Maze Generator

Generated by Doxygen 1.9.8

1 Class Index 1

1.1 Class List . . . 1

2 File Index 3

2.1 File List . . . 3

3 Class Documentation 5

3.1 AdjList Struct Reference . . . 5

3.1.1 Member Data Documentation . . . 5

3.1.1.1 headNode . . . 5

3.2 FileLoaded Struct Reference . . . 6

3.2.1 Member Data Documentation . . . 6

3.2.1.1 graph . . . 6

3.2.1.2 height . . . 6

3.2.1.3 width . . . 7

3.3 Graph Struct Reference . . . 7

3.3.1 Member Data Documentation . . . 7

3.3.1.1 adjList . . . 7

3.3.1.2 numNodes . . . 8

3.4 Maze Struct Reference . . . 8

3.4.1 Member Data Documentation . . . 8

3.4.1.1 height . . . 8

3.4.1.2 matrix . . . 8

3.4.1.3 width . . . 8

3.5 MinHeap Struct Reference . . . 9

3.5.1 Member Data Documentation . . . 9

3.5.1.1 capacity . . . 9

3.5.1.2 nodes . . . 9

3.5.1.3 positions . . . 9

3.5.1.4 size . . . 10

3.6 MinHeapNode Struct Reference . . . 10

3.6.1 Member Data Documentation . . . 10

3.6.1.1 cost . . . 10

3.6.1.2 idNode . . . 10

3.7 Node Struct Reference . . . 10

3.7.1 Member Data Documentation . . . 11

3.7.1.1 cost . . . 11

3.7.1.2 idDestinationNode . . . 11

3.7.1.3 nextNode . . . 11

3.8 PrimResult Struct Reference . . . 11

3.8.1 Member Data Documentation . . . 11

3.8.1.1 predecessors . . . 11

3.8.1.2 totalCost . . . 11

4 File Documentation **13**

4.1 AppGenerateMaze.h 13

4.2 Graph.h 14

4.3 Heap.h 14

4.4 Load.h 16

4.5 Maze.h 16

4.6 Prim.h 17

Index **19**

Chapter 1

Class Index

1.1 Class List

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

AdjList 5
FileLoaded 6
Graph 7
Maze 8
MinHeap 9
MinHeapNode 10
Node 10
PrimResult 11

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

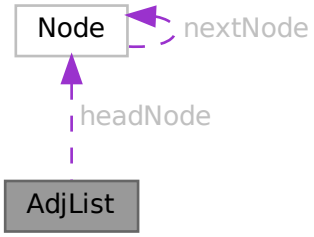
include/AppGenerateMaze.h 13
include/Graph.h 14
include/Heap.h 14
include/Load.h 16
include/Maze.h 16
include/Prim.h 17

Chapter 3

Class Documentation

3.1 AdjList Struct Reference

Collaboration diagram for AdjList:



Public Attributes

- Node * headNode

3.1.1 Member Data Documentation

3.1.1.1 headNode

```
Node* AdjList::headNode
```

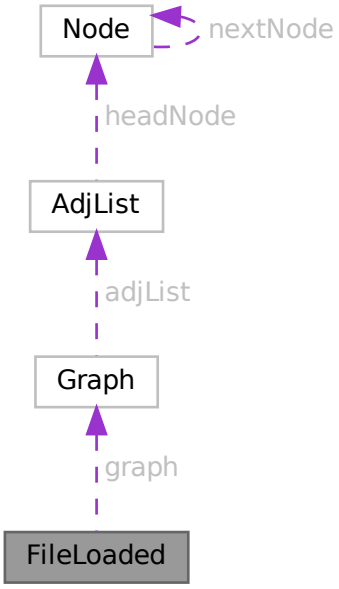
Pointer to the adjacency list for a node.

The documentation for this struct was generated from the following file:

- include/Graph.h

3.2 FileLoaded Struct Reference

Collaboration diagram for FileLoaded:



Public Attributes

- int width
- int height
- Graph * graph

3.2.1 Member Data Documentation

3.2.1.1 graph

```
Graph* FileLoaded::graph
```

The graph loaded from the file.

