yields True if there exists a Hamiltonian cycle in graph g, or, in other words, if there exists a cycle that visits each vertex exactly once.

See also EulerianQ, HamiltonianCycle ■ See page 300

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ Harary

Harary[k, n] constructs the minimal k-connected graph on n vertices.

See also EdgeConnectivity, VertexConnectivity ■ See page 292

■ HasseDiagram

HasseDiagram[g] constructs a Hasse diagram of the relation defined by directed acyclic graph g.

See also PartialOrderQ, TransitiveReduction ■ See page 357

■ Heapify

Heapify[p] builds a heap from permutation p.

See also HeapSort, RandomHeap ■ See page 5

■ HeapSort

HeapSort[l] performs a heap sort on the items of list l.

See also Heapify, SelectionSort ■ See page 5

■ HeawoodGraph

HeawoodGraph returns a smallest (6, 3)-cage, a 3-regular graph with girth 6.

New function ■ See also CageGraph, FiniteGraphs ■ See page 23

■ HerschelGraph

HerschelGraph returns a graph object that represents a Herschel graph.

New function ■ See also FiniteGraphs ■ See page 23

■ HideCycles

HideCycles[c] canonically encodes the cycle structure c into a unique permutation.

See also FromCycles, RevealCycles, ToCycles ■ See page 100

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ Highlight

`Highlight[g, p]` displays g with elements in p highlighted. The second argument p has the form $\{s_1, s_2, \ldots\}$, where the s_i's are disjoint subsets of vertices and edges of g. The options `HighlightedVertexStyle`, `HighlightedEdgeStyle`, `HighlightedVertexColors`, and `HighlightedEdgeColors` are used to determine the appearance of the highlighted elements of the graph. The default settings of the style options are `HighlightedVertexStyle->Disk[Large]` and `HighlightedEdgeStyle->Thick`. The options `HighlightedVertexColors` and `HighlightedEdgeColors` are both set to {`Black`, `Red`, `Blue`, `Green`, `Yellow`, `Purple`, `Brown`, `Orange`, `Olive`, `Pink`, `DeepPink`, `DarkGreen`, `Maroon`, `Navy`}. The colors are chosen from the palette of colors with color 1 used for s_1, color 2 used for s_2, and so on. If there are more parts than colors, then the colors are used cyclically. The function permits all the options that `SetGraphOptions` permits, for example, `VertexColor`, `VertexStyle`, `EdgeColor`, and `EdgeStyle`. These options can be used to control the appearance of the nonhighlighted vertices and edges.

New function ■ See also `AnimateGraph`, `ShowGraph` ■ See page 211

■ HighlightedEdgeColors

`HighlightedEdgeColors` is an option to `Highlight` that determines which colors are used for the highlighted edges.

New function ■ See also `GraphOptions`, `Highlight` ■ See page 211

■ HighlightedEdgeStyle

`HighlightedEdgeStyle` is an option to `Highlight` that determines how the highlighted edges are drawn.

New function ■ See also `GraphOptions`, `Highlight` ■ See page 212

■ HighlightedVertexColors

`HighlightedVertexColors` is an option to `Highlight` that determines which colors are used for the highlighted vertices.

New function ■ See also `GraphOptions`, `Highlight` ■ See page 212

■ HighlightedVertexStyle

`HighlightedVertexStyle` is an option to `Highlight` that determines how the highlighted vertices are drawn.

New function ■ See also `GraphOptions`, `Highlight` ■ See page 212

■ Hypercube

`Hypercube[n]` constructs an n-dimensional hypercube.

See also `GrayCode` ■ See page 251

■ IcosahedralGraph

`IcosahedralGraph` returns the graph corresponding to the icosahedron, a Platonic solid.

New function ■ See also `DodecahedralGraph`, `FiniteGraphs` ■ See page 370

■ IdenticalQ

`IdenticalQ[g, h]` yields `True` if graphs g and h have identical edge lists, even though the associated graphics information need not be the same.

See also `IsomorphicQ`, `Isomorphism` ■ See page 363

■ IdentityPermutation

`IdentityPermutation[n]` gives the size-n identity permutation.

New function ■ See also `DistinctPermutations`, `LexicographicPermutations` ■ See page 56

■ IncidenceMatrix

`IncidenceMatrix[g]` returns the $(0,1)$ matrix of graph g, which has a row for each vertex and a column for each edge and $(v,e) = 1$ if and only if vertex v is incident on edge e. For a directed graph, $(v,e) = 1$ if edge e is outgoing from v.

See also `LineGraph`, `ToAdjacencyMatrix` ■ See page 191

■ InDegree

`InDegree[g, n]` returns the in-degree of vertex n in the directed graph g. `InDegree[g]` returns the sequence of in-degrees of the vertices in the directed graph g.

New function ■ See also `Degrees`, `OutDegree` ■ See page 297

■ IndependentSetQ

`IndependentSetQ[g, i]` yields `True` if the vertices in list i define an independent set in graph g.

See also `CliqueQ`, `MaximumIndependentSet`, `VertexCoverQ` ■ See page 318

■ Index

`Index[p]` gives the index of permutation p, the sum of all subscripts j such that $p[j]$ is greater than $p[j+1]$.

See also `Inversions` ■ See page 73

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ InduceSubgraph

`InduceSubgraph[g, s]` constructs the subgraph of graph g induced by the list of vertices s.

See also `Contract` ■ See page 234

■ InitializeUnionFind

`InitializeUnionFind[n]` initializes a union-find data structure for n elements.

See also `FindSet`, `MinimumSpanningTree`, `UnionSet` ■ See page 336

■ InsertIntoTableau

`InsertIntoTableau[e, t]` inserts integer e into Young tableau t using the bumping algorithm. `InsertIntoTableau[e, t, All]` inserts e into Young tableau t and returns the new tableau as well as the row whose size is expanded as a result of the insertion.

See also `ConstructTableau`, `DeleteFromTableau` ■ See page 164

■ IntervalGraph

`IntervalGraph[l]` constructs the interval graph defined by the list of intervals l.

See also `FunctionalGraph`, `MakeGraph` ■ See page 319

■ Invariants

`Invariants` is an option to the functions `Isomorphism` and `IsomorphicQ` that informs these functions about which vertex invariants to use in computing equivalences between vertices.

New function ■ See also `DegreesOf2Neighborhood`, `NumberOf2Paths` ■ See page 368

■ InversePermutation

`InversePermutation[p]` yields the multiplicative inverse of permutation p.

See also `Involutions`, `Permute` ■ See page 56

■ InversionPoset

`InversionPoset[n]` returns a Hasse diagram of the partially ordered set on size-n permutations in which $p < q$ if q can be obtained from p by an adjacent transposition that places the larger element before the smaller. The function takes two options: `Type` and `VertexLabel`, with default values `Undirected` and `False`, respectively. When `Type` is set to `Directed`, the function produces the underlying directed acyclic graph. When `VertexLabel` is set to `True`, labels are produced for the vertices.

New function ■ See also `DominationLattice`, `MinimumChangePermutations` ■ See page 359

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ Inversions

Inversions[p] counts the number of inversions in permutation p.

See also FromInversionVector, ToInversionVector ▪ See page 71

■ InvolutionQ

InvolutionQ[p] yields True if permutation p is its own inverse.

See also InversePermutation, NumberOfInvolutions ▪ See page 104

■ Involutions

Involutions[l] gives the list of involutions of the elements in the list l. Involutions[l, Cycles] gives the involutions in their cycle representation. Involution[n] gives size-n involutions. Involutions[n, Cycles] gives size-n involutions in their cycle representation.

New function ▪ See also DistinctPermutations, InvolutionQ ▪ See page 105

■ IsomorphicQ

IsomorphicQ[g, h] yields True if graphs g and h are isomorphic. This function takes an option Invariants -> $\{f_1, f_2, \ldots\}$, where $f_1, f_2, \ldots$ are functions that are used to compute vertex invariants. These functions are used in the order in which they are specified. The default value of Invariants is {DegreesOf2Neighborhood, NumberOf2Paths, Distances}.

See also IdenticalQ, Isomorphism ▪ See page 367

■ Isomorphism

Isomorphism[g, h] gives an isomorphism between graphs g and h, if one exists. Isomorphism[g, h, All] gives all isomorphisms between graphs g and h. Isomorphism[g] gives the automorphism group of g. This function takes an option Invariants -> $\{f_1, f_2, \ldots\}$, where $f_1, f_2, \ldots$ are functions that are used to compute vertex invariants. These functions are used in the order in which they are specified. The default value of Invariants is {DegreesOf2Neighborhood, NumberOf2Paths, Distances}.

See also Automorphisms, IdenticalQ, IsomorphicQ ▪ See page 367

■ IsomorphismQ

IsomorphismQ[g, h, p] tests if permutation p defines an isomorphism between graphs g and h.

See also IsomorphicQ ▪ See page 367

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

Josephus

`Josephus[n, m]` generates the inverse of the permutation defined by executing every mth member in a circle of n members.

See also `InversePermutation` ■ See page 5

KnightsTourGraph

`KnightsTourGraph[m, n]` returns a graph with $m \times n$ vertices in which each vertex represents a square in an $m \times n$ chessboard and each edge corresponds to a legal move by a knight from one square to another.

New function ■ See also `GridGraph`, `HamiltonianCycle` ■ See page 302

KSetPartitions

`KSetPartitions[set, k]` returns the list of set partitions of a set with k blocks. `KSetPartitions[n, k]` returns the list of set partitions of $\{1, 2, \ldots, n\}$ with k blocks. If all set partitions of a set are needed, use the function `SetPartitions`.

