PathFinder

Generated by Doxygen 1.8.6

Sun Jun 18 2017 14:30:45

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 AdjacencyList Class Reference

```
#include <AdjacencyList.h>
```

Public Member Functions

- AdjacencyList ()
- AdjacencyList (int size)
- AdjacencyList (int size, int vertexCount)
- void resizeAdjList (int size, int vertexCount)
- void removeEdge (int firstVertex, int secondVertex)
- void addEdge (int firstVertex, int secondVertex)
- void updateInfo (int vertexToUpdate, int value)
- int getInfoValue (int vertex) const
- void setVertexConnections (int vertex, std::vector< int > &vericesToAdd)
- std::vector< int > getVertexConnections (int vertex) const
- int getVertexConnectionCount (int vertex) const
- int getVertexData (int vertex, int index) const
- bool validVertex (int vertex) const
- int size () const
- int vertexCount () const
- std::string displayAdjacencyList () const
- void saveToFile (bool useNameList, std::map< int, std::string > nameList, std::string path)

2.1.1 Detailed Description

This class creates an array of linked lists to store data. The functions contained in it allow you to manipulate this array.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 AdjacencyList::AdjacencyList ( )

Default constructor that creates an adjacency list.

2.1.2.2 AdjacencyList::AdjacencyList (int *size*)

Constructor that creates an adjacency list.

Parameters

size	Determines the size of the adjacency list

2.1.2.3 AdjacencyList::AdjacencyList (int *size*, int *vertexCount*)

Constructor that creates an adjacency list. The parameter size may be higher than vertexCount since not all vertices entered are not required to be in numerical order.

Parameters

size	Determines the size of the adjacency list
vertexCount	The number of vertices in the outer vector list

2.1.3 Member Function Documentation

2.1.3.1 void AdjacencyList::addEdge (int *firstVertex*, int *secondVertex*)

Adds an edge to the adjacencyList.

Parameters

firstVertex	The first half of the edge.
secondVertex	The second half of the edge.

2.1.3.2 std::string AdjacencyList::displayAdjacencyList ( ) const

Gets a string containing the information in the adjacencyList.

Returns

a string that contains the information in the adjacencyList.

2.1.3.3 int AdjacencyList::getInfoValue (int *vertex*) const

Gets the value of a given vertex

Parameters

vertex	The vertex that you want to look up.

Returns

an integer that contains the value of a vertex.

2.1.3.4 int AdjacencyList::getVertexConnectionCount (int *vertex*) const

Gets the number of edges a given vertex has.

Parameters

vertex	The vertex that you want to look up.

Returns

an integer that contains the number of connections for the given vertex.

2.1.3.5 std::vector< int > AdjacencyList::getVertexConnections (int *vertex*) const

Gets a vector of vertices at a given vertex in the adjacency list.

Parameters

vertex	The vertex that you want to look up.

Returns

a vector<int> that contains the edges connected to a given vertex.

2.1.3.6 int AdjacencyList::getVertexData (int *vertex,* int *index*) const

Gets the value at the given vertex and index in the adjacencyList.

Parameters

vertex	The vertex in the adjacencyList to look at.
index	The position to get the value from.

Returns

an integer which is the value at the given vertex and index.

2.1.3.7 void AdjacencyList::removeEdge (int *firstVertex,* int *secondVertex*)

Removes an edge from the adjacencyList.

Parameters

firstVertex	The first half of the edge.
secondVertex	The second half of the edge.

2.1.3.8 void AdjacencyList::resizeAdjList (int *size,* int *vertexCount*)

Constructor that creates an adjacency list. The parameter size may be higher than vertexCount since not all vertices entered are not required to be in numerical order.

Parameters

size	Determines the size of the adjacency list
vertexCount	The number of vertices in the outer vector list

2.1.3.9 void AdjacencyList::saveToFile (bool *useNameList,* std::map< int, std::string > *nameList,* std::string *path*)

Saves the adjacencyList to a file.