3.2.1.2 height

```
int FileLoaded::height
```

The height of the loaded graph.

3.2.1.3 width

```
int FileLoaded::width
```

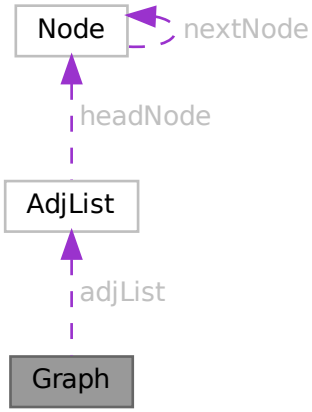
The width of the loaded graph.

The documentation for this struct was generated from the following file:

- include/Load.h

3.3 Graph Struct Reference

Collaboration diagram for Graph:



Public Attributes

- int numNodes
- AdjList * adjList

3.3.1 Member Data Documentation

3.3.1.1 adjList

```
AdjList* Graph::adjList
```

Pointer to the array of adjacency lists.

3.3.1.2 numNodes

```
int Graph::numNodes
```

The number of nodes in the graph.

The documentation for this struct was generated from the following file:

- include/Graph.h

3.4 Maze Struct Reference

Public Attributes

- int width
- int height
- bool ** matrix

3.4.1 Member Data Documentation

3.4.1.1 height

```
int Maze::height
```

Height of the maze.

3.4.1.2 matrix

```
bool** Maze::matrix
```

2D matrix representing the maze.

3.4.1.3 width

```
int Maze::width
```

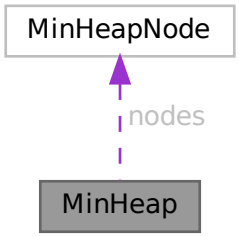
Width of the maze.

The documentation for this struct was generated from the following file:

- include/Maze.h

3.5 MinHeap Struct Reference

Collaboration diagram for MinHeap:



Public Attributes

- int size
- int capacity
- int * positions
- MinHeapNode ** nodes

3.5.1 Member Data Documentation

3.5.1.1 capacity

```
int MinHeap::capacity
```

The maximum capacity of the MinHeap.

3.5.1.2 nodes

```
MinHeapNode** MinHeap::nodes
```

An array of MinHeapNode pointers.

3.5.1.3 positions

```
int* MinHeap::positions
```

An array to store the positions of nodes in the MinHeap.

3.5.1.4 size

```
int MinHeap::size
```

The current size of the MinHeap.

The documentation for this struct was generated from the following file:

- include/Heap.h

3.6 MinHeapNode Struct Reference

Public Attributes

- int idNode
- int cost

3.6.1 Member Data Documentation

3.6.1.1 cost

```
int MinHeapNode::cost
```

The cost associated with the node.

3.6.1.2 idNode

```
int MinHeapNode::idNode
```

The ID of the node.

The documentation for this struct was generated from the following file:

- include/Heap.h

3.7 Node Struct Reference

Collaboration diagram for Node:

Public Attributes

- int idDestinationNode
- int cost
- struct Node * nextNode

3.7.1 Member Data Documentation

3.7.1.1 cost

```
int Node::cost
```

The cost of the edge to the destination node.

3.7.1.2 idDestinationNode

```
int Node::idDestinationNode
```

The ID of the destination node.

3.7.1.3 nextNode

```
struct Node* Node::nextNode
```

Pointer to the next node in the adjacency list.

The documentation for this struct was generated from the following file:

- include/Graph.h

3.8 PrimResult Struct Reference

Public Attributes

- int * predecessors
- int totalCost

3.8.1 Member Data Documentation

3.8.1.1 predecessors

```
int* PrimResult::predecessors
```

Array of predecessors for each node in the MST.

3.8.1.2 totalCost

```
int PrimResult::totalCost
```

Total cost of the minimum spanning tree.