New function ■ See also `RandomKSetPartition`, `SetPartitions` ■ See page 150

KSubsetGroup

`KSubsetGroup[pg, s]` returns the group induced by a permutation group pg on the set s of k-subsets of $\{1, 2, \ldots, n\}$, where n is the index of pg. The optional argument `Type` can be `Ordered` or `Unordered`, and, depending on the value of `Type`, s is treated as a set of k-subsets or k-tuples.

New function ■ See also `OrbitInventory`, `SymmetricGroup` ■ See page 114

KSubsetGroupIndex

`KSubsetGroupIndex[g, s, x]` returns the cycle index of the k-subset group on s expressed as a polynomial in $x[1], x[2], \ldots$. This function also takes the optional argument `Type` that tells the function whether the elements of s should be treated as sets or tuples.

New function ■ See also `OrbitInventory`, `SymmetricGroupIndex` ■ See page 6

KSubsets

`KSubsets[l, k]` gives all subsets of set l containing exactly k elements, ordered lexicographically.

See also `LexicographicSubsets`, `NextKSubset`, `RandomKSubset` ■ See page 83

K

The use of `K` to create a complete graph is obsolete. Use `CompleteGraph` to create a complete graph.

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ LabeledTreeToCode

LabeledTreeToCode[g] reduces the tree g to its Prüfer code.

See also CodeToLabeledTree, Strings ■ See page 258

■ Large

Large is a symbol used to denote the size of the object that represents a vertex. The option VertexStyle can be set to Disk[Large] or Box[Large] either inside the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, VertexStyle ■ See page 202

■ LastLexicographicTableau

LastLexicographicTableau[p] constructs the last Young tableau with shape described by partition p.

See also Tableaux ■ See page 167

■ Level

Level is an option for the function BreadthFirstTraversal that makes the function return levels of vertices.

See also BreadthFirstTraversal ■ See page 278

■ LeviGraph

LeviGraph returns the unique (8, 3)-cage, a 3-regular graph whose girth is 8.

New function ■ See also CageGraph, FiniteGraphs ■ See page 23

■ LexicographicPermutations

LexicographicPermutations[l] constructs all permutations of list l in lexicographic order.

See also NextPermutation, NthPermutation, RankPermutation ■ See page 58

■ LexicographicSubsets

LexicographicSubsets[l] gives all subsets of set l in lexicographic order. LexicographicSubsets[n] returns all subsets of $\{1, 2, \ldots, n\}$ in lexicographic order.

See also NthSubset, Subsets ■ See page 82

■ LineGraph

LineGraph[g] constructs the line graph of graph g.

See also IncidenceMatrix ■ See page 241

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

ListGraphs

ListGraphs[n, m] returns all nonisomorphic undirected graphs with n vertices and m edges. ListGraphs[n, m, Directed] returns all nonisomorphic directed graphs with n vertices and m edges. ListGraphs[n] returns all nonisomorphic undirected graphs with n vertices. ListGraphs[n, Directed] returns all nonisomorphic directed graphs with n vertices.

New function ▪ See also NumberOfGraphs, RandomGraph ▪ See page 121

ListNecklaces

ListNecklaces[n, c, Cyclic] returns all distinct necklaces whose beads are colored by colors from c. Here, c is a list of n, not necessarily distinct colors, and two colored necklaces are considered equivalent if one can be obtained by rotating the other. ListNecklaces[n, c, Dihedral] is similar except that two necklaces are considered equivalent if one can be obtained from the other by a rotation or a flip.

New function ▪ See also NecklacePolynomial, NumberOfNecklaces ▪ See page 6

LNorm

LNorm[p] is a value that the option WeightingFunction, used in the function SetEdgeWeights, can take. Here, p can be any integer or Infinity.

New function ▪ See also GetEdgeWeights, SetEdgeWeights ▪ See page 197

LongestIncreasingSubsequence

LongestIncreasingSubsequence[p] finds the longest increasing subsequence of permutation p.

See also Inversions, TableauClasses, Runs ▪ See page 171

LoopPosition

LoopPosition is an option to ShowGraph whose values tell ShowGraph where to position a loop around a vertex. This option can take on values UpperLeft, UpperRight, LowerLeft, and LowerRight.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 204

LowerLeft

LowerLeft is a value that options VertexNumberPosition, VertexLabelPosition, and EdgeLabelPosition can take on in ShowGraph.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 202

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ LowerRight

LowerRight is a value that options VertexNumberPosition, VertexLabelPosition, and EdgeLabelPosition can take on in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 202

■ M

M[g] gives the number of edges in the graph g. M[g, Directed] is obsolete because M[g] works for directed as well as undirected graphs.

See also Edges, V ■ See page 182

■ MakeDirected

MakeDirected[g] constructs a directed graph from a given undirected graph g by replacing each undirected edge in g by two directed edges pointing in opposite directions. The local options associated with edges are not inherited by the corresponding directed edges. Calling the function with the tag All, as MakeDirected[g, All], ensures that the local options associated with each edge are inherited by both corresponding directed edges.

New function ■ See also MakeUndirected, OrientGraph ■ See page 14

■ MakeGraph

MakeGraph[v, f] constructs the graph whose vertices correspond to v and edges between pairs of vertices x and y in v for which the binary relation defined by the Boolean function f is True. MakeGraph takes two options, Type and VertexLabel. Type can be set to Directed or Undirected, and this tells MakeGraph whether to construct a directed or an undirected graph. The default setting is Directed. VertexLabel can be set to True or False, with False being the default setting. Using VertexLabel -> True assigns labels derived from v to the vertices of the graph.

See also FunctionalGraph, IntervalGraph ■ See page 269

■ MakeSimple

MakeSimple[g] gives the undirected graph, free of multiple edges and self-loops derived from graph g.

See also MakeUndirected, SimpleQ ■ See page 254

■ MakeUndirected

MakeUndirected[g] gives the underlying undirected graph of the given directed graph g.

See also MakeSimple, UndirectedQ ■ See page 199

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ MaximalMatching

MaximalMatching[g] gives the list of edges associated with a maximal matching of graph *g*.

See also BipartiteMatching, BipartiteMatchingAndCover ■ See page 343

■ MaximumAntichain

MaximumAntichain[g] gives a largest set of unrelated vertices in partial order *g*.

See also BipartiteMatching, MinimumChainPartition, PartialOrderQ ■ See page 361

■ MaximumClique

MaximumClique[g] finds a largest clique in graph *g*. MaximumClique[g, k] returns a *k*-clique, if such a thing exists in *g*; otherwise, it returns {}.

See also CliqueQ, MaximumIndependentSet, MinimumVertexCover ■ See page 316

■ MaximumIndependentSet

MaximumIndependentSet[g] finds a largest independent set of graph *g*.

See also IndependentSetQ, MaximumClique, MinimumVertexCover ■ See page 318

■ MaximumSpanningTree

MaximumSpanningTree[g] uses Kruskal's algorithm to find a maximum spanning tree of graph *g*.

See also MinimumSpanningTree, NumberOfSpanningTrees ■ See page 336

■ McGeeGraph

McGeeGraph returns the unique (7, 3)-cage, a 3-regular graph with girth 7.

New function ■ See also CageGraph, FiniteGraphs ■ See page 23

■ MeredithGraph

MeredithGraph returns a 4-regular, 4-connected graph that is not Hamiltonian, providing a counterexample to a conjecture by C. St. J. A. Nash-Williams.

New function ■ See also FiniteGraphs, HamiltonianCycle ■ See page 303

■ MinimumChainPartition

MinimumChainPartition[g] partitions partial order *g* into a minimum number of chains.

See also BipartiteMatching, MaximumAntichain, PartialOrderQ ■ See page 361

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ MinimumChangePermutations

`MinimumChangePermutations[l]` constructs all permutations of list l such that adjacent permutations differ by only one transposition.

See also `DistinctPermutations`, `LexicographicPermutations` ■ See page 64

■ MinimumSpanningTree

`MinimumSpanningTree[g]` uses Kruskal's algorithm to find a minimum spanning tree of graph g.

See also `MaximumSpanningTree`, `NumberOfSpanningTrees`, `ShortestPathSpanningTree` ■ See page 336

■ MinimumVertexColoring

`MinimumVertexColoring[g]` returns a minimum vertex coloring of g. `MinimumVertexColoring[g, k]` returns a k-coloring of g, if one exists.

New function ■ See also `ChromaticNumber`, `VertexColoring` ■ See page 310

■ MinimumVertexCover

`MinimumVertexCover[g]` finds a minimum vertex cover of graph g. For bipartite graphs, the function uses the polynomial-time Hungarian algorithm. For everything else, the function uses brute force.

See also `MaximumClique`, `MaximumIndependentSet`, `VertexCoverQ` ■ See page 317

■ MultipleEdgesQ

`MultipleEdgesQ[g]` yields `True` if g has multiple edges between pairs of vertices. It yields `False` otherwise.

New function ■ See also `MakeSimple`, `SimpleQ` ■ See page 198

■ MultiplicationTable

`MultiplicationTable[l, f]` constructs the complete transition table defined by the binary relation function f on the elements of list l.

See also `PermutationGroupQ` ■ See page 112

■ MycielskiGraph

`MycielskiGraph[k]` returns a triangle-free graph with chromatic number k, for any positive integer k.

