

NAME

Graph

SYNOPSIS

use Graph;

use Graph qw(:all);

DESCRIPTION

Graph class provides the following methods:

new, AddCycle, AddEdge, AddEdges, AddPath, AddVertex, AddVertices, ClearCycles, Copy, CopyEdgesProperties, CopyVerticesAndEdges, CopyVerticesProperties, DeleteCycle, DeleteEdge, DeleteEdgeProperties, DeleteEdgeProperty, DeleteEdges, DeleteEdgesProperties, DeleteEdgesProperty, DeleteGraphProperties, DeleteGraphProperty, DeletePath, DeleteVertex, DeleteVertexProperties, DeleteVertexProperty, DeleteVertices, DeleteVerticesProperty, DetectCycles, GetAllPaths, GetAllPathsStartingAt, GetAllPathsStartingAtWithLengthUpto, GetAllPathsWithLengthUpto, GetCircumference, GetConnectedComponentsVertices, GetCycles, GetCyclesWithEvenSize, GetCyclesWithOddSize, GetCyclesWithSize, GetCyclesWithSizeGreaterThan, GetCyclesWithSizeLessThan, GetDegree, GetEdgeCycles, GetEdgeCyclesWithEvenSize, GetEdgeCyclesWithOddSize, GetEdgeCyclesWithSize, GetEdgeCyclesWithSizeGreaterThan, GetEdgeCyclesWithSizeLessThan, GetEdgeProperty, GetEdges, GetEdgesProperty, GetFusedAndNonFusedCycles, GetGirth, GetGraphProperties, GetGraphProperty, GetIsolatedVertices, GetLargestCycle, GetLargestEdgeCycle, GetLargestVertexCycle, GetLeafVertices, GetMaximumDegree, GetMinimumDegree, GetNeighborhoodVertices, GetNeighborhoodVerticesWithRadiusUpto, GetNeighbors, GetNumOfCycles, GetNumOfCyclesWithEvenSize, GetNumOfCyclesWithOddSize, GetNumOfCyclesWithSize, GetNumOfCyclesWithSizeGreaterThan, GetNumOfCyclesWithSizeLessThan, GetNumOfEdgeCycles, GetNumOfEdgeCyclesWithEvenSize, GetNumOfEdgeCyclesWithOddSize, GetNumOfEdgeCyclesWithSize, GetNumOfEdgeCyclesWithSizeGreaterThan, GetNumOfEdgeCyclesWithSizeLessThan, GetNumOfVertexCycles, GetNumOfVertexCyclesWithEvenSize, GetNumOfVertexCyclesWithOddSize, GetNumOfVertexCyclesWithSize, GetNumOfVertexCyclesWithSizeGreaterThan, GetNumOfVertexCyclesWithSizeLessThan, GetPaths, GetPathsBetween, GetPathsStartingAt, GetPathsStartingAtWithLengthUpto, GetPathsWithLengthUpto, GetSizeOfLargestCycle, GetSizeOfLargestEdgeCycle, GetSizeOfLargestVertexCycle, GetSizeOfSmallestCycle, GetSizeOfSmallestEdgeCycle, GetSizeOfSmallestVertexCycle, GetSmallestCycle, GetSmallestEdgeCycle, GetSmallestVertexCycle, GetTopologicallySortedVertices, GetVertex, GetVertexCycles, GetVertexCyclesWithEvenSize, GetVertexCyclesWithOddSize, GetVertexCyclesWithSize, GetVertexCyclesWithSizeGreaterThan, GetVertexCyclesWithSizeLessThan, GetVertexProperties, GetVertexProperty, GetVertexWithLargestDegree, GetVertexWithSmallestDegree, GetVertices, GetVerticesProperty, GetVerticesWithDegreeLessThan, HasCycle, HasEdge, HasEdgeProperty, HasEdges, HasFusedCycles, HasGraphProperty, HasPath, HasVertex, HasVertexProperty, HasVertices, IsAcyclic, IsAcyclicEdge, IsAcyclicVertex, IsCyclic, IsCyclicEdge, IsCyclicVertex, IsGraph, IsIsolatedVertex, IsLeafVertex, IsUnicyclic, IsUnicyclicEdge, IsUnicyclicVertex, SetActiveCyclicPaths, SetEdgeProperties, SetEdgeProperty, SetEdgesProperty, SetGraphProperties, SetGraphProperty, SetVertexProperties, SetVertexProperty, SetVerticesProperty, StringifyEdgesProperties, StringifyGraph, StringifyGraphProperties, StringifyProperties, StringifyVerticesAndEdges, StringifyVerticesProperties, UpdateEdgeProperty, UpdateVertexProperty

METHODS

new

```
$NewGraph = new Graph([@VertexIDs]);
```

Using specified *Graph VertexIDs*, new method creates a new Graph object and returns newly created Graph object