The documentation for this struct was generated from the following file:

- include/Prim.h

Chapter 4

File Documentation

4.1 AppGenerateMaze.h

```
00001 /*****************************************************************************************
00002  * @file AppGenerateMaze.h
00003  * @brief This file contains the declarations of the functions used to generate a maze.
00004  *****************************************************************************************/
00005
00006 #ifndef APP_GENERATE_MAZE_H
00007 #define APP_GENERATE_MAZE_H
00008
00009 #include <dirent.h>
00010 #include "Load.h"
00011
00012 /*****************************************************************************************
00013  * Checks if the user wants to use a maze file.
00014  *
00015  * @return true if the user wants to use a maze file, false otherwise.
00016  *****************************************************************************************/
00017 bool useMazeFile();
00018
00019 /*****************************************************************************************
00020  * Uploads the graph from a maze file.
00021  *
00022  * @param width The width of the maze.
00023  * @param height The height of the maze.
00024  * @param graph The graph to upload the maze to.
00025  *****************************************************************************************/
00026 void uploadGraph(int *width, int *height, Graph *graph);
00027
00028 /*****************************************************************************************
00029  * Generates a graph for the maze.
00030  *
00031  * @param width The width of the maze.
00032  * @param height The height of the maze.
00033  * @param graph The graph to generate the maze on.
00034  *****************************************************************************************/
00035 void generateGraph(int *width, int *height, Graph *graph);
00036
00037 /*****************************************************************************************
00038  * Prompts the user to choose an algorithm for maze generation.
00039  *
00040  * @return The chosen algorithm.
00041  *****************************************************************************************/
00042 int chooseAlgo();
00043
00044 /*****************************************************************************************
00045  * Asks the user if they want to save the generated maze to a file.
00046  *
00047  * @return true if the user wants to save the maze to a file, false otherwise.
00048  *****************************************************************************************/
00049 bool doYouSaveFile();
00050
00051 /*****************************************************************************************
00052  * Saves the generated maze to a file.
00053  *
00054  * @param maze The maze to save.
00055  *****************************************************************************************/
00056 void saveFile(Maze *maze);
00057
00058 #endif // APP_GENERATE_MAZE_H
```

4.2 Graph.h

```
00001 /*****************************************************************************************
00002  * @file Graph.h
00003  * @brief This file contains the declarations of the Graph data structure and related
00004  * functions.
00005  */
00006
00007 #ifndef GRAPH_H
00008 #define GRAPH_H
00009
00010 #include <stdio.h>
00011 #include <stdlib.h>
00012
00013 /*****************************************************************************************
00014  * @struct Node
00015  * @brief Represents a node in the graph.
00016  *****************************************************************************************/
00017 typedef struct Node {
00018     int idDestinationNode;
00019     int cost;
00020     struct Node* nextNode;
00021 } Node;
00022
00023 /*****************************************************************************************
00024  * @struct AdjList
00025  * @brief Represents the adjacency list for a node in the graph.
00026  *****************************************************************************************/
00027 typedef struct {
00028     Node *headNode;
00029 } AdjList;
00030
00031 /*****************************************************************************************
00032  * @struct Graph
00033  * @brief Represents the graph.
00034  *****************************************************************************************/
00035 typedef struct {
00036     int numNodes;
00037     AdjList* adjList;
00038 } Graph;
00039
00040 /*****************************************************************************************
00041  * @brief Creates a new node with the given destination node ID and cost.
00042  * @param idDestinationNode The ID of the destination node.
00043  * @param cost The cost of the edge to the destination node.
00044  * @return Pointer to the newly created node.
00045  *****************************************************************************************/
00046 Node* createNode(int idDestinationNode, int cost);
00047
00048 /*****************************************************************************************
00049  * @brief Creates a new graph with the given number of nodes.
00050  * @param numNodes The number of nodes in the graph.
00051  * @return Pointer to the newly created graph.
00052  *****************************************************************************************/
00053 Graph* createGraph(int numNodes);
00054
00055 /*****************************************************************************************
00056  * @brief Frees the memory allocated for the given graph.
00057  * @param graph Pointer to the graph to be freed.
00058  *****************************************************************************************/
00059 void freeGraph(Graph* graph);
00060
00061 /*****************************************************************************************
00062  * @brief Adds an edge between the source node and the destination node with the given cost.
00063  * @param graph Pointer to the graph.
00064  * @param idSourceNode The ID of the source node.
00065  * @param idDestinationNode The ID of the destination node.
00066  * @param cost The cost of the edge.
00067  *****************************************************************************************/
00068 void addEdge(Graph* graph, int idSourceNode, int idDestinationNode, int cost);
00069
00070 /*****************************************************************************************
00071  * @brief Prints the graph.
00072  * @param graph Pointer to the graph to be printed.
00073  *****************************************************************************************/
00074 void printGraph(Graph* graph);
00075
00076 #endif // GRAPH_H
```