New function ■ See also `ChromaticNumber`, `Harary` ■ See page 312

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ NecklacePolynomial

`NecklacePolynomial[n, c, Cyclic]` returns a polynomial in the colors in *c* whose coefficients represent numbers of ways of coloring an *n*-bead necklace with colors chosen from *c*, assuming that two colorings are equivalent if one can be obtained from the other by a rotation.
`NecklacePolynomial[n, c, Dihedral]` is different in that it considers two colorings equivalent if one can be obtained from the other by a rotation or a flip or both.

New function ■ See also `ListNecklaces`, `NumberOfNecklaces` ■ See page 6

■ Neighborhood

`Neighborhood[g, v, k]` returns the subset of vertices in *g* that are at a distance of *k* or less from vertex *v*. `Neighborhood[al, v, k]` behaves identically, except that it takes as input an adjacency list *al*.

New function ■ See also `Distances`, `GraphPower` ■ See page 364

■ NetworkFlow

`NetworkFlow[g, source, sink]` returns the value of a maximum flow through graph *g* from source to sink. `NetworkFlow[g, source, sink, Edge]` returns the edges in *g* that have positive flow along with their flows in a maximum flow from source to sink. `NetworkFlow[g, source, sink, Cut]` returns a minimum cut between source and sink. `NetworkFlow[g, source, sink, All]` returns the adjacency list of *g* along with flows on each edge in a maximum flow from source to sink. *g* can be a directed or an undirected graph.

See also `EdgeConnectivity`, `NetworkFlowEdges`, `VertexConnectivity` ■ See page 341

■ NetworkFlowEdges

`NetworkFlowEdges[g, source, sink]` returns the edges of the graph with positive flow, showing the distribution of a maximum flow from source to sink in graph *g*. This is obsolete, and `NetworkFlow[g, source, sink, Edge]` should be used instead.

See also `NetworkFlow` ■ See page 38

■ NextBinarySubset

`NextBinarySubset[l, s]` constructs the subset of *l* following subset *s* in the order obtained by interpreting subsets as binary string representations of integers.

New function ■ See also `NextKSubset`, `NextSubset` ■ See page 77

■ NextComposition

`NextComposition[l]` constructs the integer composition that follows *l* in a canonical order.

See also `Compositions`, `RandomComposition` ■ See page 148

■ NextGrayCodeSubset

`NextGrayCodeSubset[l, s]` constructs the successor of *s* in the Gray code of set *l*.

New function ■ See also `NextKSubset`, `NextSubset` ■ See page 81

■ NextKSubset

`NextKSubset[l, s]` gives the *k*-subset of list *l*, following the *k*-subset *s* in lexicographic order.

See also `KSubsets`, `RandomKSubset` ■ See page 83

■ NextLexicographicSubset

`NextLexicographicSubset[l, s]` gives the lexicographic successor of subset *s* of set *l*.

New function ■ See also `NextKSubset`, `NextSubset` ■ See page 82

■ NextPartition

`NextPartition[p]` gives the integer partition following *p* in reverse lexicographic order.

See also `Partitions`, `RandomPartition` ■ See page 137

■ NextPermutation

`NextPermutation[p]` gives the permutation following *p* in lexicographic order.

See also `NthPermutation` ■ See page 57

■ NextSubset

`NextSubset[l, s]` constructs the subset of *l* following subset *s* in canonical order.

See also `NthSubset`, `RankSubset` ■ See page 6

■ NextTableau

`NextTableau[t]` gives the tableau of shape *t*, following *t* in lexicographic order.

See also `RandomTableau`, `Tableaux` ■ See page 9

■ NoMultipleEdges

`NoMultipleEdges` is an option value for `Type`.

New function ■ See also `GraphOptions`, `ShowGraph` ■ See page 31

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ NonLineGraphs

NonLineGraphs returns a graph whose connected components are the nine graphs whose presence as a vertex-induced subgraph in a graph g makes g a nonline graph.

New function ▪ See also FiniteGraphs, LineGraph ▪ See page 242

■ NoPerfectMatchingGraph

NoPerfectMatchingGraph returns a connected graph with 16 vertices that contains no perfect matching.

New function ▪ See also BipartiteMatching, FiniteGraphs ▪ See page 344

■ Normal

Normal is a value that options VertexStyle, EdgeStyle, and PlotRange can take on in ShowGraph.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 201

■ NormalDashed

NormalDashed is a value that the option EdgeStyle can take on in the graph data structure or in ShowGraph.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 204

■ NormalizeVertices

NormalizeVertices[v] gives a list of vertices with a similar embedding as v but with all coordinates of all points scaled to be between 0 and 1.

See also DilateVertices, RotateVertices, TranslateVertices ▪

■ NoSelfLoops

NoSelfLoops is an option value for Type.

New function ▪ See also GraphOptions, ShowGraph ▪ See page 31

■ NthPair

NthPair[n] returns the nth unordered pair of distinct positive integers when sequenced to minimize the size of the larger integer. Pairs that have the same larger integer are sequenced in increasing order of their smaller integer.

See also Contract, ExactRandomGraph ▪ See page 263

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ NthPermutation

NthPermutation[n, l] gives the nth lexicographic permutation of list l. This function is obsolete; use UnrankPermutation instead.

See also LexicographicPermutations, RankPermutation ■ See page 39

■ NthSubset

NthSubset[n, l] gives the nth subset of list l in canonical order.

See also NextSubset, RankSubset ■ See page 6

■ NumberOf2Paths

NumberOf2Paths[g, v, k] returns a sorted list that contains the number of paths of length 2 to different vertices of g from v.

New function ■ See also Invariants ■ See page 365

■ NumberOfCompositions

NumberOfCompositions[n, k] counts the number of distinct compositions of integer n into k parts.

See also Compositions, RandomComposition ■ See page 146

■ NumberOfDerangements

NumberOfDerangements[n] counts the derangements on n elements, that is, the permutations without any fixed points.

See also DerangementQ, Derangements ■ See page 107

■ NumberOfDirectedGraphs

NumberOfDirectedGraphs[n] returns the number of nonisomorphic directed graphs with n vertices. NumberOfDirectedGraphs[n, m] returns the number of nonisomorphic directed graphs with n vertices and m edges.

New function ■ See also ListGraphs, NumberOfGraphs ■ See page 9

■ NumberOfGraphs

NumberOfGraphs[n] returns the number of nonisomorphic undirected graphs with n vertices. NumberOfGraphs[n, m] returns the number of nonisomorphic undirected graphs with n vertices and m edges.

New function ■ See also ListGraphs, NumberOfGraphs ■ See page 129

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ NumberOfInvolutions

NumberOfInvolutions[n] counts the number of involutions on n elements.

See also InvolutionQ ■ See page 105

■ NumberOfKPaths

NumberOfKPaths[g, v, k] returns a sorted list that contains the number of paths of length k to different vertices of g from v. NumberOfKPaths[al, v, k] behaves identically, except that it takes an adjacency list *al* as input.

New function ■ See also GraphPower, Invariants ■ See page 365

■ NumberOfNecklaces

NumberOfNecklaces[n, nc, Cyclic] returns the number of distinct ways in which an n-bead necklace can be colored with *nc* colors, assuming that two colorings are equivalent if one can be obtained from the other by a rotation. NumberOfNecklaces[n, nc, Dihedral] returns the number of distinct ways in which an n-bead necklace can be colored with *nc* colors, assuming that two colorings are equivalent if one can be obtained from the other by a rotation or a flip.

New function ■ See also ListNecklaces, NecklacePolynomial ■ See page 9

■ NumberOfPartitions

NumberOfPartitions[n] counts the number of integer partitions of n.

See also Partitions, RandomPartition ■ See page 136

■ NumberOfPermutationsByCycles

NumberOfPermutationsByCycles[n, m] gives the number of permutations of length n with exactly m cycles.

See also Polya ■ See page 103

■ NumberOfPermutationsByInversions

NumberOfPermutationsByInversions[n, k] gives the number of permutations of length n with exactly k inversions. NumberOfPermutationsByInversions[n] gives a table of the number of length-n permutations with k inversions, for all k.

New function ■ See also Inversions, ToInversionVector ■ See page 73

■ NumberOfPermutationsByType

NumberOfPermutationsByTypes[l] gives the number of permutations of type l.

New function ■ See also OrbitRepresentatives, ToCycles ■ See page 99

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ NumberOfSpanningTrees

NumberOfSpanningTrees[g] gives the number of labeled spanning trees of graph g.

See also Cofactor, MinimumSpanningTree ▪ See page 399

■ NumberOfTableaux

NumberOfTableaux[p] uses the hook length formula to count the number of Young tableaux with shape defined by partition p.

See also CatalanNumber, Tableaux ▪ See page 168

■ OctahedralGraph

OctahedralGraph returns the graph corresponding to the octahedron, a Platonic solid.

New function ▪ See also DodecahedralGraph, FiniteGraphs ▪ See page 370

■ OddGraph

OddGraph[n] returns the graph whose vertices are the size-$(n-1)$ subsets of a size-$(2n-1)$ set and whose edges connect pairs of vertices that correspond to disjoint subsets. OddGraph[3] is the Petersen graph.

New function ▪ See also GeneralizedPetersenGraph, Harary ▪ See page 23

■ One

One is a tag used in several functions to inform the functions that only one object need be considered or only one solution need be produced, as opposed to all objects or all solutions.

New function ▪ See also Backtrack ▪ See page 212

■ Optimum

Optimum is a value that the option Algorithm can take on when used in the functions VertexColoring and VertexCover.