Examples:

```
$Graph = new Graph();
$Graph = new Graph(@VertexIDs);
```

AddCycle

```
$Graph->AddCycle(@VertexIDs);
```

Adds edges between successive pair of *VertexIDs* including an additional edge from the last to first vertex ID to complete the cycle to *Graph* and returns *Graph*

AddEdge

```
$Graph->AddEdge($VertexID1, $VertexID2);
```

Adds an edge between *VertexID1* and *VertexID2* in a *Graph* and returns *Graph*

AddEdges

```
$Graph->AddEdges(@VertexIDs);
```

Adds edges between successive pair of *VertexIDs* in a *Graph* and returns *Graph*

AddPath

```
$Graph->AddPath(@VertexIDs);
```

Adds edges between successive pair of *VertexIDs* in a *Graph* and returns *Graph*

AddVertex

```
$Graph->AddVertex($VertexID);
```

Adds *VertexID* to *Graph* and returns *Graph*

AddVertices

```
$Graph->AddVertices(@VertexIDs);
```

Adds vertices using *VertexIDs* to *Graph* and returns *Graph*

ClearCycles

```
$Graph->ClearCycles();
```

Delete all cycle properties assigned to graph, vertices, and edges by *DetectCycles* method

Copy

```
$NewGraph = $Graph->Copy();
```

Copies *Graph* and its associated data using `Storable::dclone` and returns a new *Graph* object

CopyEdgesProperties

```
$OtherGraph = $Graph->CopyEdgesProperties($OtherGraph);
```

Copies all properties associated with edges from *Graph* to *\$OtherGraph* and returns *OtherGraph*

CopyVerticesAndEdges

```
$OtherGraph = $Graph->CopyVerticesAndEdges($OtherGraph);
```

Copies all vertices and edges from *Graph* to *\$OtherGraph* and returns *OtherGraph*

CopyVerticesProperties

```
$OtherGraph = $Graph->CopyVerticesProperties($OtherGraph);
```

Copies all properties associated with vertices from *Graph* to *\$OtherGraph* and returns *OtherGraph*

DeleteCycle

```
$Graph->DeleteCycle(@VertexIDs);
```

Deletes edges between successive pair of *VertexIDs* including an additional edge from the last to first vertex ID to complete the cycle to *Graph* and returns *Graph*

DeleteEdge

```
$Graph->DeleteEdge($VertexID1, $VertexID2);
```

Deletes an edge between *VertexID1* and *VertexID2* in a *Graph* and returns *Graph*

DeleteEdgeProperties

```
$Graph->DeleteEdgeProperties($VertexID1, $VertexID2);
```

Deletes all properties associated with edge between *VertexID1* and *VertexID2* in a *Graph* and returns *Graph*

DeleteEdgeProperty

```
$Graph->DeleteEdgeProperty($PropertyName, $VertexID1, $VertexID2);
```

Deletes *PropertyName* associated with edge between *VertexID1* and *VertexID2* in a *Graph* and returns *Graph*

DeleteEdges

```
$Graph->DeleteEdges(@VertexIDs);
```

Deletes edges between successive pair of *VertexIDs* and returns *Graph*

DeleteEdgesProperties

```
$Graph->DeleteEdgesProperties(@VertexIDs);
```

Deletes all properties associated with edges between successive pair of *VertexIDs* and returns *Graph*

DeleteEdgesProperty

```
$Graph->DeleteEdgesProperty($PropertyName, @VertexIDs);
```

Deletes *PropertyName* associated with edges between successive pair of *VertexIDs* and returns *Graph*

DeleteGraphProperties

```
$Graph->DeleteGraphProperties();
```

Deletes all properties associated as graph not including properties associated to vertices or edges and returns *Graph*

DeleteGraphProperty

```
$Graph->DeleteGraphProperty($PropertyName);
```

Deletes a *PropertyName* associated as graph property and returns *Graph*

DeletePath

```
$Graph->DeletePath(@VertexIDs);
```

Deletes edges between successive pair of *VertexIDs* in a *Graph* and returns *Graph*

DeleteVertex

```
$Graph->DeleteVertex($VertexID);
```

Deletes *VertexID* to *Graph* and returns *Graph*

DeleteVertexProperties

```
$Graph->DeleteVertexProperties($VertexID);
```

Deletes all properties associated with *VertexID* and returns *Graph*

DeleteVertexProperty

```
$Graph->DeleteVertexProperty($PropertyName, $VertexID);
```

Deletes a *PropertyName* associated with *VertexID* and returns *Graph*

DeleteVertices

```
$Graph->DeleteVertices(@VertexIDs);
```

Deletes vertices specified in *VertexIDs* and returns *Graph*

DeleteVerticesProperty

```
$Graph->DeleteVerticesProperty($PropertyName, @VertexIDs);
```

Deletes a *PropertyName* associated with *VertexIDs* and returns *Graph*

DetectCycles

```
$Graph->DetectCycles();
```

Detect cycles using CyclesDetection class and associate found cycles to *Graph* object as graph properties: *ActiveCyclicPaths*, *AllCyclicPaths*, *IndependentCyclicPaths*.