Parameters

useNameList	Determines whether or not to save the information with name or integer values.
nameList	The map of integers to names to use if useNameList is True.
path	The path to save the file too. The path must include the file name.

2.1.3.10 void AdjacencyList::setVertexConnections (int *vertex,* std::vector< int > & *vericesToAdd*)

Adds all the vertices in verticesToAdd to the adjacencyList at vertex

Parameters

vertex	The vertex add connections to.
vericesToAdd	Vector of vertices to add.

2.1.3.11 int AdjacencyList::size ( ) const

Gets the size of the adjacencyList.

Returns

an integer that contains the size of the adjacencyList.

2.1.3.12 void AdjacencyList::updateInfo (int *vertexToUpdate,* int *value*)

Changes the value of a vertex in the infoList.

Parameters

vertexToUpdate	The vertex to update in the infoList.
value	The second half of the edge.

2.1.3.13 bool AdjacencyList::validVertex (int *vertex*) const

Checks to see if a given vertex is within in the adjacency list.

Parameters

vertex	The vertex to search the adjacency list for.

Returns

True if the vertex is valid, False otherwise.

2.1.3.14 int AdjacencyList::vertexCount ( ) const

Gets the number of vertices in the adjacencyList.

Returns

an integer that contains the number of vertices in the adjacencyList.

The documentation for this class was generated from the following files:

- Header_Files/AdjacencyList.h
- Source_Code/AdjacencyList.cpp

2.2 CommunityIdentifier Class Reference

`#include <CommunityIdentifier.h>`

Static Public Member Functions

- static AdjacencyList identifyCommunities (const AdjacencyList &adjacencyList, const std::vector< std::vector< int >> &keyVertices, int numVerticesFirstIter, int numVerticesSubsequentIter, int criticalVerticesNextIter, int depthToSearch)

2.2.1 Detailed Description

This class creates an AdjacencyList containing a subset nodes and edges from the adjacencylist. Nodes and edges are identified the KNV heuristic.

2.2.2 Member Function Documentation

2.2.2.1 AdjacencyList CommunityIdentifier::identifyCommunities (const AdjacencyList & *adjacencyList*, const std::vector< std::vector< int >> & *keyVertices*, int *numVerticesFirstIter*, int *numVerticesSubsequentIter*, int *criticalVerticesNextIter*, int *depthToSearch*) [static]

This function creates a reduced AdjacencyList that contains a subset of nodes and edges from the adjacencyList.

Parameters

adjacencyList	The data structure containing all nodes and edges.
keyVertices	Contains query vertices and path vertices.
numVertices-FirstIter	Controls how many vertices are included in the reduced adjacency list on the first depth level.
numVertices-SubsequentIter	Controls how many vertices are included in the reduced adjacency list on evey depth level except the first.
criticalVertices-NextIter	Controls how many of the vertices in the current iteration are carried into the next iteration.
depthToSearch	Determines how many "hops" or edge connections away from a critial vertex to search.

The documentation for this class was generated from the following files:

- Header_Files/CommunityIdentifier.h
- Source_Code/CommunityIdentifier.cpp

2.3 DijkstrasAlgorithm Class Reference

Static Public Member Functions

- static void singleSourceShortestPath (int startVertex, int endVertex, AdjacencyList adjacencyList, PathInformation &pathInfo, int firstEdgeToDelete, int secondEdgeToDelete, bool removeEdge)

2.3.1 Member Function Documentation

2.3.1.1 void DijkstrasAlgorithm::singleSourceShortestPath (int *startVertex*, int *endVertex*, AdjacencyList *adjacencyList*, PathInformation & *pathInfo*, int *firstEdgeToDelete*, int *secondEdgeToDelete*, bool *removeEdge*) [static]

This function computes the shortest path from a start to end vertex.