New function ▪ See also VertexColoring, VertexCover ▪ See page 31

■ OrbitInventory

OrbitInventory[ci, x, w] returns the value of the cycle index *ci* when each formal variable $x[i]$ is replaced by w. OrbitInventory[ci, x, weights] returns the inventory of orbits induced on a set of functions by the action of a group with cycle index *ci*. It is assumed that each element in the range of the functions is assigned a weight in list weights.

New function ▪ See also OrbitRepresentatives, Orbits ▪ See page 127

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ `OrbitRepresentatives`

`OrbitRepresentatives[pg, x]` returns a representative of each orbit of x induced by the action of the group pg on x. pg is assumed to be a set of permutations on the first n natural numbers, and x is a set of functions whose domain is the first n natural numbers. Each function in x is specified as an n-tuple.

New function ■ See also `OrbitInventory`, `Orbits` ■ See page 117

■ `Orbits`

`Orbits[pg, x]` returns the orbits of x induced by the action of the group pg on x. pg is assumed to be a set of permutations on the first n natural numbers, and x is a set of functions whose domain is the first n natural numbers. Each function in x is specified as an n-tuple.

New function ■ See also `OrbitInventory`, `OrbitRepresentatives` ■ See page 117

■ `Ordered`

`Ordered` is an option to the functions `KSubsetGroup` and `KSubsetGroupIndex` that tells the functions whether they should treat the input as sets or tuples.

New function ■ See also `KSubsetGroup` ■ See page 114

■ `OrientGraph`

`OrientGraph[g]` assigns a direction to each edge of a bridgeless, undirected graph g, so that the graph is strongly connected.

See also `ConnectedQ`, `StronglyConnectedComponents` ■ See page 286

■ `OutDegree`

`OutDegree[g, n]` returns the out-degree of vertex n in directed graph g. `OutDegree[g]` returns the sequence of out-degrees of the vertices in directed graph g.

New function ■ See also `Degrees`, `InDegree` ■ See page 297

■ `PairGroup`

`PairGroup[g]` returns the group induced on 2-sets by the permutation group g.
`PairGroup[g, Ordered]` returns the group induced on ordered pairs with distinct elements by the permutation group g.

New function ■ See also `OrbitInventory`, `SymmetricGroup` ■ See page 6

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ PairGroupIndex

PairGroupIndex[g, x] returns the cycle index of the pair group induced by g as a polynomial in $x[1], x[2], \ldots$. PairGroupIndex[ci, x] takes the cycle index ci of a group g with formal variables $x[1], x[2], \ldots$ and returns the cycle index of the pair group induced by g.
PairGroupIndex[g, x, Ordered] returns the cycle index of the ordered pair group induced by g as a polynomial in $x[1], x[2], \ldots$. PairGroupIndex[ci, x, Ordered] takes the cycle index ci of a group g with formal variables $x[1], x[2], \ldots$ and returns the cycle index of the ordered pair group induced by g.

New function ▪ See also OrbitInventory, SymmetricGroupIndex ▪ See page 129

■ Parent

Parent is a tag used as an argument to the function AllPairsShortestPath in order to inform this function that information about parents in the shortest paths is also wanted.

New function ▪ See also AllPairsShortestPath ▪ See page 331

■ ParentsToPaths

ParentsToPaths[l, i, j] takes a list of parents l and returns the path from i to j encoded in the parent list. ParentsToPaths[l, i] returns the paths from i to all vertices.

New function ▪ See also AllPairsShortestPath, BreadthFirstTraversal ▪ See page 31

■ PartialOrderQ

PartialOrderQ[g] yields True if the binary relation defined by edges of the graph g is a partial order, meaning it is transitive, reflexive, and antisymmetric. PartialOrderQ[r] yields True if the binary relation defined by the square matrix r is a partial order.

See also HasseDiagram, TransitiveQ ▪ See page 352

■ PartitionLattice

PartitionLattice[n] returns a Hasse diagram of the partially ordered set on set partitions of 1 through n in which $p < q$ if q is finer than p, that is, each block in q is contained in some block in p. The function takes two options: Type and VertexLabel, with default values Undirected and False, respectively. When Type is set to Directed, the function produces the underlying directed acyclic graph. When VertexLabel is set to True, labels are produced for the vertices.

New function ▪ See also DominationLattice, HasseDiagram ▪ See page 360

■ PartitionQ

PartitionQ[p] yields True if p is an integer partition. PartitionQ[n, p] yields True if p is a partition of n.

See also Partitions ▪ See page 136

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ Partitions

`Partitions[n]` constructs all partitions of integer n in reverse lexicographic order. `Partitions[n, k]` constructs all partitions of the integer n with maximum part at most k, in reverse lexicographic order.

See also `NextPartition`, `RandomPartition` ■ See page 136

■ Path

`Path[n]` constructs a tree consisting only of a path on n vertices. `Path[n]` permits an option `Type` that takes on the values `Directed` and `Undirected`. The default setting is `Type -> Undirected`.

See also `GridGraph`, `ShortestPath` ■ See page 240

■ PathConditionGraph

Usage of `PathConditionGraph` is obsolete. This functionality is no longer supported in *Combinatorica*.

■ PerfectQ

`PerfectQ[g]` yields `True` if g is a perfect graph, meaning that for every induced subgraph of g the size of the largest clique equals the chromatic number.

See also `ChromaticNumber`, `MaximumClique` ■ See page 26

■ PermutationGraph

`PermutationGraph[p]` gives the permutation graph for the permutation p.

New function ■ See also `DistinctPermutations`, `MakeGraph` ■ See page 71

■ PermutationGroupQ

`PermutationGroupQ[l]` yields `True` if the list of permutations l forms a permutation group.

See also `Automorphisms`, `MultiplicationTable` ■ See page 6

■ PermutationQ

`PermutationQ[p]` yields `True` if p is a list representing a permutation and `False` otherwise.

See also `Permute` ■ See page 55

■ PermutationToTableaux

`PermutationToTableaux[p]` returns the tableaux pair that can be constructed from p using the Robinson-Schensted-Knuth correspondence.

New function ■ See also `Involutions`, `Tableaux` ■ See page 166

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ PermutationType

`PermutationType[p]` returns the type of permutation p.

New function ■ See also `NumberOfPermutationsByType`, `ToCycles` ■ See page 98

■ PermutationWithCycle

`PermutationWithCycle[n, i, j, ...]` gives a size-n permutation in which $\{i, j, ...\}$ is a cycle and all other elements are fixed points.

New function ■ See also `FromCycles`, `ToCycles` ■ See page 96

■ Permute

`Permute[l, p]` permutes list l according to permutation p.

See also `InversePermutation`, `PermutationQ` ■ See page 55

■ PermuteSubgraph

`PermuteSubgraph[g, p]` permutes the vertices of a subgraph of g induced by p according to p.

New function ■ See also `InduceSubgraph`, `Isomorphism` ■ See page 235

■ PetersenGraph

`PetersenGraph` returns the Petersen graph, a graph whose vertices can be viewed as the size-2 subsets of a size-5 set with edges connecting disjoint subsets.

New function ■ See also `FiniteGraphs`, `GeneralizedPetersenGraph` ■ See page 190

■ PlanarQ

`PlanarQ[g]` yields `True` if graph g is planar, meaning it can be drawn in the plane so no two edges cross.

See also `ShowGraph` ■ See page 370

■ PointsAndLines

`PointsAndLines` is now obsolete.

■ Polya

`Polya[g, m]` returns the polynomial giving the number of colorings, with m colors, of a structure defined by the permutation group g. `Polya` is obsolete; use `OrbitInventory` instead.

See also `Automorphisms`, `PermutationGroupQ` ■ See page 39

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ PseudographQ

PseudographQ[g] yields True if graph g is a pseudograph, meaning it contains self-loops.

See also RemoveSelfLoops ■ See page 198

■ RadialEmbedding

RadialEmbedding[g, v] constructs a radial embedding of the graph g in which vertices are placed on concentric circles around v depending on their distance from v. RadialEmbedding[g] constructs a radial embedding of graph g, radiating from the center of the graph.

See also RandomTree, RootedEmbedding ■ See page 217

■ Radius

Radius[g] gives the radius of graph g, the minimum eccentricity of any vertex of g.

See also Diameter, Eccentricity ■ See page 332

■ RandomComposition

RandomComposition[n, k] constructs a random composition of integer n into k parts.

See also Compositions, NumberOfCompositions ■ See page 146

■ RandomGraph

RandomGraph[n, p] constructs a random labeled graph on n vertices with an edge probability of p. An option Type is provided, which can take on values Directed and Undirected and whose default value is Undirected. Type->Directed produces a corresponding random directed graph. The usages Random[n, p, Directed], Random[n, p, range], and Random[n, p, range, Directed] are all obsolete. Use SetEdgeWeights to set random edge weights.

See also ExactRandomGraph, RealizeDegreeSequence ■ See page 262

■ RandomHeap

RandomHeap[n] constructs a random heap on n elements.

See also Heapify, HeapSort ■ See page 5

■ RandomInteger

RandomInteger is a value that the WeightingFunction option of the function SetEdgeWeights can take.

New function ■ See also GraphOptions, SetEdgeWeights ■ See page 197

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ RandomKSetPartition

RandomKSetPartition[set, k] returns a random set partition of a set with k blocks. RandomKSetPartition[n, k] returns a random set partition of the first n natural numbers into k blocks.

New function ■ See also KSetPartitions, RandomSetPartition ■ See page 158

■ RandomKSubset

RandomKSubset[l, k] gives a random subset of set l with exactly k elements.