Notes:

- o CyclesDetection class detects all cycles in the graph and filters

them to find independent cycles.

- o All cycles related methods in the graph operate on `ActiveCyclicPaths`. By default, active cyclic paths correspond to `IndependentCyclicPaths`. This behavior can be changed using `SetActiveCyclicPaths` method.

GetAllPaths

```
$PathsRef = $Graph->GetAllPaths([$AllowCycles]);
```

Returns a reference to an array containing `Path` objects corresponding to all possible lengths starting from each vertex in graph with sharing of edges in paths traversed. By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle. Duplicate paths are not removed.

GetAllPathsStartingAt

```
@Paths = $Graph->GetAllPathsStartingAt($StartVertexID,
[$AllowCycles]);
```

Returns an array of `Path` objects starting from a `StartVertexID` of any length with sharing of edges in paths traversed. By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

GetAllPathsStartingAtWithLengthUpto

```
@Paths = $Graph->GetAllPathsStartingAtWithLengthUpto($StartVertexID,
$Length, [$AllowCycles]);
```

Returns an array of `Path` objects starting from a `StartVertexID` with length upto a `Length` with sharing of edges in paths traversed. By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

GetAllPathsWithLengthUpto

```
$PathsRef = $Graph->GetAllPathsWithLengthUpto($Length,
[$AllowCycles]);
```

Returns a reference to an array containing `Path` objects corresponding to paths with `Length` starting from each vertex in graph with sharing of edges in paths traversed. By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle. Duplicate paths are not removed.

GetCircumference

```
$Circumference = $Graph->GetCircumference();
```

Returns size of largest cycle in a `Graph`

GetConnectedComponentsVertices

```
@ConnectedComponents = $Graph->GetConnectedComponentsVertices();
```

Returns an array `ConnectedComponents` containing references to arrays with vertex IDs for each component sorted in order of their decreasing size

GetCycles

```
@CyclicPaths = $Graph->GetCycles();
```

Returns an array `CyclicPaths` containing `Path` objects corresponding to cycles in a `Graph`

GetCyclesWithEvenSize

```
@CyclicPaths = $Graph->GetCyclesWithEvenSize();
```

Returns an array `CyclicPaths` containing `Path` objects corresponding to cycles with even size in a `Graph`

GetCyclesWithOddSize

```
@CyclicPaths = $Graph->GetCyclesWithOddSize();
```

Returns an array `CyclicPaths` containing `Path` objects corresponding to cycles with odd size in a `Graph`

GetCyclesWithSize

```
@CyclicPaths = $Graph->GetCyclesWithSize($CycleSize);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with *CycleSize* in a *Graph*

GetCyclesWithSizeGreaterThan

```
@CyclicPaths = $Graph->GetCyclesWithSizeGreaterThan($CycleSize);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with size greater than *CycleSize* in a *Graph*

GetCyclesWithSizeLessThan

```
@CyclicPaths = $Graph->GetCyclesWithSizeGreaterThan($CycleSize);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with size less than *CycleSize* in a *Graph*

GetDegree

```
$Degree = $Graph->GetDegree($VertexID);
```

Returns Degree for *VertexID* in a *Graph* corresponding to sum of in and out vertex degree values

GetEdgeCycles

```
@CyclicPaths = $Graph->GetEdgeCycles($VertexID1, $VertexID2);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to all cycles containing edge between *VertexID1* and *VertexID2* in a *Graph*

GetEdgeCyclesWithEvenSize

```
@CyclicPaths = $Graph->GetEdgeCyclesWithEvenSize($VertexID1,
$VertexID2);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with even size containing edge between *VertexID1* and *VertexID2* in a *Graph*

GetEdgeCyclesWithOddSize

```
@CyclicPaths = $Graph->GetEdgeCyclesWithOddSize($VertexID1,
$VertexID2);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with odd size containing edge between *VertexID1* and *VertexID2* in a *Graph*

GetEdgeCyclesWithSize

```
@CyclicPaths = $Graph->GetEdgeCyclesWithSize($VertexID1, $VertexID2,
$CycleSize);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with size *CycleSize* containing edge between *VertexID1* and *VertexID2* in a *Graph*

GetEdgeCyclesWithSizeGreaterThan

```
@CyclicPaths = $Graph->GetEdgeCyclesWithSizeGreaterThan($VertexID1,
$VertexID2, $CycleSize);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with size greater than *CycleSize* containing edge between *VertexID1* and *VertexID2* in a *Graph*.