Parameters

startVertex	The start point of the path to be calculated.
endVertex	The end point of the path to be calculated.
adjacencyList	An AdjacencyList that hold all the edges for each vertex.
pathInfo	Shortest path information will be stored in here.
firstEdgeTo-Delete	This is the vertex for the first half of the edge to delete.
secondEdgeTo-Delete	This is the vertex for the second half of the edge to delete.
removeEdge	This value determines whether or not to remove the edge given by firstEdgeToDelte and secondEdgeToDelete.

The documentation for this class was generated from the following files:

- Header_Files/DijkstrasAlgorithm.h
- Source_Code/DijkstrasAlgorithm.cpp

2.4 GraphViz Class Reference

```
#include <GraphViz.h>
```

Static Public Member Functions

- static void convertKSSPAdjListToDOT (const AdjacencyList &nonSearchReducedAdjList, const AdjacencyList &adjList, const std::vector< std::vector< int >> &keyVertices, const std::vector< PathInformation > &shortestPaths, const std::unordered_map< int, std::string > &nameList, bool useNameList, int colorSchemeFilter, std::string nodeSize, int edgeSchemeFilter, std::string dotFileLocation, std::string dotFileLocationDefaultFont, std::string adjListLocation)
- static void convertMSTAdjListToDOT (const AdjacencyList &nonSearchReducedAdjList, const AdjacencyList &adjList, const std::vector< std::vector< int >> &keyVertices, const std::vector< PathInformation > &shortestPaths, const std::unordered_map< int, std::string > &nameList, bool useNameList, int colorSchemeFilter, std::string nodeSize, int edgeSchemeFilter, std::string dotFileLocation, std::string dotFileLocationDefaultFont, std::string adjListLocation)

2.4.1 Detailed Description

This class creates a DOT and adjacency list file that contains information on the nodes and edges used to create the visualizations.

2.4.2 Member Function Documentation

2.4.2.1 void GraphViz::convertKSSPAdjListToDOT (const AdjacencyList & *nonSearchReducedAdjList,* const AdjacencyList & *adjList,* const std::vector< std::vector< int >> & *keyVertices,* const std::vector< PathInformation > & *shortestPaths,* const std::unordered_map< int, std::string > & *nameList,* bool *useNameList,* int *colorSchemeFilter,* std::string *nodeSize,* int *edgeSchemeFilter,* std::string *dotFileLocation,* std::string *dotFileLocationDefaultFont,* std::string *adjListLocation*) [static]

Creates a DOT and adjacency list file for the KSSP algorithm. This sets all visualization paramters.

Parameters

nonSearch-ReducedAdjList	This adj. list contains all the orginal edges and vertices from the dataset.
adjList	Reduced adj. list containing all the nodes and edges to be visualized.
keyVertices	Contains the query and path vertices.
shortestPaths	Contains the shortest paths.
nameList	Contains the mapping from integer to name.
useNameList	Determines whether or not to map an integer to a name.
colorScheme-Filter	Determines which vertex color scheme to use.
nodeSize	Determines the vertex size
edgeScheme-Filter	Determines whether the edges are gray or colored.
dotFileLocation	The path to save the DOT file.
dotFileLocation-DefaultFont	The path to save the second DOT file.
adjListLocation	The path to save the adjacency list.

2.4.2.2 void GraphViz::convertMSTAdjListToDOT (const **AdjacencyList** & *nonSearchReducedAdjList,* const **AdjacencyList** & *adjList,* const std::vector< std::vector< int >> & *keyVertices,* const std::vector< **PathInformation** > & *shortestPaths,* const std::unordered_map< int, std::string > & *nameList,* bool *useNameList,* int *colorSchemeFilter,* std::string *nodeSize,* int *edgeSchemeFilter,* std::string *dotFileLocation,* std::string *dotFileLocationDefaultFont,* std::string *adjListLocation*) `[static]`

Creates a DOT and adjacency list file for the MST algorithm. This sets all visualization paramters.