See also KSubsets, NextKSubset ■ See page 87

■ RandomPartition

RandomPartition[n] constructs a random partition of integer n.

See also NumberOfPartitions, Partitions ■ See page 144

■ RandomPermutation

RandomPermutation[n] generates a random permutation of the first n natural numbers.

See also NthPermutation ■ See page 61

■ RandomPermutation1

RandomPermutation1 is now obsolete. Use RandomPermutation instead.

See also RandomPermutation

■ RandomPermutation2

RandomPermutation2 is now obsolete. Use RandomPermutation instead.

See also RandomPermutation

■ RandomRGF

RandomRGF[n] returns a random restricted growth function (RGF) defined on the first n natural numbers. RandomRGF[n, k] returns a random RGF defined on the first n natural numbers having a maximum element equal to k.

New function ■ See also RandomSetPartition, RGFs ■ See page 9

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ RandomSetPartition

`RandomSetPartition[set]` returns a random set partition of *set*. `RandomSetPartition[n]` returns a random set partition of the first n natural numbers.

New function ■ See also `RandomPartition`, `RandomRGF` ■ See page 158

■ RandomSubset

`RandomSubset[l]` creates a random subset of set l.

See also `NthSubset`, `Subsets` ■ See page 78

■ RandomTableau

`RandomTableau[p]` constructs a random Young tableau of shape p.

See also `NextTableau`, `Tableaux` ■ See page 170

■ RandomTree

`RandomTree[n]` constructs a random labeled tree on n vertices.

See also `CodeToLabeledTree`, `TreeQ` ■ See page 260

■ RandomVertices

`RandomVertices[g]` assigns a random embedding to graph g.

See also `RandomGraph` ■ See page 304

■ RankBinarySubset

`RankBinarySubset[l, s]` gives the rank of subset s of set l in the ordering of subsets of l, obtained by interpreting these subsets as binary string representations of integers.

New function ■ See also `RankSubset`, `UnrankSubset` ■ See page 77

■ RankedEmbedding

`RankedEmbedding[l]` takes a set partition l of vertices $\{1, 2, \ldots, n\}$ and returns an embedding of the vertices in the plane such that the vertices in each block occur on a vertical line with block 1 vertices on the leftmost line, block 2 vertices on the next line, and so on. `RankedEmbedding[g, l]` takes a graph g and a set partition l of the vertices of g and returns the graph g with vertices embedded according to `RankedEmbedding[l]`. `RankedEmbedding[g, s]` takes a graph g and a set s of vertices of g and returns a ranked embedding of g in which vertices in s are in block 1, vertices at distance 1 from any vertex in block 1 are in block 2, and so on.

See also `RankGraph` ■ See page 215

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ RankGraph

RankGraph[g, l] partitions the vertices into classes based on the shortest geodesic distance to a member of list *l*.

See also RankedEmbedding ■ See page 215

■ RankGrayCodeSubset

RankGrayCodeSubset[l, s] gives the rank of subset *s* of set *l* in the Gray code ordering of the subsets of *l*.

New function ■ See also GrayCodeSubsets, UnrankGrayCodeSubset ■ See page 80

■ RankKSetPartition

RankKSetPartition[sp, s] ranks *sp* in the list of all *k*-block set partitions of *s*. RankSetPartition[sp] ranks *sp* in the list of all *k*-block set partitions of the set of elements that appear in any subset in *sp*.

New function ■ See also RankSetPartition, UnrankKSetPartition ■ See page 156

■ RankKSubset

RankKSubset[s, l] gives the rank of *k*-subset *s* of set *l* in the lexicographic ordering of *k*-subsets of *l*.

New function ■ See also UnrankGrayCodeSubset, UnrankKSubset ■ See page 84

■ RankPermutation

RankPermutation[p] gives the rank of permutation *p* in lexicographic order.

See also LexicographicPermutations, NthPermutation ■ See page 60

■ RankRGF

RankRGF[f] returns the rank of a restricted growth function (RGF) *f* in the lexicographic order of all RGFs.

New function ■ See also RankSetPartition, UnrankRGF ■ See page 9

■ RankSetPartition

RankSetPartition[sp, s] ranks *sp* in the list of all set partitions of set *s*. RankSetPartition[sp] ranks *sp* in the list of all set partitions of the set of elements that appear in any subset in *sp*.

New function ■ See also RankRGF, UnrankSetPartition ■ See page 156

■ RankSubset

RankSubset[l, s] gives the rank, in canonical order, of subset *s* of set *l*.

See also NextSubset, NthSubset ■ See page 6

■ ReadGraph

ReadGraph[f] reads a graph represented as edge lists from file *f* and returns a graph object.

See also WriteGraph

■ RealizeDegreeSequence

RealizeDegreeSequence[s] constructs a semirandom graph with degree sequence *s*.

See also GraphicQ, DegreeSequence ■ See page 267

■ ReflexiveQ

ReflexiveQ[g] yields True if the adjacency matrix of *g* represents a reflexive binary relation.

New function ■ See also PartialOrderQ, SymmetricQ ■ See page 115

■ RegularGraph

RegularGraph[k, n] constructs a semirandom *k*-regular graph on *n* vertices, if such a graph exists.

See also RealizeDegreeSequence, RegularQ ■ See page 268

■ RegularQ

RegularQ[g] yields True if *g* is a regular graph.

See also DegreeSequence, RegularGraph ■ See page 268

■ RemoveMultipleEdges

RemoveMultipleEdges[g] returns the graph obtained by deleting multiple edges from *g*.

New function ■ See also MakeSimple, MultipleEdgesQ ■ See page 199

■ RemoveSelfLoops

RemoveSelfLoops[g] returns the graph obtained by deleting self-loops in *g*.

See also PseudographQ, SimpleQ ■ See page 199

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ ResidualFlowGraph

`ResidualFlowGraph[g, flow]` returns the directed residual flow graph for a graph g with respect to *flow*.

New function ■ See also `NetworkFlow` ■ See page 341

■ RevealCycles

`RevealCycles[p]` unveils the canonical hidden cycle structure of permutation p.

See also `ToCycles`, `FromCycles`, `RevealCycles` ■ See page 100

■ ReverseEdges

`ReverseEdges[g]` flips the directions of all edges in a directed graph.

New function ■ See also `MakeUndirected`, `OrientGraph` ■ See page 14

■ RGFQ

`RGFQ[l]` yields `True` if l is a restricted growth function. It yields `False` otherwise.

New function ■ See also `RGFs`, `SetPartitionQ` ■ See page 9

■ RGFs

`RGFs[n]` lists all restricted growth functions on the first n natural numbers in lexicographic order.

New function ■ See also `KSetPartitions`, `SetPartitions` ■ See page 159

■ RGFToSetPartition

`RGFToSetPartition[rgf, set]` converts the restricted growth function *rgf* into the corresponding set partition of *set*. If the optional second argument, set, is not supplied, then *rgf* is converted into a set partition of $\{1, 2, \ldots, n\}$, where n is the length of *rgf*.

New function ■ See also `RandomRGF`, `SetPartitionToRGF` ■ See page 159

■ RobertsonGraph

`RobertsonGraph` returns a 19-vertex graph that is the unique (4, 5)-cage graph.

New function ■ See also `CageGraph`, `FiniteGraphs` ■ See page 23

■ RootedEmbedding

`RootedEmbedding[g, v]` constructs a rooted embedding of graph *g* with vertex *v* as the root. `RootedEmbedding[g]` constructs a rooted embedding with a center of *g* as the root.

See also `RadialEmbedding` ■ See page 218

■ RotateVertices

`RotateVertices[v, theta]` rotates each vertex position in list *v* by *theta* radians about the origin (0, 0). `RotateVertices[g, theta]` rotates the embedding of the graph *g* by *theta* radians about the origin (0, 0).

See also `RotateVertices`, `TranslateVertices` ■ See page 219

■ Runs

`Runs[p]` partitions *p* into contiguous increasing subsequences.

See also `Eulerian` ■ See page 74

■ SamenessRelation

`SamenessRelation[l]` constructs a binary relation from a list *l* of permutations, which is an equivalence relation if *l* is a permutation group.

See also `EquivalenceRelationQ`, `PermutationGroupQ` ■ See page 6

■ SelectionSort

`SelectionSort[l, f]` sorts list *l* using ordering function *f*.

See also `BinarySearch`, `HeapSort` ■ See page 5

■ SelfComplementaryQ

`SelfComplementaryQ[g]` yields `True` if graph *g* is self-complementary, meaning it is isomorphic to its complement.

See also `GraphComplement`, `Isomorphism` ■ See page 369

■ SelfLoopsQ

`SelfLoopsQ[g]` yields `True` if graph *g* has self-loops.

