NAME

PathsTraversal

SYNOPSIS

use Graph::PathsTraversal;

use Graph::PathsTraversal qw(:all);

DESCRIPTION

PathsTraversal class provides the following methods:

new, Copy, GetConnectedComponentsVertices, GetPaths, GetVertices, GetVerticesDepth, GetVerticesNeighborhoods, GetVerticesNeighborhoodsWithSuccessors, GetVerticesPredecessors, GetVerticesRoots, PerformAllPathsSearch, PerformAllPathsSearchWithLength, PerformAllPathsSearchWithLengthUpto, PerformBreadthFirstSearch, PerformBreadthFirstSearchWithLimit, PerformDepthFirstSearch, PerformDepthFirstSearchWithLimit, PerformNeighborhoodVerticesSearch, PerformNeighborhoodVerticesSearchWithRadiusUpto, PerformNeighborhoodVerticesSearchWithSuccessors, PerformNeighborhoodVerticesSearchWithSuccessorsAndRadiusUpto, PerformPathsSearch, PerformPathsSearchBetween, PerformPathsSearchWithLength, PerformPathsSearchWithLengthUpto, StringifyPaths, StringifyPathsTraversal, StringifyVerticesDepth, StringifyVerticesNeighborhoods, StringifyVerticesNeighborhoodsWithSuccessors, StringifyVerticesPredecessors, StringifyVerticesRoots, StringifyVerticesSuccessors

METHODS

new

```
$PathsTraversal = new Graph::PathsTraversal($Graph);
```

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

Copy

```
$PathsTraversal = $PathsTraversal->Copy();
```

Copies *PathsTraversal* and its associated data using Storable::dclone and returns a new PathsTraversal object.

GetConnectedComponentsVertices

```
@Components = $PathsTraversal->GetConnectedComponentsVertices();
$NumOfComponents = $PathsTraversal->GetConnectedComponentsVertices();
```

Returns an array of Components containing references to arrays of vertex IDs corresponding to connected components of graph after a search. In scalar context, the number of connected components is returned.

Connected Components is sorted in descending order of number of vertices in each connected component.

GetPaths

```
@Paths = $PathsTraversal->GetPaths();
$NumOfPaths = $PathsTraversal->GetPaths();
```

Returns an array of Paths containing references to arrays of vertex IDs corresponding to to paths traversed in a graph after a search. In scalar context, number of paths is returned.

Paths array is sorted in ascending order of path lengths.

GetVertices

```
@Vertices = $PathsTraversal->GetVertices();
$NumOfVertices = $PathsTraversal->GetVertices();
```

Returns an array containing an ordered list of vertex IDs traversed during a search. In scalar context, the number of vertices is returned.

GetVerticesDepth

```
%VerticesDepth = $PathsTraversal->GetVerticesDepth();
```

Returns a hash *VerticesDepth* containing vertex ID and depth from root vertex as a key and value pair for all vertices traversed during a search.

GetVerticesNeighborhoods

```
@VerticesNeighborhoods =
   $PathsTraversal->GetVerticesNeighborhoods();
$NumOfVerticesNeighborhoods =
  $PathsTraversal->GetVerticesNeighborhoods();
```

Returns an array *VerticesNeighborhoods* containing references to arrays corresponding to vertices collected at various neighborhood radii around a specified vertex during a vertex neighborhood search. In scalar context, the number of neighborhoods is returned.

GetVerticesNeighborhoodsWithSuccessors

```
@VerticesNeighborhoodsWithSucceessors =
   $PathsTraversal->GetVerticesNeighborhoodsWithSuccessors();
$NumOfVerticesNeighborhoodsWithSucceessors =
  $PathsTraversal->GetVerticesNeighborhoodsWithSuccessors();
```

Returns an array *VerticesNeighborhoodsWithSucceessors* containing references to arrays with first value corresponding to vertex IDs corresponding to a vertex at a specific neighborhood radius level and second value a reference to an arraty containing its successors.

GetVerticesPredecessors

```
%VerticesPredecessors = $PathsTraversal->GetVerticesPredecessors();
```

Returns a hash *VerticesPredecessors* containing vertex ID and predecessor vertex ID as key and value pair for all vertices traversed during a search.

GetVerticesRoots

```
%VerticesRoots = $PathsTraversal->GetVerticesRoots();
```

Returns a hash *VerticesPredecessors* containing vertex ID and root vertex ID as a key and value pair for all vertices traversed during a search.

PerformAllPathsSearch

```
$PathsTraversal->PerformAllPathsSearch($StartVertexID, [$AllowCycles]);
```

Searches all paths starting from a *StartVertexID* with sharing of edges in paths traversed and returns *PathsTraversal*.

By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

PerformAllPathsSearchWithLength

```
$PathsTraversal->PerformAllPathsSearchWithLength($StartVertexID,
                 $Length, [$AllowCycles]);
```

Searches all paths starting from *StartVertexID* of specific *Length* with sharing of edges in paths traversed and returns *PathsTraversal*.

By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

PerformAllPathsSearchWithLengthUpto

```
$PathsTraversal->PerformAllPathsSearchWithLengthUpto($StartVertexID,
                 $Length, [$AllowCycles]);
```

Searches all paths starting from *StartVertexID* of length upto a *Length* with sharing of edges in paths traversed and returns *PathsTraversal*.