Parameters

nonSearch-ReducedAdjList	This adj. list contains all the orginal edges and vertices from the dataset.
adjList	Reduced adj. list containing all the nodes and edges to be visualized.
keyVertices	Contains the query and path vertices.
shortestPaths	Contains the shortest paths.
nameList	Contains the mapping from integer to name.
useNameList	Determines whether or not to map an integer to a name.
colorScheme-Filter	Determines which vertex color scheme to use.
nodeSize	Determines the vertex size
edgeScheme-Filter	Determines whether the edges are gray or colored.
dotFileLocation	The path to save the DOT file.
dotFileLocation-DefaultFont	The path to save the second DOT file.
adjListLocation	The path to save the adjacency list.

The documentation for this class was generated from the following files:

- Header_Files/GraphViz.h
- Source_Code/GraphViz.cpp

2.5 KSimpleShortestPaths Class Reference

```
#include <KSimpleShortestPaths.h>
```

Static Public Member Functions

- static int runKSSP (const AdjacencyList &adjacencyList, std::vector< std::string > programInfo, std::ofstream &output, bool debug, std::unordered_map< std::string, int > nameListKey, std::unordered_map< int, std::string > nameList, bool useKSSP)
- static void kSSP (int startVertex, int endVertex, int numberOfPathsToCalc, AdjacencyList adjacencyList, std::vector< PathInformation > &kShortestPaths, bool &validPathFound, std::ofstream &output, bool debug)
- static void kSSPR (int startVertex, int endVertex, int numberOfPathsToCalc, AdjacencyList adjacencyList, int depthToSearch, bool debug, std::vector< PathInformation > &kShortestPaths, bool &validPathFound, int numVerticesFirstIter, int numVerticesSubsequentIter, int criticalVerticesNextIter, int numVerticesFirstIter2, int numVerticesSubsequentIter2, int criticalVerticesNextIter2, int depthSSR2, std::ofstream &output, bool searchSpaceReduce)

2.5.1 Detailed Description

This class finds the paths from one vertex to another. There exits both a single threaded and multi-threaded variation.

2.5.2 Member Function Documentation

2.5.2.1 void KSimpleShortestPaths::kSSP (int *startVertex*, int *endVertex*, int *numberOfPathsToCalc*, **AdjacencyList** *adjacencyList*, std::vector< **PathInformation** > & *kShortestPaths*, bool & *validPathFound*, std::ofstream & *output*, bool *debug*) [static]

Computes the K-Simple Shortest Paths in single threaded mode.

Parameters

startVertex	This is the start of the path that all calculations will originate.
endVertex	This is the end destination that all paths will converge to.
numberOfPaths-ToCalc	This is the number of paths to calculate.
adjacencyList	Contains the graph information.
kShortestPaths	Contains the information for each shortest path found.
validPathFound	True if a path was found, false otherwise.
output	An ofstream reference to print time measurements to a file.
debug	If true, measure the run time of the program.

2.5.2.2 void KSimpleShortestPaths::kSSPR (int *startVertex*, int *endVertex*, int *numberOfPathsToCalc*, **AdjacencyList** *adjacencyList*, int *depthToSearch*, bool *debug*, std::vector< **PathInformation** > & *kShortestPaths*, bool & *validPathFound*, int *numVerticesFirstIter*, int *numVerticesSubsequentIter*, int *criticalVerticesNextIter*, int *numVerticesFirstIter2*, int *numVerticesSubsequentIter2*, int *criticalVerticesNextIter2*, int *depthSSR2*, std::ofstream & *output*, bool *searchSpaceReduce*) [static]

Computes the K-Simple Shortest Paths in multi-threaded mode.