4.3 Heap.h

```
00001 /*****************************************************************************************
00002  * @file Heap.h
```

```
00003  * @brief This file contains the declarations of the MinHeap data structure and related
00004  * functions.
00005  *****************************************************************************************/
00006
00007 #ifndef HEAP_H
00008 #define HEAP_H
00009
00010 #include <stdio.h>
00011 #include <stdlib.h>
00012 #include <stdbool.h>
00013 #include <limits.h>
00014
00015 /*****************************************************************************************
00016  * @struct MinHeapNode
00017  * @brief Represents a node in the MinHeap.
00018  *****************************************************************************************/
00019 typedef struct {
00020     int idNode;
00021     int cost;
00022 } MinHeapNode;
00023
00024 /*****************************************************************************************
00025  * @struct MinHeap
00026  * @brief Represents a MinHeap data structure.
00027  *****************************************************************************************/
00028 typedef struct {
00029     int size;
00030     int capacity;
00031     int *positions;
00032     MinHeapNode **nodes;
00033 } MinHeap;
00034
00035 /*****************************************************************************************
00036  * @brief Creates a new MinHeapNode with the given ID and cost.
00037  * @param idNode The ID of the node.
00038  * @param cost The cost associated with the node.
00039  * @return A pointer to the newly created MinHeapNode.
00040  *****************************************************************************************/
00041 MinHeapNode *createMinHeapNode(int idNode, int cost);
00042
00043 /*****************************************************************************************
00044  * @brief Creates a new MinHeap with the given capacity.
00045  * @param capacity The maximum capacity of the MinHeap.
00046  * @return A pointer to the newly created MinHeap.
00047  *****************************************************************************************/
00048 MinHeap *createMinHeap(int capacity);
00049
00050 /*****************************************************************************************
00051  * @brief Frees the memory allocated for the given MinHeap.
00052  * @param minHeap Pointer to the MinHeap to be freed.
00053  *****************************************************************************************/
00054 void freeMinHeap(MinHeap *minHeap);
00055
00056 /*****************************************************************************************
00057  * @brief Swaps two MinHeapNode pointers.
00058  * @param a Pointer to the first MinHeapNode pointer.
00059  * @param b Pointer to the second MinHeapNode pointer.
00060  *****************************************************************************************/
00061 void swapMinHeapNode(MinHeapNode **a, MinHeapNode **b);
00062
00063 /*****************************************************************************************
00064  * @brief Restores the MinHeap property starting from the given node index.
00065  * @param minHeap Pointer to the MinHeap.
00066  * @param idNode The index of the node to start heapify from.
00067  *****************************************************************************************/
00068 void minHeapify(MinHeap *minHeap, int idNode);
00069
00070 /*****************************************************************************************
00071  * @brief Checks if the MinHeap is empty.
00072  * @param minHeap Pointer to the MinHeap.
00073  * @return True if the MinHeap is empty, false otherwise.
00074  *****************************************************************************************/
00075 bool isMinHeapEmpty(MinHeap *minHeap);
00076
00077 /*****************************************************************************************
00078  * @brief Extracts the node with the minimum cost from the MinHeap.
00079  * @param minHeap Pointer to the MinHeap.
00080  * @return A pointer to the MinHeapNode with the minimum cost.
00081  *****************************************************************************************/
00082 MinHeapNode *extractMinHeapNode(MinHeap *minHeap);
00083
00084 /*****************************************************************************************
00085  * @brief Decreases the cost of the node with the given ID in the MinHeap.
00086  * @param minHeap Pointer to the MinHeap.
00087  * @param idNode The ID of the node.
00088  * @param newCost The new cost for the node.
00089  *****************************************************************************************/
```

```
00090 void decreaseMinHeapNodeCost(MinHeap *minHeap, int idNode, int newCost);
00091
00092 /*****************************************************************************************
00093  * @brief Checks if the node with the given ID is present in the MinHeap.
00094  * @param minHeap Pointer to the MinHeap.
00095  * @param idNode The ID of the node.
00096  * @return True if the node is present in the MinHeap, false otherwise.
00097  *****************************************************************************************/
00098 bool isInMinHeap(MinHeap *minHeap, int idNode);
00099
00100 #endif // HEAP_H
```

