maze

Generated by Doxygen 1.8.11

Contents

1 README **1**

2 Data Structure Index **3**

2.1 Data Structures 3

3 File Index **5**

3.1 File List 5

4 Data Structure Documentation **7**

4.1 _item Struct Reference 7

4.1.1 Field Documentation 7

4.1.1.1 data 7

4.1.1.2 priority 7

4.2 _map Struct Reference 7

4.2.1 Field Documentation 8

4.2.1.1 capacity 8

4.2.1.2 data 8

4.2.1.3 size 8

4.3 _pqueue Struct Reference 8

4.3.1 Field Documentation 8

4.3.1.1 capacity 8

4.3.1.2 cmp 8

4.3.1.3 data 8

4.3.1.4 size 8

4.4 _vmap Struct Reference 8

4.4.1 Field Documentation . . . 9
4.4.1.1 capacity . . . 9
4.4.1.2 data . . . 9
4.4.1.3 size . . . 9
4.5 edge_ Struct Reference . . . 9
4.5.1 Field Documentation . . . 9
4.5.1.1 next . . . 9
4.5.1.2 to . . . 9
4.5.1.3 weight . . . 9
4.6 entry Struct Reference . . . 9
4.6.1 Field Documentation . . . 10
4.6.1.1 key . . . 10
4.6.1.2 next . . . 10
4.6.1.3 value . . . 10
4.6.1.4 value . . . 10
4.7 graph Struct Reference . . . 10
4.7.1 Field Documentation . . . 10
4.7.1.1 root . . . 10
4.8 node_ Struct Reference . . . 10
4.8.1 Field Documentation . . . 11
4.8.1.1 edges . . . 11
4.8.1.2 name . . . 11
4.8.1.3 next . . . 11

5 File Documentation **13**

5.1 Dijkstra.c File Reference . . . 13

5.2 Dijkstra.h File Reference . . . 13

5.2.1 Function Documentation . . . 13

5.2.1.1 Dijkstra_path(const Graph *g, const char *start, const char *end, char ***path) 13

5.2.1.2 Dijkstra_solveMaze(char **mazeFromFile, char **route, size_t hops) . . . 14

5.3 dpqueue/driver.c File Reference . . . 14

5.3.1 Function Documentation . . . 14

5.3.1.1 heap_print(const pqueue *pq) . . . 14

5.3.1.2 main(void) . . . 14

5.4 graph/driver.c File Reference . . . 14

5.4.1 Function Documentation . . . 15

5.4.1.1 main(void) . . . 15

5.5 map/driver.c File Reference . . . 15

5.5.1 Function Documentation . . . 15

5.5.1.1 hashtable_print(map *m) . . . 15

5.5.1.2 main(void) . . . 15

5.6 dpqueue/heap.c File Reference . . . 15

5.6.1 Function Documentation . . . 16

5.6.1.1 heap_print(const pqueue *pq) . . . 16

5.6.1.2 pqueue_create(int(*cmp)(void *, void *)) . . . 16

5.6.1.3 pqueue_dequeue(pqueue *pq, void **item) . . . 16

5.6.1.4 pqueue_destroy(pqueue *pq) . . . 16

5.6.1.5 pqueue_enqueue(pqueue *pq, void *item, double priority) . . . 16

5.6.1.6 pqueue_reprioritize(pqueue *pq, void *item, double priority) . . . 16

5.6.1.7 pqueue_search(const pqueue *pq, void *item) . . . 16

5.6.1.8 pqueue_size(const pqueue *pq) . . . 16

5.7 dpqueue/pqueue.c File Reference . . . 16

5.7.1 Function Documentation . . . 17

5.7.1.1 heap_print(const pqueue *pq) . . . 17

5.7.1.2 pqueue_create(int(*cmp)(void *, void *)) . . . 17
5.7.1.3 pqueue_dequeue(pqueue *pq, void **item) . . . 17
5.7.1.4 pqueue_destroy(pqueue *pq) . . . 17
5.7.1.5 pqueue_enqueue(pqueue *pq, void *item, double priority) . . . 17
5.7.1.6 pqueue_reprioritize(pqueue *pq, void *item, double priority) . . . 17
5.7.1.7 pqueue_search(const pqueue *pq, void *item) . . . 17
5.7.1.8 pqueue_size(const pqueue *pq) . . . 17
5.8 dpqueue/pqueue.h File Reference . . . 17
5.8.1 Typedef Documentation . . . 18
5.8.1.1 pqueue . . . 18
5.8.2 Function Documentation . . . 18
5.8.2.1 pqueue_create(int(*cmp)(void *, void *)) . . . 18
5.8.2.2 pqueue_dequeue(pqueue *pq, void **item) . . . 18
5.8.2.3 pqueue_destroy(pqueue *pq) . . . 18
5.8.2.4 pqueue_enqueue(pqueue *pq, void *item, double priority) . . . 18
5.8.2.5 pqueue_reprioritize(pqueue *pq, void *item, double priority) . . . 18
5.8.2.6 pqueue_search(const pqueue *pq, void *item) . . . 18
5.8.2.7 pqueue_size(const pqueue *pq) . . . 18
5.9 graph/adjlist.c File Reference . . . 18
5.9.1 Macro Definition Documentation . . . 19
5.9.1.1 _XOPEN_SOURCE . . . 19
5.9.2 Typedef Documentation . . . 19
5.9.2.1 edge_ . . . 19
5.9.2.2 node_ . . . 19
5.9.3 Function Documentation . . . 19
5.9.3.1 Graph_addEdge(Graph *g, const char *from, const char *to, double weight) . . . 19
5.9.3.2 Graph_addNode(Graph *g, const char *name) . . . 20
5.9.3.3 Graph_create(void) . . . 20
5.9.3.4 Graph_deleteEdge(Graph *g, const char *from, const char *to) . . . 20
5.9.3.5 Graph_deleteNode(Graph *g, const char *name) . . . 20