Parameters

startVertex	This is the start of the path that all calculations will originate.
endVertex	This is the end destination that all paths will converge to.
numberOfPaths-ToCalc	This is the number of paths to calculate.

adjacencyList	Contains the graph information.
depthToSearch	The depth to search for the first search space reduction (SSR1).
debug	If true, measure the run time of the program.
kShortestPaths	Contains the information for each shortest path found.
validPathFound	True if a path was found, false otherwise.
numVertices-FirstIter	Controls how many vertices are included in the reduced adjacency list on the first depth level (SSR1).
numVertices-SubsequentIter	Controls how many vertices are included in the reduced adjacency list on evey depth level except the first (SSR1).
criticalVertices-NextIter	Controls how many of the vertices in the current iteration are carried into the next iteration (SSR1).
numVertices-FirstIter2	Controls how many vertices are included in the reduced adjacency list on the first depth level (SSR2).
numVertices-SubsequentIter2	Controls how many vertices are included in the reduced adjacency list on evey depth level except the first (SSR2).
criticalVertices-NextIter2	Controls how many of the vertices in the current iteration are carried into the next iteration (SSR2).
depthSSR2	The depth to search for the first search space reduction (SSR2).
output	An ofstream reference to print time measurements to a file.
searchSpace-Reduce	Determines whether or not to use the search space reduction algorithm.

2.5.2.3 int KSimpleShortestPaths::runKSSP (const AdjacencyList & *adjacencyList*, std::vector< std::string > *programInfo*, std::ofstream & *output*, bool *debug*, std::unordered_map< std::string, int > *nameListKey*, std::unordered_map< int, std::string > *nameList*, bool *useKSSP*) `[static]`

Runs the KSSP calculations, creates DOT and adjacency list of results.

Parameters

adjacencyList	Contains the graph information.
programInfo	Contains arguements to program.
output	An ofstream reference to print time measurements to a file.
debug	If true, measure the run time of the program.
nameListKey	Creates a map between the string and integer representation of the data.
nameList	Creates a map between the integer and string representation of the data.
useKSSP	Determines whether or not to use single threaded non-search reduced version of KSSP or multithreaded search reduced version.

Returns

int The status of the program calculations.

The documentation for this class was generated from the following files:

- Header_Files/KSimpleShortestPaths.h
- Source_Code/KSimpleShortestPaths.cpp

2.6 MST Class Reference

```
#include <MST.h>
```

Static Public Member Functions

- static int runMST (const AdjacencyList &adjacencyList, std::vector< std::string > programInfo, std::ofstream &output, bool debug, std::unordered_map< std::string, int > nameListKey, std::unordered_map< int, std::string > nameList)
- static std::vector
< PathInformation > calculateMST (const AdjacencyList &adjacencyList, const std::vector< int > &verticesToSearch, int depthToSearch, bool debug, int numVerticesFirstIter, int numVerticesSubsequentIter, int criticalVerticesNextIter, std::ofstream &output, bool searchSpaceReduce)

2.6.1 Detailed Description

This class finds a MST using shortest paths between a set of user specified query nodes.

2.6.2 Member Function Documentation

2.6.2.1 std::vector< **PathInformation** > MST::calculateMST (const **AdjacencyList** & *adjacencyList,* const std::vector< int > & *verticesToSearch,* int *depthToSearch,* bool *debug,* int *numVerticesFirstIter,* int *numVerticesSubsequentIter,* int *criticalVerticesNextIter,* std::ofstream & *output,* bool *searchSpaceReduce*) [static]

Creates a MST between a set of user specified query nodes.

Parameters

adjacencyList	Contains the graph information.
verticesTo-Search	The set of user specified query nodes
depthToSearch	The depth to search for the first search space reduction (SSR1).
debug	If true, measure the run time of the program.
numVertices-FirstIter	Controls how many vertices are included in the reduced adjacency list on the first depth level (SSR1).
numVertices-SubsequentIter	Controls how many vertices are included in the reduced adjacency list on evey depth level except the first (SSR1).
criticalVertices-NextIter	Controls how many of the vertices in the current iteration are carried into the next iteration (SSR1).
searchSpace-Reduce	Determines whether or not to use the search space reduction algorithm.
output	An ofstream reference to print time measurements to a file.