GetEdgeCyclesWithSizeLessThan

```
@CyclicPaths = $Graph->GetEdgeCyclesWithSizeLessThan($VertexID1,
$VertexID2, $CycleSize);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with size less than *CycleSize* containing edge between *VertexID1* and *VertexID2*

GetEdgeProperties

```
%EdgeProperties = $Graph->GetEdgeProperties($VertexID1, $VertexID2);
```

Returns a hash EdgeProperties containing all PropertyName and PropertyValue pairs associated with an edge between *VertexID1* and *VertexID2* in a *Graph*

GetEdgeProperty

```
$Value = $Graph->GetEdgeProperty($PropertyName, $VertexID1, $VertexID2);
```

Returns value of *PropertyName* associated with an edge between *VertexID1* and *VertexID2* in a *Graph*

GetEdges

```
@EdgeVertexIDs = $Graph->GetEdges($VertexID);  
$NumOfEdges = $Graph->GetEdges($VertexID);
```

Returns an array *EdgeVertexIDs* with successive pair of IDs corresponding to edges involving *VertexID* or number of edges for *VertexID* in a *Graph*

GetEdgesProperty

```
@PropertyValues = $Graph->GetEdgesProperty($PropertyName, @VertexIDs);
```

Returns an array *PropertyValues* containing property values corresponding to *PropertyName* associated with edges between successive pair of *VertexIDs*

GetFusedAndNonFusedCycles

```
($FusedCycleSetsRef, $NonFusedCyclesRef) =  
$Graph->GetFusedAndNonFusedCycles();
```

Returns references to arrays *FusedCycleSetsRef* and *NonFusedCyclesRef* containing references to arrays of cyclic *Path* objects corresponding to fuses and non-fused cyclic paths

GetGirth

```
$Girth = $Graph->GetGirth();
```

Returns size of smallest cycle in a *Graph*

GetGraphProperties

```
%GraphProperties = $Graph->GetGraphProperties();
```

Returns a hash *EdgeProperties* containing all *PropertyName* and *PropertyValue* pairs associated with graph in a *Graph*

GetGraphProperty

```
$PropertyValue = $Graph->GetGraphProperty($PropertyName);
```

Returns value of *PropertyName* associated with graph in a *Graph*

GetIsolatedVertices

```
@VertexIDs = $Graph->GetIsolatedVertices();
```

Returns an array *VertexIDs* containing vertices without any edges in *Graph*

GetLargestCycle

```
$CyclicPath = $Graph->GetLargestCycle();
```

Returns a cyclic *Path* object corresponding to largest cycle in a *Graph*

GetLargestEdgeCycle

```
$CyclicPath = $Graph->GetLargestEdgeCycle($VertexID1, $VertexID2);
```

Returns a cyclic *Path* object corresponding to largest cycle containing edge between *VertexID1* and *VertexID2* in a *Graph*

GetLargestVertexCycle

```
$CyclicPath = $Graph->GetLargestVertexCycle($VertexID);
```

Returns a cyclic *Path* object corresponding to largest cycle containing *VertexID* in a *Graph*

GetLeafVertices

```
@VertexIDs = $Graph->GetLeafVertices();
```

Returns an array *VertexIDs* containing vertices with degree of 1 in a *Graph*

GetMaximumDegree

```
$Degree = $Graph->GetMaximumDegree();
```

Returns value of maximum vertex degree in a *Graph*.

GetMinimumDegree

```
$Degree = $Graph->GetMinimumDegree();
```

Returns value of minimum vertex degree in a *Graph*

GetNeighborhoodVertices

```
@VertexNeighborhoods = GetNeighborhoodVertices($StartVertexID);
```

Returns an array *VertexNeighborhoods* containing references to arrays corresponding to neighborhood vertices around a specified *StartVertexID* at all possible radii levels

GetNeighborhoodVerticesWithRadiusUpto

```
@VertexNeighborhoods = GetNeighborhoodVerticesWithRadiusUpto(  
    $StartVertexID, $Radius);
```

Returns an array *VertexNeighborhoods* containing references to arrays corresponding to neighborhood vertices around a specified *StartVertexID* upto specified *Radius* levels

GetNeighbors

```
@VertexIDs = $Graph->GetNeighbors($VertexID);  
$NumOfNeighbors = $Graph->GetNeighbors($VertexID);
```

Returns an array *VertexIDs* containing vertices connected to *VertexID* of number of neighbors of a *VertexID* in a *Graph*

GetNumOfCycles

```
$NumOfCycles = $Graph->GetNumOfCycles();
```

Returns number of cycles in a *Graph*

GetNumOfCyclesWithEvenSize

```
$NumOfCycles = $Graph->GetNumOfCyclesWithEvenSize();
```

Returns number of cycles with even size in a *Graph*

GetNumOfCyclesWithOddSize

```
$NumOfCycles = $Graph->GetNumOfCyclesWithOddSize();
```

Returns number of cycles with odd size in a *Graph*

GetNumOfCyclesWithSize