5.9.3.6 Graph_disassemble(Graph *g) . . . 21
5.9.3.7 Graph_getEdgeWeight(const Graph *g, const char *from, const char *to) . . . 21
5.9.3.8 Graph_getNeighbors(const Graph *g, const char *name, char ***neighbors) . . 21
5.9.3.9 Graph_getNodes(const Graph *g, char ***nodes) . . . 21
5.9.3.10 Graph_isAdjacent(const Graph *g, const char *from, const char *to) . . . 22
5.9.3.11 Graph_print(const Graph *g) . . . 22
5.10 graph/Graph.c File Reference . . . 22
5.10.1 Macro Definition Documentation . . . 23
5.10.1.1 _XOPEN_SOURCE . . . 23
5.10.2 Typedef Documentation . . . 23
5.10.2.1 edge_ . . . 23
5.10.2.2 node_ . . . 23
5.10.3 Function Documentation . . . 23
5.10.3.1 Graph_addEdge(Graph *g, const char *from, const char *to, double weight) . . 23
5.10.3.2 Graph_addNode(Graph *g, const char *name) . . . 24
5.10.3.3 Graph_create(void) . . . 24
5.10.3.4 Graph_deleteEdge(Graph *g, const char *from, const char *to) . . . 24
5.10.3.5 Graph_deleteNode(Graph *g, const char *name) . . . 25
5.10.3.6 Graph_disassemble(Graph *g) . . . 25
5.10.3.7 Graph_getEdgeWeight(const Graph *g, const char *from, const char *to) . . . 25
5.10.3.8 Graph_getNeighbors(const Graph *g, const char *name, char ***neighbors) . . 25
5.10.3.9 Graph_getNodes(const Graph *g, char ***nodes) . . . 26
5.10.3.10 Graph_isAdjacent(const Graph *g, const char *from, const char *to) . . . 26
5.10.3.11 Graph_print(const Graph *g) . . . 26
5.11 graph/Graph.h File Reference . . . 26
5.11.1 Typedef Documentation . . . 27
5.11.1.1 Graph . . . 27
5.11.2 Function Documentation . . . 27
5.11.2.1 Graph_addEdge(Graph *g, const char *from, const char *to, double weight) . . 27
5.11.2.2 Graph_addNode(Graph *g, const char *name) . . . 27

5.11.2.3 Graph_create(void) . . . 28
5.11.2.4 Graph_deleteEdge(Graph *g, const char *from, const char *to) . . . 28
5.11.2.5 Graph_deleteNode(Graph *g, const char *name) . . . 28
5.11.2.6 Graph_disassemble(Graph *g) . . . 28
5.11.2.7 Graph_getEdgeWeight(const Graph *g, const char *from, const char *to) . . . 29
5.11.2.8 Graph_getNeighbors(const Graph *g, const char *name, char ***neighbors) . . . 29
5.11.2.9 Graph_getNodes(const Graph *g, char ***nodes) . . . 29
5.11.2.10 Graph_isAdjacent(const Graph *g, const char *from, const char *to) . . . 29
5.11.2.11 Graph_print(const Graph *g) . . . 30
5.12 graph/GraphSerializer.c File Reference . . . 30
5.12.1 Macro Definition Documentation . . . 31
5.12.1.1 _GNU_SOURCE . . . 31
5.12.2 Function Documentation . . . 31
5.12.2.1 GraphSerializer_fromFile(FILE *fp, char ***mazeFromFile, size_t *maxLength, size_t *lineCount, char flags) . . . 31
5.12.2.2 GraphSerializer_fromMaze(Graph **g, char **maze, size_t *maxLength, size_t *lineCount, char flags) . . . 31
5.12.2.3 GraphSerializer_toStdout(const Graph *g) . . . 31
5.13 graph/GraphSerializer.h File Reference . . . 32
5.13.1 Function Documentation . . . 32
5.13.1.1 GraphSerializer_fromFile(FILE *fp, char ***mazeFromFile, size_t *maxLength, size_t *lineCount, char flags) . . . 32
5.13.1.2 GraphSerializer_fromMaze(Graph **g, char **maze, size_t *maxLength, size_t *lineCount, char flags) . . . 32
5.13.1.3 GraphSerializer_toStdout(const Graph *g) . . . 33
5.14 graph/test/test01.c File Reference . . . 33
5.14.1 Function Documentation . . . 33
5.14.1.1 main(void) . . . 33
5.15 graph/test/test02.c File Reference . . . 33
5.15.1 Function Documentation . . . 33
5.15.1.1 main(void) . . . 33
5.16 graph/test/test03.c File Reference . . . 33