Returns

PathInformation containing the MST between the query nodes.

2.6.2.2 int MST::runMST (const **AdjacencyList** & *adjacencyList,* std::vector< std::string > *programInfo,* std::ofstream & *output,* bool *debug,* std::unordered_map< std::string, int > *nameListKey,* std::unordered_map< int, std::string > *nameList*) [static]

Runs the MST calculations, creates DOT and adjacency list of results.

Parameters

adjacencyList	Contains the graph information.

programInfo	Contains arguements to program.
output	An ofstream reference to print time measurements to a file.
debug	If true, measure the run time of the program.
nameListKey	Creates a map between the string and integer representation of the data.
nameList	Creates a map between the integer and string representation of the data.

Returns

int The status of the program calculations.

The documentation for this class was generated from the following files:

- Header_Files/MST.h
- Source_Code/MST.cpp

2.7 Parsing Class Reference

Static Public Member Functions

- static void getGraphData (std::string pathToDatasetFile, AdjacencyList &adjacencyList)
- static void getNamedGraphData (std::string pathToDatasetFile, AdjacencyList &adjacencyList, std::unordered_map< int, std::string > &nameList, std::unordered_map< std::string, int > &nameListKey)
- static void removeVertices (AdjacencyList &adjacencyList, const std::unordered_map< std::string, int > &nameListKey, const std::vector< std::string > &programInfo, bool useNameList)
- static void removeEdges (AdjacencyList &adjacencyList, const std::unordered_map< std::string, int > &nameListKey, const std::vector< std::string > &programInfo, bool useNameList)
- static void parseMSTResults (const std::vector< PathInformation > &mstShortestPaths, const std::vector< int > &MSTVertices, std::vector< std::vector< int >> &keyVertices, std::ofstream &output)
- static void parseKSSPResults (std::vector< std::vector< int >> &keyVertices, const std::vector< PathInformation > &shortestPaths, int startVertex, int endVertex)

2.7.1 Member Function Documentation

2.7.1.1 void Parsing::getGraphData (std::string *pathToDatasetFile*, AdjacencyList & *adjacencyList*) `[static]`

Reads in integer graph data.

Parameters

pathToDataset-File	Path to the dataset.
adjacencyList	Data structure to hold all the dataset information.

2.7.1.2 void Parsing::getNamedGraphData (std::string *pathToDatasetFile*, AdjacencyList & *adjacencyList*, std::unordered_map< int, std::string > & *nameList*, std::unordered_map< std::string, int > & *nameListKey*) `[static]`

Reads in string graph data.

Parameters

pathToDataset-File	Path to the dataset.
adjacencyList	Data structure to hold all the dataset information.
nameList	Creates a map between the integer and string representation of the data.
nameListKey	Creates a map between the string and integer representation of the data.

2.7.1.3 void Parsing::parseKSSPResults (std::vector< std::vector< int >> & *keyVertices*, const std::vector< **PathInformation** > & *shortestPaths*, int *startVertex*, int *endVertex*) [static]

Takes the KSSP information and parses the paths.

Parameters

keyVertices	Stores the path and query vertices.
shortestPaths	Contains the shortest paths.
startVertex	The start vertex.
endVertex	The end vertex.

2.7.1.4 void Parsing::parseMSTResults (const std::vector< **PathInformation** > & *mstShortestPaths*, const std::vector< int > & *MSTVertices*, std::vector< std::vector< int >> & *keyVertices*, std::ofstream & *output*) [static]

Takes the MST information and parses the path and query vertices.

Parameters

mstShortest-Paths	Contains the MST path information.
MSTVertices	Contains the MST query vertices.
keyVertices	Stores the path and query vertices.
output	An ofstream reference to print time measurements to a file.