```
$NumOfCycles = $Graph->GetNumOfCyclesWithSize($CycleSize);
```

Returns number of cycles with *CyclesSize* in a *Graph*

GetNumOfCyclesWithSizeGreaterThan

```
$NumOfCycles = $Graph->GetNumOfCyclesWithSizeGreaterThan(  
    $CycleSize);
```

Returns number of cycles with size greater than *CyclesSize* in a *Graph*

GetNumOfCyclesWithSizeLessThan

```
$NumOfCycles = $Graph->GetNumOfCyclesWithSizeLessThan($CycleSize);
```

Returns number of cycles with size less than *CyclesSize* in a *Graph*

GetNumOfEdgeCycles

```
$NumOfCycles = $Graph->GetNumOfEdgeCycles($VertexID1, $VertexID2);
```

Returns number of cycles containing edge between *VertexID1* and *VertexID2* in a *Graph*

GetNumOfEdgeCyclesWithEvenSize

```
$NumOfCycles = $Graph->GetNumOfEdgeCyclesWithEvenSize($VertexID1,  
$VertexID2);
```

Returns number of cycles containing edge between *VertexID1* and *VertexID2* with even size in a *Graph*

GetNumOfEdgeCyclesWithOddSize

```
$NumOfCycles = $Graph->GetNumOfEdgeCyclesWithOddSize($VertexID1,  
$VertexID2);
```

Returns number of cycles containing edge between *VertexID1* and *VertexID2* with odd size in a *Graph*

GetNumOfEdgeCyclesWithSize

```
$NumOfCycles = $Graph->GetNumOfEdgeCyclesWithSize($VertexID1,  
$VertexID2, $CycleSize);
```

Returns number of cycles containing edge between *VertexID1* and *VertexID2* with *CycleSize* size in a *Graph*

GetNumOfEdgeCyclesWithSizeGreaterThan

```
$NumOfCycles = $Graph->GetNumOfEdgeCyclesWithSizeGreaterThan(  
$VertexID1, $VertexID2, $CycleSize);
```

Returns number of cycles containing edge between *VertexID1* and *VertexID2* with size greater than *CycleSize* size in a *Graph*

GetNumOfEdgeCyclesWithSizeLessThan

```
$NumOfCycles = $Graph->GetNumOfEdgeCyclesWithSizeLessThan(  
$VertexID1, $VertexID2, $CycleSize);
```

Returns number of cycles containing edge between *VertexID1* and *VertexID2* with size less than *CycleSize* size in a *Graph*

GetNumOfVertexCycles

```
$NumOfCycles = $Graph->GetNumOfVertexCycles($VertexID);
```

Returns number of cycles containing *VertexID* in a *Graph*

GetNumOfVertexCyclesWithEvenSize

```
$NumOfCycles = $Graph->GetNumOfVertexCyclesWithEvenSize($VertexID);
```

Returns number of cycles containing *VertexID* with even size in a *Graph*.

GetNumOfVertexCyclesWithEvenSize

```
$NumOfCycles = $Graph->GetNumOfVertexCyclesWithEvenSize($VertexID);
```

Returns number of cycles containing *VertexID* with even size in a *Graph*

GetNumOfVertexCyclesWithOddSize

```
$NumOfCycles = $Graph->GetNumOfVertexCyclesWithOddSize($VertexID);
```

Returns number of cycles containing *VertexID* with odd size in a *Graph*.

GetNumOfVertexCyclesWithSize

```
$NumOfCycles = $Graph->GetNumOfVertexCyclesWithSize($VertexID);
```

Returns number of cycles containing *VertexID* with even size in a *Graph*

GetNumOfVertexCyclesWithSizeGreaterThan

```
$NumOfCycles = $Graph->GetNumOfVertexCyclesWithSizeGreaterThan(  
$VertexID, $CycleSize);
```

Returns number of cycles containing *VertexID* with size greater than *CycleSize* in a *Graph*

GetNumOfVertexCyclesWithSizeLessThan

```
$NumOfCycles = $Graph->GetNumOfVertexCyclesWithSizeLessThan(  
$VertexID, $CycleSize);
```

Returns number of cycles containing *VertexID* with size less than *CycleSize* in a *Graph*

GetPaths

```
$PathsRefs = $Graph->GetPaths([$AllowCycles]);
```

Returns a reference to an array of *Path* objects corresponding to paths of all possible lengths starting from each vertex with no sharing of edges in paths traversed. By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

GetPathsBetween

```
@Paths = $Graph->GetPathsBetween($StartVertexID, $EndVertexID);
```

Returns an arrays of *Path* objects list of paths between *StartVertexID* and *EndVertexID*. For cyclic graphs, the list contains may contain more than one *Path* object.

GetPathsStartingAt