New function ■ See also `MultipleEdgesQ`, `SimpleQ` ■ See page 26

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ SetEdgeLabels

SetEdgeLabels[g, l] assigns the labels in l to edges of g. If l is shorter than the number of edges in g, then labels get assigned cyclically. If l is longer than the number of edges in g, then the extra labels are ignored.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 197

■ SetEdgeWeights

SetEdgeWeights[g] assigns random real weights in the range $[0,1]$ to edges in g. SetEdgeWeights accepts options WeightingFunction and WeightRange. WeightingFunction can take values Random, RandomInteger, Euclidean, or LNorm[n] for nonnegative n, or any pure function that takes two arguments, each argument having the form {Integer, {Number, Number}}. WeightRange can be an integer range or a real range. The default value for WeightingFunction is Random, and the default value for WeightRange is $[0,1]$. SetEdgeWeights[g, e] assigns edge weights to the edges in the edge list e. SetEdgeWeights[g, w] assigns the weights in the weight list w to the edges of g. SetEdgeWeights[g, e, w] assigns the weights in the weight list w to the edges in edge list e.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 197

■ SetGraphOptions

SetGraphOptions[g, opts] returns g with the options *opts* set.
SetGraphOptions[g, v1, v2, ..., vopts, gopts] returns the graph with the options *vopts* set for vertices $v1, v2, \ldots$ and the options *gopts* set for the graph g.
SetGraphOptions[g, e1, e2,..., eopts, gopts], with edges $e1, e2, \ldots$, works similarly.
SetGraphOptions[g, elements1, opts1, elements2, opts2,..., opts] returns g with the options *opts*1 set for the elements in the sequence *elements*1, the options *opts*2 set for the elements in the sequence *elements*2, and so on. Here, elements can be a sequence of edges or a sequence of vertices. A tag that takes on values One or All can also be passed in as an argument before any options. The default value of the tag is All, and it is useful if the graph has multiple edges. It informs the function about whether all edges that connect a pair of vertices are to be affected or only one edge is affected.

New function ▪ See also GraphOptions ▪ See page 196

■ SetPartitionListViaRGF

SetPartitionListViaRGF[n] lists all set partitions of the first n natural numbers, by first listing all restricted growth functions (RGFs) on these and then mapping the RGFs to corresponding set partitions. SetPartitionListViaRGF[n, k] lists all RGFs on the first n natural numbers whose maximum element is k and then maps these RGFs into the corresponding set partitions, all of which contain exactly k blocks.

New function ▪ See also RGFs, SetPartitions ▪ See page 9

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ SetPartitionQ

SetPartitionQ[sp, s] determines if *sp* is a set partition of set *s*. SetPartitionQ[sp] tests if *sp* is a set of disjoint sets.

New function ■ See also CoarserSetPartitionQ, PartitionQ ■ See page 149

■ SetPartitions

SetPartitions[s] returns the list of set partitions of *s*. SetPartitions[n] returns the list of set partitions of $\{1, 2, \ldots, n\}$. If all set partitions with a fixed number of subsets are needed, use KSetPartitions.

New function ■ See also KSetPartitions, RandomSetPartition ■ See page 152

■ SetPartitionToRGF

SetPartitionToRGF[sp, s] converts the set partition *sp* of set *s* into the corresponding restricted growth function. If the optional argument *s* is not specified, then it is assumed that *Mathematica* knows the underlying order on the set for which *sp* is a set partition.

New function ■ See also RGFToSetPartition, RGFs ■ See page 159

■ SetVertexLabels

SetVertexLabels[g, l] assigns the labels in *l* to vertices of *g*. If *l* is shorter than the number of vertices in *g*, then labels get assigned cyclically. If *l* is longer than the number of vertices in *g*, then the extra labels are ignored.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 197

■ SetVertexWeights

SetVertexWeights[g] assigns random real weights in the range $[0, 1]$ to vertices in *g*. SetVertexWeights accepts options WeightingFunction and WeightRange. WeightingFunction can take values Random, RandomInteger, or any pure function that takes two arguments, an integer as the first argument and a pair $\{number, number\}$ as the second argument. WeightRange can be an integer range or a real range. The default value for WeightingFunction is Random, and the default value for WeightRange is $[0, 1]$. SetVertexWeights[g, w] assigns the weights in the weight list *w* to the vertices of *g*. SetVertexWeights[g, vs, w] assigns the weights in the weight list *w* to the vertices in the vertex list *vs*.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 197

■ ShakeGraph

ShakeGraph[g, d] performs a random perturbation of the vertices of graph *g*, with each vertex moving, at most, a distance *d* from its original position.

See also ShowGraph, SpringEmbedding ■ See page 220

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

ShortestPath

`ShortestPath[g, start, end]` finds a shortest path between vertices *start* and *end* in graph *g*. An option `Algorithm` that takes on the values `Automatic`, `Dijkstra`, or `BellmanFord` is provided. This allows a choice between using Dijkstra's algorithm and the Bellman-Ford algorithm. The default is `Algorithm -> Automatic`. In this case, depending on whether edges have negative weights and depending on the density of the graph, the algorithm chooses between Bellman-Ford and Dijkstra.

See also `AllPairsShortestPath`, `ShortestPathSpanningTree` ▪ See page 329

ShortestPathSpanningTree

`ShortestPathSpanningTree[g, v]` constructs a shortest-path spanning tree rooted at *v*, so that a shortest path in graph *g* from *v* to any other vertex is a path in the tree. An option `Algorithm` that takes on the values `Automatic`, `Dijkstra`, or `BellmanFord` is provided. This allows a choice between Dijkstra's algorithm and the Bellman-Ford algorithm. The default is `Algorithm -> Automatic`. In this case, depending on whether edges have negative weights and depending on the density of the graph, the algorithm chooses between Bellman-Ford and Dijkstra.

See also `AllPairsShortestPath`, `MinimumSpanningTree`, `ShortestPath` ▪ See page 329

ShowGraph

`ShowGraph[g]` displays the graph *g*. `ShowGraph[g, options]` modifies the display using the given options. `ShowGraph[g, Directed]` is obsolete and is currently identical to `ShowGraph[g]`. All options that affect the look of a graph can be specified as options in `ShowGraph`. The list of options is `VertexColor`, `VertexStyle`, `VertexNumber`, `VertexNumberColor`, `VertexNumberPosition`, `VertexLabel`, `VertexLabelColor`, `VertexLabelPosition`, `EdgeColor`, `EdgeStyle`, `EdgeLabel`, `EdgeLabelColor`, `EdgeLabelPosition`, `LoopPosition`, and `EdgeDirection`. In addition, options of the Mathematica function `Plot` and options of the graphics primitive `Arrow` can also be specified here. If an option specified in `ShowGraph` differs from options explicitly set within a graph object, then options specified inside the graph object are used.

See also `AnimateGraph`, `ShowGraphArray` ▪ See page 200

ShowGraphArray

`ShowGraphArray[{g1, g2, ...}]` displays a row of graphs.
`ShowGraphArray[{ {g1, ...}, {g2, ...}, ...}]` displays a two-dimensional table of graphs.
`ShowGraphArray` accepts all the options accepted by `ShowGraph`, and the user can also provide the option `GraphicsSpacing -> d`.

New function ▪ See also `AnimateGraph`, `ShowGraph` ▪ See page 210

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ ShowLabeledGraph

ShowLabeledGraph[g] displays graph g according to its embedding, with each vertex labeled with its vertex number. ShowLabeledGraph[g, l] uses the ith element of list l as the label for vertex i.

See also ShowGraph ■ See page 19

■ ShuffleExchangeGraph

ShuffleExchangeGraph[n] returns the n-dimensional shuffle-exchange graph whose vertices are length-n binary strings with an edge from w to w' if (i) w' differs from w in its last bit or (ii) w' is obtained from w by a cyclic shift left or a cyclic shift right. An option VertexLabel is provided, with default setting False, which can be set to True if the user wants to associate the binary strings to the vertices as labels.

New function ■ See also ButterflyGraph, DeBruijnGraph, Hypercube ■ See page 255

■ SignaturePermutation

SignaturePermutation[p] gives the signature of permutation p.

See also MinimumChangePermutations, ToCycles ■ See page 97

■ Simple

Simple is an option value for Type.

New function ■ See also MakeGraph, ToAdjacencyMatrix ■ See page 185

■ SimpleQ

SimpleQ[g] yields True if g is a simple graph, meaning it has no multiple edges and contains no self-loops.

See also PseudographQ, UnweightedQ ■ See page 198

■ Small

Small is a symbol used to denote the size of the object that represents a vertex. The option VertexStyle can be set to Disk[Small] or Box[Small] either inside the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 202

■ SmallestCyclicGroupGraph

SmallestCyclicGroupGraph returns a smallest nontrivial graph whose automorphism group is cyclic.

New function ■ See also Automorphisms, FiniteGraphs ■ See page 23

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ Spectrum

`Spectrum[g]` gives the eigenvalues of graph g.

See also `Edges` ■ See page 27

■ SpringEmbedding

`SpringEmbedding[g]` beautifies the embedding of graph g by modeling the embedding as a system of springs. `SpringEmbedding[g, step, increment]` can be used to refine the algorithm. The value of *step* tells the function how many iterations to run the algorithm. The value of *increment* tells the function the distance to move the vertices at each step. The default values are 10 and 0.15 for *step* and *increment*, respectively.

See also `ShakeGraph`, `ShowGraph` ■ See page 221

■ StableMarriage

`StableMarriage[mpref, fpref]` finds the male optimal stable marriage defined by lists of permutations describing male and female preferences.

See also `BipartiteMatching`, `MaximalMatching` ■ See page 350

■ Star

`Star[n]` constructs a star on n vertices, which is a tree with one vertex of degree $n - 1$.

See also `TreeQ`, `Wheel` ■ See page 248

■ StirlingFirst

`StirlingFirst[n, k]` returns a Stirling number of the first kind. This is obsolete. Use the built-in *Mathematica* function `StirlingS1` instead.

See also `NumberOfPermutationsByCycles` ■ See page 103

■ StirlingSecond

`StirlingSecond[n, k]` returns a Stirling number of the second kind.

See also `KSetPartitions` and `BellB` ■ See page 153

■ Strings

`Strings[l, n]` constructs all possible strings of length n from the elements of list l.