2.7.1.5 void Parsing::removeEdges (**AdjacencyList** & *adjacencyList*, const std::unordered_map< std::string, int > & *nameListKey*, const std::vector< std::string > & *programInfo*, bool *useNameList*) [static]

Removes user specified edges from the adjacency list.

Parameters

adjacencyList	Data structure to have information removed from.
nameListKey	Creates a map between the string and integer
programInfo	Contains the user specified edges to remove from the adjacency list.
useNameList	Determines whether or not to use the nameList mapping.

2.7.1.6 void Parsing::removeVertices (**AdjacencyList** & *adjacencyList*, const std::unordered_map< std::string, int > & *nameListKey*, const std::vector< std::string > & *programInfo*, bool *useNameList*) [static]

Removes user specified vertices from the adjacency list.

Parameters

adjacencyList	Data structure to have information removed from.

nameListKey	Creates a map between the string and integer
programInfo	Contains the user specified vertices to remove from the adjacency list.
useNameList	Determines whether or not to use the nameList mapping.

The documentation for this class was generated from the following files:

- Header_Files/Parsing.h
- Source_Code/Parsing.cpp

2.8 PathInformation Class Reference

```
#include <PathInformation.h>
```

Public Member Functions

- PathInformation ()
- PathInformation (std::vector< int > path, int pathLength)
- std::vector< int > getPath () const
- int getPathAt (int index) const
- int getPathLength () const
- void setPath (std::vector< int > updatedPath)
- void setPathLength (int updatedPathLength)
- std::string toString () const
- bool equals (PathInformation other) const

2.8.1 Detailed Description

This class stores and manipulates data for a string and integer

2.8.2 Constructor & Destructor Documentation

2.8.2.1 PathInformation::PathInformation ( )

Constructor. Creates an empty string for path and sets pathLength to 0.

2.8.2.2 PathInformation::PathInformation (std::vector< int > *path,* int *pathLength*)

Constructor.

Parameters

path	Initializes path with path.
pathLength	Initializes the pathLength with pathLength.

2.8.3 Member Function Documentation

2.8.3.1 bool PathInformation::equals (PathInformation *other*) const

Compares two PathInformation objects.

Returns

true if the paths are equal, false otherwise

2.8.3.2 std::vector< int > PathInformation::getPath ( ) const

Gets the path.

Returns

vector Containing the path information.

2.8.3.3 int PathInformation::getPathAt (int *index*) const

Gets the path information at a given index.

Returns

int The value at the index in the path.

2.8.3.4 int PathInformation::getPathLength ( ) const

Returns the path length

Returns

integer The length of the path.

2.8.3.5 void PathInformation::setPath (std::vector< int > *updatedPath*)

Sets the path

Parameters

updatedPath	Sets the path to updatedPath.

2.8.3.6 void PathInformation::setPathLength (int *updatedPathLength*)

Sets the path length

Parameters

updatedPath-Length	Sets the pathLength to updatedPathLength.

2.8.3.7 std::string PathInformation::toString ( ) const

Returns the path and pathLength

Returns

a string that contains the path and pathLength

The documentation for this class was generated from the following files:

- Header_Files/PathInformation.h
- Source_Code/PathInformation.cpp

2.9 SearchSpaceReduction Class Reference

```
#include <SearchSpaceReduction.h>
```

Static Public Member Functions

- static void searchSpaceReduce (const AdjacencyList &adjacencyList, AdjacencyList &reducedAdjacencyList, const PathInformation &pathToSearch, int depthToSearch, int numVerticesFirstIter, int numVerticesSubsequentIter, int criticalVerticesNextIter)
- static void keyNeighboringVertices (const AdjacencyList &adjacencyList, const PathInformation &pathToSearch, AdjacencyList &updatedAdjacencyList, int &currentDepth, int depthToSearch, int averageCentrality, int numVerticesFirstIter, int numVerticesSubsequentIter, int criticalVerticesNextIter)

2.9.1 Detailed Description

This class creates a subset of the original graph to reduce the search space and computation time of the network connectivity algorithms.