```
@Paths = $Graph->GetPathsStartingAt($StartVertexID, [$AllowCycles]);
```

Returns an array of *Path* objects corresponding to all possible lengths starting from a specified *StartVertexID* with no sharing of edges in paths traversed. By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

GetPathsStartingAtWithLengthUpto

```
@Paths = $Graph->StartingAtWithLengthUpto($StartVertexID, $Length,  
    $AllowCycles);
```

Returns an array of *Path* objects corresponding to all paths starting from a specified *StartVertexID* with length upto *Length* and no sharing of edges in paths traversed. By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

GetPathsWithLengthUpto

```
@Paths = $Graph->GetPathsWithLengthUpto($Length, $AllowCycles);
```

Returns an array of *Path* objects corresponding to to paths starting from each vertex in graph with length upto specific *Length* and no sharing of edges in paths traversed. By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

GetSizeOfLargestCycle

```
$Size = $Graph->GetSizeOfLargestCycle();
```

Returns size of the largest cycle in a *Graph*

GetSizeOfLargestEdgeCycle

```
$Size = $Graph->GetSizeOfLargestEdgeCycle($VertexID1, $VertexID2);
```

Returns size of the largest cycle containing edge between *VertexID1* and *VertexID2* in a *Graph*

GetSizeOfLargestVertexCycle

```
$Size = $Graph->GetSizeOfLargestVertexCycle($VertexID);
```

Returns size of the largest cycle containing *VertexID* in a *Graph*

GetSizeOfSmallestCycle

```
$Size = $Graph->GetSizeOfSmallestCycle();
```

Returns size of the smallest cycle in a *Graph*

GetSizeOfSmallestEdgeCycle

```
$Size = $Graph->GetSizeOfSmallestEdgeCycle($VertexID1, $VertexID2);
```

Returns size of the smallest cycle containing edge between *VertexID1* and *VertexID2* in a *Graph*

GetSizeOfSmallestVertexCycle

```
$Size = $Graph->GetSizeOfSmallestVertexCycle($VertexID);
```

Returns size of the smallest cycle containing *VertexID* in a *Graph*

GetSmallestCycle

```
$CyclicPath = $Graph->GetSmallestCycle();
```

Returns a cyclic *Path* object corresponding to smallest cycle in a *Graph*

GetSmallestEdgeCycle

```
$CyclicPath = $Graph->GetSmallestEdgeCycle($VertexID1, $VertexID2);
```

Returns a cyclic *Path* object corresponding to smallest cycle containing edge between *VertexID1* and *VertexID2* in a *Graph*

GetSmallestVertexCycle

```
$CyclicPath = $Graph->GetSmallestVertexCycle($VertexID);
```

Returns a cyclic *Path* object corresponding to smallest cycle containing *VertexID* in a *Graph*

GetTopologicallySortedVertices

```
@VertexIDs = $Graph->GetTopologicallySortedVertices(  
    [$RootVertexID]);
```

Returns an array of *VertexIDs* sorted topologically starting from a specified *RootVertexID* or from an arbitrary vertex ID

GetVertex

```
$VertexValue = $Graph->GetVertex($VertexID);
```

Returns vertex value for *VertexID* in a *Graph*. Vertex IDs and values are equivalent in the current implementation of Graph

GetVertexCycles

```
@CyclicPaths = $Graph->GetVertexCycles($VertexID);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to all cycles containing *VertexID* in a *Graph*

GetVertexCyclesWithEvenSize

```
@CyclicPaths = $Graph->GetVertexCyclesWithEvenSize($VertexID);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with even size containing *VertexID* in a *Graph*

GetVertexCyclesWithOddSize

```
@CyclicPaths = $Graph->GetVertexCyclesWithOddSize($VertexID);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with odd size containing *VertexID* in a *Graph*

GetVertexCyclesWithSize

```
@CyclicPaths = $Graph->GetVertexCyclesWithSize($VertexID,  
    $CycleSize);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with size *CycleSize* containing *VertexID* in a *Graph*

GetVertexCyclesWithSizeGreaterThan

```
@CyclicPaths = $Graph->GetVertexCyclesWithSizeGreaterThan($VertexID,  
    $CycleSize);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with size greater than *CycleSize* containing *VertexID* in a *Graph*

GetVertexCyclesWithSizeLessThan

```
@CyclicPaths = $Graph->GetVertexCyclesWithSizeLessThan($VertexID,  
    $CycleSize);
```

Returns an array *CyclicPaths* containing *Path* objects corresponding to cycles with size less than

CycleSize containing *VertexID* in a *Graph*

GetVertexProperties

```
%VertexProperties = $Graph->GetVertexProperties($VertexID);
```

Returns a hash *VertexProperties* containing all *PropertyName* and *PropertyValue* pairs associated with a *VertexID* in a *Graph*

GetVertexProperty

```
$Value = $Graph->GetVertexProperty($PropertyName, $VertexID);
```

Returns value of *PropertyName* associated with a *VertexID* in a *Graph*

GetVertexWithLargestDegree

```
$VertexID = $Graph->GetVertexWithLargestDegree();
```

Returns *VertexID* with largest degree in a *Graph*

GetVertexWithSmallestDegree

```
$VertexID = $Graph->GetVertexWithSmallestDegree();
```

Returns *VertexID* with smallest degree in a *Graph*

GetVertices

```
@VertexIDs = $Graph->GetVertices();
$VertexCount = $Graph->GetVertices();
```

Returns an array of *VertexIDs* corresponding to all vertices in a *Graph*; in a scalar context, number of vertices is returned.