4.4 Load.h

```
00001 /*****************************************************************************************
00002  * @file Load.h
00003  * @brief This file contains the declarations of the functions used to load a graph
00004  * from a file.
00005  *****************************************************************************************/
00006
00007 #ifndef LOAD_H
00008 #define LOAD_H
00009
00010 #include "Maze.h"
00011
00012 /*****************************************************************************************
00013  * @struct FileLoaded
00014  * @brief Represents the loaded file data.
00015  *****************************************************************************************/
00016 typedef struct {
00017     int width;
00018     int height;
00019     Graph *graph;
00020 } FileLoaded;
00021
00022 /*****************************************************************************************
00023  * @brief Loads a graph from a file.
00024  *
00025  * @param filename The name of the file to load.
00026  * @return A pointer to the loaded file data.
00027  *****************************************************************************************/
00028 FileLoaded *loadGraph(char *filename);
00029
00030 #endif // LOAD_H
```

4.5 Maze.h

```
00001 /*****************************************************************************************
00002  * @file Maze.h
00003  * @brief This file contains the declarations of the functions used to create and print
00004  * a maze.
00005  *****************************************************************************************/
00006
00007 #ifndef MAZE_H
00008 #define MAZE_H
00009
00010 #include "Prim.h"
00011 #include <string.h>
00012
00013 /*****************************************************************************************
00014  * @struct Maze
00015  * @brief Structure representing a maze.
00016  *****************************************************************************************/
00017 typedef struct {
00018     int width;
00019     int height;
00020     bool **matrix;
00021 } Maze;
00022
00023 /*****************************************************************************************
00024  * @brief Creates a maze based on a given graph and dimensions.
00025  * @param graph The graph used to generate the maze.
00026  * @param width The width of the maze.
00027  * @param height The height of the maze.
00028  * @param typePrim The type of Prim's algorithm to use.
00029  * @return A pointer to the created maze.
00030  *****************************************************************************************/
00031 Maze *createMaze(Graph *graph, int width, int height, char *typePrim);
00032
```

```
00033 /*****************************************************************************************
00034  * @brief Frees the memory allocated for the given maze.
00035  * @param maze The maze to free.
00036  *****************************************************************************************/
00037 void freeMaze(Maze *maze);
00038
00039 /*****************************************************************************************
00040  * @brief Prints the maze as a PPM image file.
00041  * @param maze The maze to print.
00042  * @param filename The name of the PPM image file.
00043  *****************************************************************************************/
00044 void printPPM(Maze *maze, char *filename);
00045
00046 /*****************************************************************************************
00047  * @brief Prints the maze to the console.
00048  * @param maze The maze to print.
00049  *****************************************************************************************/
00050 void printMaze(Maze *maze);
00051
00052 #endif // MAZE_H
```

