NAME

Path - Path class

SYNOPSIS

```
use Graph::Path;

use Graph::Path qw(:all);
```

DESCRIPTION

Path class provides the following methods:

new, AddVertex, AddVertices, Copy, GetCommonVertices, GetEdges, GetEndVertex, GetLength, GetStartVertex, GetTerminalVertices, GetVertex, GetVertices, IsCycle, IsIndependentCyclicPath, IsIndependentPath, IsPath, Join, JoinAtVertex, PopVertex, PushVertex, PushVertices, Reverse, ShiftVertex, StringifyPath, UnshiftVertex, UnshiftVertices

Path is a sequential list of vertices with an edge between two successive vertices. The path becomes a cycle when start vertex and end vertex are the same.

The following operators are overloaded:

```
"" == eq
```

METHODS

new

```
$NewPath = new Path();
$NewPath = new Path(@VertexIDs);
```

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

AddVertex

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

Adds *VertexID* to *Path* and returns *Path*.

AddVertices

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

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

Copy

```
$Return = $Path->Copy();
```

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

GetCommonVertices

```
@CommonVertices = $Path->GetCommonVertices($OtherPath);
$NumOfCommonVertices = $Path->GetCommonVertices($OtherPath);
```

Returns an array containing common vertex IDs between two paths. In scalar context, number of common vertices is returned.

GetEdges

```
@EdgesVertexIDs = $Path->GetEdges();
$NumOfEdges = $Path->GetEdges();
```

Returns an array containg successive paris of vertex IDs corresponding to all edges in *Path*. In scalar context, the number of edges is returned.

GetEndVertex

```
$VertexID = $Path->GetEndVertex();
```

Returns VertexID of end vertex in *Path*.

GetLength

```
$Length = $Path->GetLength();
```

Returns Length of *Path* corresponding to number of vertices in *Path*.

GetStartVertex

```
$VertexID = $Path->GetStartVertex();
```

Returns VertexID of start vertex in *Path*.

GetTerminalVertices

```
($StartVertexID, $EndVertexID) = $Path->GetTerminalVertices();
```

Returns vertex IDs of start and end vertices in *Path*.

GetVertex

```
$VertexID = $Path->GetVertex($Index);
```

Returns specific vertex ID from *Path* corresponding to *Index* with indicies starting from 0.

GetVertices

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

Returns an array containing all vertex IDs in *Path*. In scalar context, number of vertices is returned.

IsCycle

```
$Status = $Path->IsCycle();
```

Returns 1 or 0 based on whether *Path* is a CyclicPath which has the same start and end vertex IDs.

IsIndependentCyclicPath

```
$Status = $Path->IsIndependentCyclicPath();
```

Returns 1 or 0 based on whether *Path* is an independent CyclicPath. For a *Path* to be an independent cyclic path, it must be a cyclic path and have unique vertices.

IsIndependentPath

```
$Status = $Path->IsIndependentPath();
```

Returns 1 or 0 based on whether *Path* is an independent Path. For a *Path* to be an independent path, it must have unique vertices.

IsPath

```
$Status = Graph::Path::IsPath();
```

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

Join

```
$NewPath = $Path->Join($OtherPath);
$NewPath = $Path->Join(@VertexIDs);
```

Joins existing *Path* with a new path specified as a *OtherPath* object or an array of *VertexIDs* and returns *NewPath*.

In order to successfully join two paths, terminal vertices must have a common vertex. Based on the common terminal vertex found, additional path vertices are added to the current *Path* in one of the following four ways:

```
. EndVertex = NewStartVertex: New path at end of current path with
  same vertices order

. EndVertex = NewEndVertex: New path at end of current path with
  reversed vertices order

. StartVertex = NewEndVertex: New path at front of current path
  with same vertices order

. StartVertex = NewStartVertex: New path at front of current path
  with reversed vertices order
```

JoinAtVertex

```
$NewPath = $Path->JoinAtVertex($OtherPath, $CenterVertexID);
```

Joins existing *Path* with *OtherPath* at a specified *CeterVertexID* and returns a *NewPath*.

PopVertex

```
$Path->PopVertex();
```

Removes end vertex from *Path* and returns *Path*.

PushVertex

```
$Path->PushVertex($VertexID);
```

Adds *VertexID* to *Path* after end vertex and returns *Path*.

PushVertices

```
$Path->PushVertices(@VertexIDs);
```

Adds *VertexIDs* to *Path* after end vertex and returns *Path*.

Reverse

```
$Path->Reverse();
```

Reverses order of vertices in *Path* and returns *Path*.

ShiftVertex

```
$Path->ShiftVertex();
```

Removes start vertex from *Path* and returns *Path*.

StringifyPath

```
$String = $Path->StringifyPath();
```

Returns a string containing information about *Path* object.

UnshiftVertex

```
$Path->UnshiftVertex($VertexID);
```

Adds *VertexID* to *Path* before start vertex and returns *Path*.

UnshiftVertices

```
$Path->UnshiftVertices(@VertexIDs);
```

Adds *VertexIDs* to *Path* before start vertex and returns *Path*.

AUTHOR

Manish Sud <msud@san.rr.com>

SEE ALSO

PathGraph.pm, PathsTraversal.pm

COPYRIGHT