See also `CodeToLabeledTree`, `DistinctPermutations` ■ See page 88

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ StronglyConnectedComponents

StronglyConnectedComponents[g] gives the strongly connected components of directed graph g as lists of vertices.

See also ConnectedQ, WeaklyConnectedComponents ■ See page 285

■ Strong

Strong is an option to ConnectedQ that seeks to determine if a directed graph is strongly connected.

New function ■ See also ConnectedQ, StronglyConnectedComponents ■ See page 31

■ Subsets

Subsets[l] gives all subsets of set l.

See also BinarySubsets, GrayCode, LexicographicSubsets ■ See page 6

■ SymmetricGroup

SymmetricGroup[n] returns the symmetric group on n symbols.

New function ■ See also AlternatingGroup, OrbitInventory ■ See page 110

■ SymmetricGroupIndex

SymmetricGroupIndex[n, x] returns the cycle index of the symmetric group on n symbols, expressed as a polynomial in $x[1], x[2], \ldots, x[n]$.

New function ■ See also AlternatingGroupIndex, OrbitInventory ■ See page 123

■ SymmetricQ

SymmetricQ[r] tests if a given square matrix r represents a symmetric relation. SymmetricQ[g] tests if the edges of a given graph represent a symmetric relation.

New function ■ See also PartialOrderQ, ReflexiveQ ■ See page 115

■ TableauClasses

TableauClasses[p] partitions the elements of permutation p into classes according to their initial columns during Young tableaux construction.

See also InsertIntoTableau, LongestIncreasingSubsequence ■ See page 171

■ TableauQ

TableauQ[t] yields True if and only if t represents a Young tableau.

See also RandomTableau, Tableaux ▪ See page 162

■ Tableaux

Tableaux[p] constructs all tableaux having a shape given by integer partition p.

See also NextTableau, RandomTableau ▪ See page 163

■ TableauxToPermutation

TableauxToPermutation[t1, t2] constructs the unique permutation associated with Young tableaux $t1$ and $t2$, where both tableaux have the same shape.

See also DeleteFromTableau, InsertIntoTableau ▪ See page 166

■ TetrahedralGraph

TetrahedralGraph returns the graph corresponding to the tetrahedron, a Platonic solid.

New function ▪ See also DodecahedralGraph, FiniteGraphs ▪ See page 370

■ Thick

Thick is a value that the option EdgeStyle can take on in the graph data structure or in ShowGraph.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 204

■ ThickDashed

ThickDashed is a value that the option EdgeStyle can take on in the graph data structure or in ShowGraph.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 204

■ Thin

Thin is a value that the option EdgeStyle can take on in the graph data structure or in ShowGraph.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 204

■ ThinDashed

ThinDashed is a value that the option EdgeStyle can take on in the graph data structure or in ShowGraph.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 204

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ ThomassenGraph

ThomassenGraph returns a hypotraceable graph, a graph G that has no Hamiltonian path but whose subgraph $G - v$ for every vertex v has a Hamiltonian path.

New function ■ See also FiniteGraphs, HamiltonianCycle ■ See page 303

■ ToAdjacencyLists

ToAdjacencyLists[g] constructs an adjacency list representation for graph g. It allows an option called Type that takes on values All or Simple. Type -> All is the default setting of the option, and this permits self-loops and multiple edges to be reported in the adjacency lists. Type -> Simple deletes self-loops and multiple edges from the constructed adjacency lists. ToAdjacencyLists[g, EdgeWeight] returns an adjacency list representation along with edge weights.

See also FromAdjacencyLists, ToOrderedPairs ■ See page 186

■ ToAdjacencyMatrix

ToAdjacencyMatrix[g] constructs an adjacency matrix representation for graph g. An option Type that takes on values All or Simple can be used to affect the matrix constructed. Type -> All is the default, and Type -> Simple ignores any self-loops or multiple edges that g may have. ToAdjacencyMatrix[g, EdgeWeight] returns edge weights as entries of the adjacency matrix with Infinity representing missing edges.

New function ■ See also FromAdjacencyMatrix, ToAdjacencyLists ■ See page 189

■ ToCanonicalSetPartition

ToCanonicalSetPartition[sp, s] reorders *sp* into a canonical order with respect to *s*. In the canonical order, the elements of each subset of the set partition are ordered as they appear in *s*, and the subsets themselves are ordered by their first elements. ToCanonicalSetPartition[sp] reorders *sp* into canonical order, assuming that *Mathematica* knows the underlying order on the set for which *sp* is a set partition.

New function ■ See also KSetPartitions, SetPartitions ■ See page 149

■ ToCycles

ToCycles[p] gives the cycle structure of permutation p as a list of cyclic permutations.

See also FromCycles, HideCycles, RevealCycles ■ See page 94

■ ToInversionVector

ToInversionVector[p] gives the inversion vector associated with permutation p.

See also FromInversionVector, Inversions ■ See page 69

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ ToOrderedPairs

`ToOrderedPairs[g]` constructs a list of ordered pairs representing the edges of the graph g. If g is undirected, each edge is interpreted as two ordered pairs. An option called `Type` that takes on values `Simple` or `All` can be used to affect the constructed representation. `Type -> Simple` forces the removal of multiple edges and self-loops. `Type -> All` keeps all information and is the default option.

See also `FromOrderedPairs`, `FromUnorderedPairs`, `ToUnorderedPairs` ■ See page 184

■ TopologicalSort

`TopologicalSort[g]` gives a permutation of the vertices of directed acyclic graph g such that an edge (i, j) implies that vertex i appears before vertex j.

See also `AcyclicQ`, `PartialOrderQ` ■ See page 352

■ ToUnorderedPairs

`ToUnorderedPairs[g]` constructs a list of unordered pairs representing the edges of graph g. Each edge, directed or undirected, results in a pair in which the smaller vertex appears first. An option called `Type` that takes on values `All` or `Simple` can be used, and `All` is the default value. `Type -> Simple` ignores multiple edges and self-loops in g.

See also `FromOrderedPairs`, `FromUnorderedPairs`, `ToOrderedPairs` ■ See page 184

■ TransitiveClosure

`TransitiveClosure[g]` finds the transitive closure of graph g, the supergraph of g that contains edge $\{x, y\}$ if and only if there is a path from x to y.

See also `TransitiveQ`, `TransitiveReduction` ■ See page 354

■ TransitiveQ

`TransitiveQ[g]` yields `True` if graph g defines a transitive relation.

See also `PartialOrderQ`, `TransitiveClosure`, `TransitiveReduction` ■ See page 115

■ TransitiveReduction

`TransitiveReduction[g]` finds a smallest graph that has the same transitive closure as g.

See also `HasseDiagram`, `TransitiveClosure` ■ See page 355

■ TranslateVertices

`TranslateVertices[v, x, y]` adds the vector $\{x, y\}$ to the vertex embedding location of each vertex in list v. `TranslateVertices[g, x, y]` translates the embedding of the graph g by the vector $\{x, y\}$.

See also `NormalizeVertices` ■ See page 220

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ TransposePartition

`TransposePartition[p]` reflects a partition p of k parts along the main diagonal, creating a partition with maximum part k.

See also `DurfeeSquare`, `FerrersDiagram` ▪ See page 143

■ TransposeTableau

`TransposeTableau[t]` reflects a Young tableau t along the main diagonal, creating a different tableau.

See also `Tableaux` ▪ See page 162

■ TravelingSalesman

`TravelingSalesman[g]` finds an optimal traveling salesman tour in graph g.

See also `HamiltonianCycle`, `TravelingSalesmanBounds` ▪ See page 303

■ TravelingSalesmanBounds

`TravelingSalesmanBounds[g]` gives upper and lower bounds on a minimum cost traveling salesman tour of graph g.

See also `TravelingSalesman`, `TriangleInequalityQ` ▪ See page 31

■ Tree

`Tree` is an option that informs certain functions for which the user wants the output to be a tree.

New function ▪ See also `BreadthFirstTraversal`, `DepthFirstTraversal` ▪ See page 278

■ TreeIsomorphismQ

`TreeIsomorphismQ[t1, t2]` yields `True` if the trees $t1$ and $t2$ are isomorphic and `False` otherwise.

New function ▪ See also `Isomorphism`, `TreeToCertificate` ▪ See page 369

■ TreeQ

`TreeQ[g]` yields `True` if graph g is a tree.

See also `AcyclicQ`, `RandomTree` ▪ See page 295

■ TreeToCertificate

`TreeToCertificate[t]` returns a binary string that is a certificate for the tree t such that trees have the same certificate if and only if they are isomorphic.

New function ▪ See also `Isomorphism`, `TreeIsomorphismQ` ▪ See page 369

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ TriangleInequalityQ

TriangleInequalityQ[g] yields True if the weights assigned to the edges of graph g satisfy the triangle inequality.

See also AllPairsShortestPath, TravelingSalesmanBounds ■ See page 304

■ Turan

Turan[n, p] constructs the Turan graph, the extremal graph on n vertices that does not contain K_p, the complete graph on p vertices.

See also CompleteKPartiteGraph, MaximumClique ■ See page 247

■ TutteGraph

TutteGraph returns the Tutte graph, the first known example of a 3-connected, 3-regular, planar graph that is non-Hamiltonian.

New function ■ See also FiniteGraphs, HamiltonianCycle ■ See page 301

■ TwoColoring

TwoColoring[g] finds a two-coloring of graph g if g is bipartite. It returns a list of the labels 1 and 2 corresponding to the vertices. This labeling is a valid coloring if and only the graph is bipartite.