2.9.2 Member Function Documentation

2.9.2.1 void SearchSpaceReduction::keyNeighboringVertices (const **AdjacencyList** & *adjacencyList,* const **PathInformation** & *pathToSearch,* **AdjacencyList** & *updatedAdjacencyList,* int & *currentDepth,* int *depthToSearch,* int *averageCentrality,* int *numVerticesFirstIter,* int *numVerticesSubsequentIter,* int *criticalVerticesNextIter*) `[static]`

This function determines which vertices and edges are important in the graph.

Parameters

adjacencyList	Contains the graph information.
pathToSearch	Contains the important vertices to search in the vicinity of.
updated-AdjacencyList	Contains the reduced adjacency list with all vertices and edges identified as 'key'.
currentDepth	The current depth that the program is at in the breadth first search of the network.
average-Centrality	The average degree centrality of the network.
numVertices-FirstIter	Controls how many vertices are included in the reduced adjacency list on the first depth level.
numVertices-SubsequentIter	Controls how many vertices are included in the reduced adjacency list on evey depth level except the first.
criticalVertices-NextIter	Controls how many of the vertices in the current iteration are carried into the next iteration.
depthToSearch	Determines how many "hops" or edge connections away from a critial vertex to search.

2.9.2.2 void SearchSpaceReduction::searchSpaceReduce (const **AdjacencyList** & *adjacencyList,* **AdjacencyList** & *reducedAdjacencyList,* const **PathInformation** & *pathToSearch,* int *depthToSearch,* int *numVerticesFirstIter,* int *numVerticesSubsequentIter,* int *criticalVerticesNextIter*) `[static]`

Reduces the search space and computation time of the network connectivity algorithms.

Parameters

adjacencyList	The data structure containing all nodes and edges.
reduced-AdjacencyList	This will the search space reduced adj. list containing the key nodes and edges.
pathToSearch	Contains the important vertices to search in the vicinity of.
depthToSearch	Determines how many "hops" or edge connections away from a critial vertex to search.
numVertices-FirstIter	Controls how many vertices are included in the reduced adjacency list on the first depth level.
numVertices-SubsequentIter	Controls how many vertices are included in the reduced adjacency list on evey depth level except the first.
criticalVertices-NextIter	Controls how many of the vertices in the current iteration are carried into the next iteration.

The documentation for this class was generated from the following files:

- Header_Files/SearchSpaceReduction.h
- Source_Code/SearchSpaceReduction.cpp

2.10 Utility Class Reference

```
#include <Utility.h>
```

Static Public Member Functions

- static int vectorIntersection (std::vector< int > v1, std::vector< int > v2)
- static bool vectorIntIntersection (const std::vector< int > &v1, int v2)
- static std::string vectorToString (std::vector< int > &v1)

2.10.1 Detailed Description

This class contains helper functions that are used to manipulate vectors.

2.10.2 Member Function Documentation

2.10.2.1 int Utility::vectorIntersection (std::vector< int > *v1*, std::vector< int > *v2*) [static]

Determines the number of elements that two vectors have in common.

Parameters

v1	The first vector.
v2	The second vector.

Returns

integer containing the number of intersected elements.

2.10.2.2 bool Utility::vectorIntIntersection (const std::vector< int > & *v1*, int *v2*) [static]

Determines if an integer exists in a vector.

Parameters

v1	The vector.
v2	The integer.

Returns

bool True if it's found, false otherwise

2.10.2.3 std::string Utility::vectorToString (std::vector< int > & *v1*) `[static]`

Converts a vector to a string.

Parameters

v1	The vector.

Returns

string containing the vector information.

The documentation for this class was generated from the following files:

- Header_Files/Utility.h
- Source_Code/Utility.cpp

Index