4.6 Prim.h

```
00001 /*****************************************************************************************
00002  * @file Prim.h
00003  * @brief Header file for Prim's algorithm implementation.
00004  * This file contains the declarations of functions and structures related to Prim's
00005  * algorithm.
00006  * Prim's algorithm is used to find the minimum spanning tree (MST) of a graph.
00007  *****************************************************************************************/
00008
00009 #ifndef PRIM_H
00010 #define PRIM_H
00011
00012 #include "Graph.h"
00013 #include "Heap.h"
00014
00015 /*****************************************************************************************
00016  * @struct PrimResult
00017  * @brief Structure to store the result of Prim's algorithm.
00018  *
00019  * This structure contains the predecessors array and the total cost of the MST.
00020  *****************************************************************************************/
00021 typedef struct {
00022     int *predecessors;
00023     int totalCost;
00024 } PrimResult;
00025
00026 /*****************************************************************************************
00027  * @brief Find the index of the node with the minimum cost that is not yet included in
00028  * the MST.
00029  * This function takes an array of costs and a boolean array indicating whether a node
00030  * is already
00031  * included in the MST. It returns the index of the node with the minimum cost that is
00032  * not yet included
00033  * in the MST.
00034  *
00035  * @param cost The array of costs for each node.
00036  * @param mstSet The boolean array indicating whether a node is already included in the MST.
00037  * @param numNodes The number of nodes in the graph.
00038  * @return The index of the node with the minimum cost that is not yet included in the MST.
00039  *****************************************************************************************/
00040 int minCostIndex(int *cost, bool *mstSet, int numNodes);
00041
00042 /*****************************************************************************************
00043  * @brief Find the minimum spanning tree (MST) of a graph using Prim's algorithm.
00044  *
00045  * This function takes a graph and returns the minimum spanning tree (MST) using Prim's
00046  * algorithm.
00047  * It uses an adjacency matrix representation of the graph.
00048  *
00049  * @param graph The graph for which to find the MST.
00050  * @return A pointer to the PrimResult structure containing the MST.
00051  *****************************************************************************************/
00052 PrimResult *prim(Graph* graph);
00053
00054 /*****************************************************************************************
00055  * @brief Find the minimum spanning tree (MST) of a graph using Prim's algorithm with a
00056  * heap.
00057  *
00058  * This function takes a graph and returns the minimum spanning tree (MST) using Prim's
00059  * algorithm.
00060  * It uses a binary heap to efficiently find the node with the minimum cost.
```

```
00061  *
00062  * @param graph The graph for which to find the MST.
00063  * @return A pointer to the PrimResult structure containing the MST.
00064  ********************************************************************************************/
00065 PrimResult *primHeap(Graph* graph);
00066
00067 /********************************************************************************************
00068  * @brief Print the minimum spanning tree (MST) of a graph.
00069  *
00070  * This function takes a graph, an array of predecessors, and an array of costs, and prints
00071  * the minimum
00072  * spanning tree (MST) of the graph.
00073  *
00074  * @param graph The graph for which to print the MST.
00075  * @param predecessors The array of predecessors for each node in the MST.
00076  * @param costs The array of costs for each node in the MST.
00077  ********************************************************************************************/
00078 void printMST(Graph* graph, int *predecessors, int *costs);
00079
00080 #endif // PRIM_H
```

Index

AdjList, 5
    headNode, 5
adjList
    Graph, 7

capacity
    MinHeap, 9
cost
    MinHeapNode, 10
    Node, 11

FileLoaded, 6
    graph, 6
    height, 6
    width, 6

Graph, 7
    adjList, 7
    numNodes, 7
graph
    FileLoaded, 6

headNode
    AdjList, 5
height
    FileLoaded, 6
    Maze, 8

idDestinationNode
    Node, 11
idNode
    MinHeapNode, 10
include/AppGenerateMaze.h, 13
include/Graph.h, 14
include/Heap.h, 14
include/Load.h, 16
include/Maze.h, 16
include/Prim.h, 17

matrix
    Maze, 8
Maze, 8
    height, 8
    matrix, 8
    width, 8
MinHeap, 9
    capacity, 9
    nodes, 9
    positions, 9
    size, 9
MinHeapNode, 10
    cost, 10
    idNode, 10

nextNode
    Node, 11
Node, 10
    cost, 11
    idDestinationNode, 11
    nextNode, 11
nodes
    MinHeap, 9
numNodes
    Graph, 7

positions
    MinHeap, 9
predecessors
    PrimResult, 11
PrimResult, 11
    predecessors, 11
    totalCost, 11

size
    MinHeap, 9

totalCost
    PrimResult, 11

width
    FileLoaded, 6
    Maze, 8