See also BipartiteQ, CompleteKPartiteGraph ■ See page 306

■ Type

Type is an option for many functions that transform graphs. Depending on the functions it is being used in, it can take on values such as Directed, Undirected, Simple, etc.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 184

■ Undirected

Undirected is an option to inform certain functions that the graph is undirected.

New function ■ See also GraphOptions, SetGraphOptions

■ UndirectedQ

UndirectedQ[g] yields True if graph g is undirected.

See also MakeUndirected ■ See page 198

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ UnionSet

`UnionSet[a, b, s]` merges the sets containing a and b in union-find data structure s.

See also `FindSet`, `InitializeUnionFind`, `MinimumSpanningTree` ■ See page 336

■ Uniquely3ColorableGraph

`Uniquely3ColorableGraph` returns a 12-vertex, triangle-free graph with chromatic number 3 that is uniquely 3-colorable.

New function ■ See also `ChromaticPolynomial`, `FiniteGraphs` ■ See page 23

■ UnitransitiveGraph

`UnitransitiveGraph` returns a 20-vertex, 3-unitransitive graph discovered by Coxeter that is not isomorphic to a 4-cage or a 5-cage.

New function ■ See also `FiniteGraphs`, `Isomorphism` ■ See page 23

■ UnrankBinarySubset

`UnrankBinarySubset[n, l]` gives the nth subset of list l, listed in increasing order of integers corresponding to the binary representations of the subsets.

New function ■ See also `BinarySubsets`, `RankBinarySubset` ■ See page 77

■ UnrankGrayCodeSubset

`UnrankGrayCodeSubset[n, l]` gives the nth subset of list l, listed in Gray code order.

New function ■ See also `GrayCodeSubsets`, `RankGrayCodeSubset` ■ See page 80

■ UnrankKSetPartition

`UnrankSetPartition[r, s, k]` finds a k-block set partition of s with rank r.
`UnrankSetPartition[r, n, k]` finds a k-block set partition of $\{1, 2, \ldots, n\}$ with rank r.

New function ■ See also `RankKSetPartition`, `UnrankSetPartition` ■ See page 157

■ UnrankKSubset

`UnrankKSubset[m, k, l]` gives the mth k-subset of set l, listed in lexicographic order.

New function ■ See also `RankKSubset`, `UnrankSubset` ■ See page 85

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ UnrankPermutation

UnrankPermutation[r, l] gives the rth permutation in the lexicographic list of permutations of list l. UnrankPermutation[r, n] gives the rth permutation in the lexicographic list of permutations of $\{1, 2, \ldots, n\}$.

New function ▪ See also DistinctPermutations, RankPermutation ▪ See page 60

■ UnrankRGF

UnrankRGF[r, n] returns a restricted growth function defined on the first n natural numbers whose rank is r.

New function ▪ See also RankRGF, UnrankSetPartition ▪ See page 9

■ UnrankSetPartition

UnrankSetPartition[r, s] finds a set partition of s with rank r. UnrankSetPartition[r, n] finds a set partition of $\{1, 2, \ldots, n\}$ with rank r.

New function ▪ See also RankSetPartition, UnrankKSetPartition ▪ See page 158

■ UnrankSubset

UnrankSubset[n, l] gives the nth subset of list l, listed in some canonical order.

New function ▪ See also RankSubset, UnrankGrayCodeSubset ▪ See page 6

■ UnweightedQ

UnweightedQ[g] yields True if all edge weights are 1 and False otherwise.

See also SimpleQ ▪ See page 198

■ UpperLeft

UpperLeft is a value that options VertexNumberPosition, VertexLabelPosition, and EdgeLabelPosition can take on in ShowGraph.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 202

■ UpperRight

UpperRight is a value that options VertexNumberPosition, VertexLabelPosition, and EdgeLabelPosition can take on in ShowGraph.

New function ▪ See also GraphOptions, SetGraphOptions ▪ See page 202

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ V

V[g] gives the order or number of vertices of the graph *g*.

See also M, Vertices ■ See page 182

■ Value

Value is an option for the function NetworkFlow that makes the function return the value of the maximum flow.

New function ■ See also NetworkFlow

■ VertexColor

VertexColor is an option that allows the user to associate colors with vertices. Black is the default color. VertexColor can be set as part of the graph data structure, and it can be used in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 202

■ VertexColoring

VertexColoring[g] uses Brelaz's heuristic to find a good, but not necessarily minimal, vertex coloring of graph *g*. An option Algorithm that can take on the values Brelaz or Optimum is allowed. The setting Algorithm -> Brelaz is the default, while the setting Algorithm -> Optimum forces the algorithm to do an exhaustive search to find an optimum vertex coloring.

See also ChromaticNumber, ChromaticPolynomial, EdgeColoring ■ See page 313

■ VertexConnectivity

VertexConnectivity[g] gives the minimum number of vertices whose deletion from graph *g* disconnects it. VertexConnectivity[g, Cut] gives a set of vertices of minimum size, whose removal disconnects the graph.

See also EdgeConnectivity, Harary, NetworkFlow ■ See page 291

■ VertexConnectivityGraph

VertexConnectivityGraph[g] returns a directed graph that contains an edge corresponding to each vertex in *g* and in which edge-disjoint paths correspond to vertex-disjoint paths in *g*.

New function ■ See also NetworkFlow, VertexConnectivity ■ See page 291

■ VertexCover

VertexCover[g] returns a vertex cover of the graph *g*. An option Algorithm that can take on values Greedy, Approximate, or Optimum is allowed. The default setting is Algorithm -> Approximate. Different algorithms are used to compute a vertex cover depending on the setting of the option Algorithm.

New function ■ See also BipartiteMatchingAndCover, MinimumVertexCover ■ See page 317

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ VertexCoverQ

VertexCoverQ[g, c] yields True if the vertices in list *c* define a vertex cover of graph *g*.

See also CliqueQ, IndependentSetQ, MinimumVertexCover ■ See page 317

■ VertexLabel

VertexLabel is an option that can take on values True or False, allowing the user to set and display vertex labels. By default, there are no vertex labels. VertexLabel can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 202

■ VertexLabelColor

VertexLabelColor is an option that allows the user to associate different colors to vertex labels. Black is the default color. VertexLabelColor can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 202

■ VertexLabelPosition

VertexLabelPosition is an option that allows the user to place a vertex label in a certain position relative to the vertex. The default position is upper right. VertexLabelPosition can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 202

■ VertexNumber

VertexNumber is an option that can take on values True or False. This can be used in ShowGraph to display or suppress vertex numbers. By default, the vertex numbers are hidden. VertexNumber can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, ShowGraph ■ See page 202

■ VertexNumberColor

VertexNumberColor is an option that can be used in ShowGraph to associate different colors to vertex numbers. Black is the default color. VertexNumberColor can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 202

■ VertexNumberPosition

VertexNumberPosition is an option that can be used in ShowGraph to display a vertex number in a certain position relative to the vertex. By default, vertex numbers are positioned to the lower left of the vertices. VertexNumberPosition can be set as part of the graph data structure or in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 202

■ VertexStyle

VertexStyle is an option that allows the user to associate different sizes and shapes to vertices. A disk is the default shape. VertexStyle can be set as part of the graph data structure, and it can be used in ShowGraph.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 202

■ VertexWeight

VertexWeight is an option that allows the user to associate weights with vertices. 0 is the default weight. VertexWeight can be set as part of the graph data structure.

New function ■ See also GraphOptions, SetGraphOptions ■ See page 17

■ Vertices

Vertices[g] gives the embedding of graph g, that is, the coordinates of each vertex in the plane. Vertices[g, All] gives the embedding of the graph along with graphics options associated with each vertex.

See also Edges, V ■ See page 182

■ WaltherGraph

WaltherGraph returns the Walther graph.

New function ■ See also FiniteGraphs ■ See page 23

■ Weak

Weak is an option to ConnectedQ that seeks to determine if a directed graph is weakly connected.

New function ■ See also ConnectedQ, WeaklyConnectedComponents ■ See page 31

■ WeaklyConnectedComponents

WeaklyConnectedComponents[g] gives the weakly connected components of directed graph g as lists of vertices.

See also ConnectedQ, StronglyConnectedComponents ■ See page 285

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press

■ `WeightingFunction`

`WeightingFunction` is an option to the functions `SetEdgeWeights` and `SetVertexWeights`, and it tells the functions how to compute edge-weights and vertex-weights, respectively. The default value for this option is `Random`.

New function ▪ See also `Euclidean`, `LNorm` ▪ See page 197

■ `WeightRange`

`WeightRange` is an option to the functions `SetEdgeWeights` and `SetVertexWeights` that gives the range for these weights. The default range is $[0, 1]$ for real as well as integer weights.

New function ▪ See also `GraphOptions`, `SetGraphOptions` ▪ See page 197

■ `Wheel`

`Wheel[n]` constructs a wheel on n vertices, which is the join of a single vertex and the cycle with $n - 1$ vertices.

See also `Cycle`, `Star` ▪ See page 249

■ `WriteGraph`

`WriteGraph[g, f]` writes graph g to file f using an edge list representation.

See also `ReadGraph`

■ `Zoom`

`Zoom[{i, j, k, ...}]` is a value that the `PlotRange` option can take on in `ShowGraph`. Setting `PlotRange` to this value zooms the display to contain the specified subset of vertices, $i, j, k, \ldots$.

New function ▪ See also `Highlight`, `ShowGraph` ▪ See page 208

https://doi.org/10.1017/CBO9781139164849.010 Published online by Cambridge University Press