By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the

PerformBreadthFirstSearch

```
$PathsTraversal->PerformBreadthFirstSearch();
```

Performs Breadth First Search (BFS) and returns *PathsTraversal*.

PerformBreadthFirstSearchWithLimit

```
$PathsTraversal->PerformBreadthFirstSearchWithLimit($DepthLimit,
                [$RootVertexID]);
```

Performs BFS with depth up to *DepthLimit* starting at *RootVertexID* and returns *PathsTraversal*. By default, root vertex ID corresponds to an arbitrary vertex.

PerformDepthFirstSearch

```
$Return = $PathsTraversal->PerformDepthFirstSearch();
```

Performs Depth First Search (DFS) and returns *PathsTraversal*.

PerformDepthFirstSearchWithLimit

```
$PathsTraversal->PerformDepthFirstSearchWithLimit($DepthLimit,
                [$RootVertexID]);
```

Performs DFS with depth up to *DepthLimit* starting at *RootVertexID* and returns *PathsTraversal*. By default, root vertex ID corresponds to an arbitrary vertex.

PerformNeighborhoodVerticesSearch

```
$PathsTraversal->PerformNeighborhoodVerticesSearch($StartVertexID);
```

Searches vertices around *StartVertexID* at all neighborhood radii and returns *PathsTraversal* object.

PerformNeighborhoodVerticesSearchWithRadiusUpto

```
$PathsTraversal->PerformNeighborhoodVerticesSearchWithRadiusUpto(
                $StartVertexID, $Radius);
```

Searches vertices around *StartVertexID* with neighborhood radius up to *Radius* and returns *PathsTraversal* object.

PerformNeighborhoodVerticesSearchWithSuccessors

```
$PathsTraversal->PerformNeighborhoodVerticesSearchWithSuccessors(
                $StartVertexID);
```

Searches vertices around *StartVertexID* at all neighborhood radii along with identification of successor vertices for each vertex found during the traversal and returns *PathsTraversal*.

PerformNeighborhoodVerticesSearchWithSuccessorsAndRadiusUpto

```
$PathsTraversal->
            PerformNeighborhoodVerticesSearchWithSuccessorsAndRadiusUpto(
                $StartVertexID, $Radius);
```

Searches vertices around *StartVertexID* with neighborhood radius upto *Radius* along with identification of successor vertices for each vertex found during the traversal and returns *PathsTraversal*.

PerformPathsSearch

```
$PathsTraversal->PerformPathsSearch($StartVertexID, [$AllowCycles]);
```

Searches paths starting from *StartVertexID* with no sharing of edges in paths traversed and returns *PathsTraversal*.

By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

PerformPathsSearchBetween

```
$PathsTraversal->PerformPathsSearchBetween($StartVertexID, $EndVertexID);
```

Searches paths between *StartVertexID* and *EndVertexID* and returns *PathsTraversal*

PerformPathsSearchWithLength

```
$PathsTraversal->PerformPathsSearchWithLength($StartVertexID, $Length,
                [$AllowCycles]);
```

Searches paths starting from *StartVertexID* with length *Length* with no sharing of edges in paths traversed and returns *PathsTraversal*.

By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

PerformPathsSearchWithLengthUpto

```
$PathsTraversal->PerformPathsSearchWithLengthUpto($StartVertexID, $Length,
                [$AllowCycles]);
```

Searches paths starting from *StartVertexID* with length upto *Length* with no sharing of edges in paths traversed and returns *PathsTraversal*.

By default, cycles are included in paths. A path containing a cycle is terminated at a vertex completing the cycle.

StringifyPaths

```
$String = $PathsTraversal->StringifyPaths();
```

Returns a string containing information about traversed paths in *PathsTraversal* object

StringifyPathsTraversal

```
$String = $PathsTraversal->StringifyPathsTraversal();
```

Returns a string containing information about *PathsTraversal* object.

StringifyVerticesDepth

```
$String = $PathsTraversal->StringifyVerticesDepth();
```

Returns a string containing information about depth of vertices found during search by *PathsTraversal* object.

StringifyVerticesNeighborhoods

```
$String = $PathsTraversal->StringifyVerticesNeighborhoods();
```

Returns a string containing information about neighborhoods of vertices found during search by *PathsTraversal* object.

StringifyVerticesNeighborhoodsWithSuccessors

```
$String = $PathsTraversal->StringifyVerticesNeighborhoodsWithSuccessors();
```

Returns a string containing information about neighborhoods of vertices along with their successors found during search by *PathsTraversal* object.

StringifyVerticesPredecessors

```
$String = $PathsTraversal->StringifyVerticesPredecessors();
```

Returns a string containing information about predecessors of vertices found during search by *PathsTraversal* object.

StringifyVerticesRoots

```
$String = $PathsTraversal->StringifyVerticesRoots();
```

Returns a string containing information about roots of vertices found during search by *PathsTraversal* object.

StringifyVerticesSuccessors

```
$String = $PathsTraversal->StringifyVerticesSuccessors();
```

Returns a string containing information about successors of vertices found during search by *PathsTraversal* object.

AUTHOR

Manish Sud <msud@san.rr.com>

SEE ALSO

Graph.pm, Path.pm

COPYRIGHT