GetVerticesProperty

```
@PropertyValues = $Graph->GetVerticesProperty($PropertyName, @VertexIDs);
```

Returns an array *PropertyValues* containing property values corresponding to *PropertyName* associated with *VertexIDs* in a *Graph*

GetVerticesWithDegreeLessThan

```
@VertexIDs = $Graph->GetVerticesWithDegreeLessThan($Degree);
```

Returns an array of *VertexIDs* containing vertices with degree less than *Degree* in a *Graph*

HasCycle

```
$Status = $Graph->HasCycle(@VertexIDs);
```

Returns 1 or 0 based on whether edges between successive pair of *VertexIDs* including an additional edge from the last to first vertex ID exists in a *Graph*

HasEdge

```
$Status = $Graph->HasEdge($VertexID1, $VertexID2);
```

Returns 1 or 0 based on whether an edge between *VertexID1* and *VertexID2* exist in a *Graph*

HasEdgeProperty

```
$Status = $Graph->HasEdgeProperty($PropertyName, $VertexID1,
    $VertexID2);
```

Returns 1 or 0 based on whether *PropertyName* has already been associated with an edge between *VertexID1* and *VertexID2* in a *Graph*

HasEdges

```
@EdgesStatus = $Graph->HasEdges(@VertexIDs);
$FoundEdgesCount = $Graph->HasEdges(@VertexIDs);
```

Returns an array *EdgesStatus* containing 1s and 0s corresponding to whether edges between successive pairs of *VertexIDs* exist in a *Graph*. In a scalar context, number of edges found is returned.

HasFusedCycles

```
$Status = $Graph->HasFusedCycles();
```

Returns 1 or 0 based on whether any fused cycles exist in a *Graph*

HasGraphProperty

```
$Status = $Graph->HasGraphProperty($PropertyName);
```

Returns 1 or 0 based on whether *PropertyName* has already been associated as a graph property as opposed to vertex or edge property in a *Graph*

HasPath

```
$Status = $Graph->HasPath(@VertexIDs);
```

Returns 1 or 0 based on whether edges between all successive pairs of *VertexIDs* exist in a *Graph*

HasVertex

```
$Status = $Graph->HasVertex($VertexID);
```

Returns 1 or 0 based on whether *VertexID* exists in a *Graph*

HasVertexProperty

```
$Status = $Graph->HasGraphProperty($HasVertexProperty, $VertexID);
```

Returns 1 or 0 based on whether *PropertyName* has already been associated with *VertexID* in a *Graph*

HasVertices

```
@VerticesStatus = $Graph->HasVertices(@VertexIDs);  
$VerticesFoundCount = $Graph->HasVertices(@VertexIDs);
```

Returns an array containing 1s and 0s corresponding to whether *VertexIDs* exist in a *Graph*. In a scalar context, number of vertices found is returned.

IsAcyclic

```
$Status = $Graph->IsAcyclic();
```

Returns 0 or 1 based on whether a cycle exist in a *Graph*

IsAcyclicEdge

```
$Status = $Graph->IsAcyclicEdge($VertexID1, $VertexID2);
```

Returns 0 or 1 based on whether a cycle containing an edge between *VertexID1* and *VertexID2* exists in a *Graph*

IsAcyclicVertex

```
$Status = $Graph->IsAcyclicVertex($VertexID1);
```

Returns 0 or 1 based on whether a cycle containing a *VertexID* exists in a *Graph*

IsCyclic

```
$Status = $Graph->IsCyclic();
```

Returns 1 or 0 based on whether a cycle exist in a *Graph*

IsCyclicEdge

```
$Status = $Graph->IsCyclicEdge($VertexID1, $VertexID2);
```

Returns 1 or 0 based on whether a cycle containing an edge between *VertexID1* and *VertexID2* exists in a *Graph*

IsCyclicVertex

```
$Status = $Graph->IsCyclicVertex($VertexID1);
```

Returns 1 or 0 based on whether a cycle containing a *VertexID* exists in a *Graph*

IsGraph

```
$Status = Graph::IsGraph($Object);
```

Returns 1 or 0 based on whether *Object* is a Graph object.

IsIsolatedVertex

```
$Status = $Graph->IsIsolatedVertex($VertexID);
```

Returns 1 or 0 based on whether *VertexID* is an isolated vertex in a *Graph*. A vertex with zero as its degree value is considered an isolated vertex

IsLeafVertex

```
$Status = $Graph->IsLeafVertex($VertexID);
```

Returns 1 or 0 based on whether *VertexID* is an isolated vertex in a *Graph*. A vertex with one as its degree value is considered an isolated vertex

IsUnicyclic

```
$Status = $Graph->IsUnicyclic();
```

Returns 1 or 0 based on whether only one cycle is present in a *Graph*

IsUnicyclicEdge

```
$Status = $Graph->IsUnicyclicEdge($VertexID1, $VertexID2);
```

Returns 1 or 0 based on whether only one cycle contains the edge between *VertexID1* and *VertexID2* in a *Graph*

IsUnicyclicVertex

```
$Status = $Graph->IsUnicyclicVertex($VertexID);
```

Returns 1 or 0 based on whether only one cycle contains *VertexID* in a *Graph*

SetActiveCyclicPaths

```
$Graph->SetActiveCyclicPaths($CyclicPathsType);
```

Sets the type of cyclic paths to use during all methods related to cycles and returns *Graph*. Possible values for cyclic paths: *Independent* or *All*.

SetEdgeProperties

```
$Graph->SetEdgeProperties($VertexID1, $VertexID2, @NamesAndValues);
```

Associates property names and values corresponding to successive pairs of values in *NamesAndValues* to an edge between *VertexID1* and *VertexID2* in a *Graph* and returns *Graph*

SetEdgeProperty

```
$Graph->SetEdgeProperty($Name, $Value, $VertexID1, $VertexID2);
```

Associates property *Name* and *Value* to an edge between *VertexID1* and *VertexID2* in a *Graph* and returns *Graph*

SetEdgesProperty

```
$Graph->SetEdgesProperty($Name, @ValuesAndVertexIDs);
```

Associates a same property *Name* but different *Values* for different edges specified using triplets of *PropertyValue*, *VertexID1*, *VertexID2* via *ValuesAndVertexIDs* in a *graph*

SetGraphProperties

```
$Graph->SetGraphProperties(%NamesAndValues);
```

Associates property names and values *NamesAndValues* hash to graph as opposed to vertex or edge and returns *Graph*

SetGraphProperty

```
$Graph->SetGraphProperty($Name, $Value);
```

Associates property *Name* and *Value* to graph as opposed to vertex or edge and returns *Graph*

SetVertexProperties

```
$Graph->SetVertexProperties($VertexID, @NamesAndValues);
```

Associates property names and values corresponding to successive pairs of values in *NamesAndValues* to *VertexID* in a *Graph* and returns *Graph*

SetVertexProperty

```
$Graph->SetVertexProperty($Name, $Value, $VertexID);
```

Associates property *Name* and *Value* to *VertexID* in a *Graph* and returns *Graph*

SetVerticesProperty

```
$Graph->SetVerticesProperty($Name, @ValuesAndVertexIDs);
```

Associates a same property *Name* but different *Values* for different vertices specified using doublets of *PropertyValue*, *\$VertexID* via *ValuesAndVertexIDs* in a *graph*

StringifyEdgesProperties

```
$String = $Graph->StringifyEdgesProperties();
```

Returns a string containing information about properties associated with all edges in a *Graph* object

StringifyGraph

```
$String = $Graph->StringifyGraph();
```

Returns a string containing information about *Graph* object

StringifyGraphProperties

```
$String = $Graph->StringifyGraphProperties();
```

Returns a string containing information about properties associated with graph as opposed to vertex or an edge in a *Graph* object

StringifyProperties

```
$String = $Graph->StringifyProperties();
```

Returns a string containing information about properties associated with graph, vertices, and edges in a *Graph* object

StringifyVerticesAndEdges

```
$String = $Graph->StringifyVerticesAndEdges();
```

Returns a string containing information about vertices and edges in a *Graph* object

StringifyVerticesProperties

```
$String = $Graph->StringifyVerticesProperties();
```

Returns a string containing information about properties associated with vertices a *Graph* object

UpdateEdgeProperty

```
$Graph->UpdateEdgeProperty($Name, $Value, $VertexID1, $VertexID2);
```

Updates property *Value* for *Name* associated with an edge between *VertexID1* and *VertexID1* and returns *Graph*

UpdateVertexProperty

```
$Graph->UpdateVertexProperty($Name, $Value, $VertexID);
```

Updates property *Value* for *Name* associated with *VertexID* and returns *Graph*

AUTHOR

Manish Sud <msud@san.rr.com>

SEE ALSO

CyclesDetection.pm, Path.pm, PathGraph.pm, PathsTraversal.pm

COPYRIGHT

Copyright (C) 2004-2008 Manish Sud. All rights reserved.

This file is part of MayaChemTools.

MayaChemTools is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.