5.16.1 Function Documentation . . . 34
5.16.1.1 main(void) . . . 34
5.17 map/hashtable.c File Reference . . . 34
5.17.1 Macro Definition Documentation . . . 35
5.17.1.1 _XOPEN_SOURCE . . . 35
5.17.2 Function Documentation . . . 35
5.17.2.1 hashtable_print(map *m) . . . 35
5.17.2.2 map_create(void) . . . 35
5.17.2.3 map_destroy(map *m) . . . 35
5.17.2.4 map_exists(map *m, const char *key) . . . 35
5.17.2.5 map_insert(map *m, const char *key, double value) . . . 35
5.17.2.6 map_lookup(map *m, const char *key) . . . 36
5.18 map/map.c File Reference . . . 36
5.18.1 Macro Definition Documentation . . . 37
5.18.1.1 _XOPEN_SOURCE . . . 37
5.18.2 Function Documentation . . . 37
5.18.2.1 hashtable_print(map *m) . . . 37
5.18.2.2 map_create(void) . . . 37
5.18.2.3 map_destroy(map *m) . . . 37
5.18.2.4 map_exists(map *m, const char *key) . . . 37
5.18.2.5 map_insert(map *m, const char *key, double value) . . . 37
5.18.2.6 map_lookup(map *m, const char *key) . . . 38
5.19 map/map.h File Reference . . . 38
5.19.1 Typedef Documentation . . . 38
5.19.1.1 map . . . 38
5.19.2 Function Documentation . . . 38
5.19.2.1 map_create(void) . . . 38
5.19.2.2 map_destroy(map *m) . . . 39
5.19.2.3 map_exists(map *m, const char *key) . . . 39
5.19.2.4 map_insert(map *m, const char *key, double value) . . . 39

5.19.2.5 map_lookup(map *m, const char *key) . . . 39
5.20 map/vmap.c File Reference . . . 40
5.20.1 Macro Definition Documentation . . . 40
5.20.1.1 _XOPEN_SOURCE . . . 40
5.20.2 Function Documentation . . . 40
5.20.2.1 vmap_create(void) . . . 40
5.20.2.2 vmap_destroy(vmap *m) . . . 40
5.20.2.3 vmap_exists(vmap *m, const char *key) . . . 41
5.20.2.4 vmap_insert(vmap *m, const char *key, void *value) . . . 41
5.20.2.5 vmap_lookup(vmap *m, const char *key) . . . 41
5.21 map/vmap.h File Reference . . . 42
5.21.1 Typedef Documentation . . . 42
5.21.1.1 vmap . . . 42
5.21.2 Function Documentation . . . 42
5.21.2.1 vmap_create(void) . . . 42
5.21.2.2 vmap_destroy(vmap *m) . . . 42
5.21.2.3 vmap_exists(vmap *m, const char *key) . . . 43
5.21.2.4 vmap_insert(vmap *m, const char *key, void *value) . . . 43
5.21.2.5 vmap_lookup(vmap *m, const char *key) . . . 43
5.22 maze.c File Reference . . . 43
5.22.1 Function Documentation . . . 44
5.22.1.1 main(int argc, char *argv[]) . . . 44
5.22.1.2 Maze_generate(char ***customMaze, size_t maxLineLength, size_t lineCount) . . . 44
5.23 README.md File Reference . . . 44
5.24 tests/door-water.txt File Reference . . . 44
5.25 tests/door.txt File Reference . . . 44
5.26 tests/map01.txt File Reference . . . 44
5.27 tests/map02.txt File Reference . . . 44
5.28 tests/map03.txt File Reference . . . 44
5.29 tests/map04.txt File Reference . . . 45
5.30 tests/map05.txt File Reference . . . 45
5.31 tests/water.txt File Reference . . . 45

Index **47**

Chapter 1

README

Maze Solver

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

_item 7
_map 7
_pqueue 8
_vmap 8
edge_ 9
entry 9
graph 10
node_ 10

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Dijkstra.c . . . 13
Dijkstra.h . . . 13
maze.c . . . 43
dpqueue/driver.c . . . 14
dpqueue/heap.c . . . 15
dpqueue/pqueue.c . . . 16
dpqueue/pqueue.h . . . 17
graph/adjlist.c . . . 18
graph/driver.c . . . 14
graph/Graph.c . . . 22
graph/Graph.h . . . 26
graph/GraphSerializer.c . . . 30
graph/GraphSerializer.h . . . 32
graph/test/test01.c . . . 33
graph/test/test02.c . . . 33
graph/test/test03.c . . . 33
map/driver.c . . . 15
map/hashtable.c . . . 34
map/map.c . . . 36
map/map.h . . . 38
map/vmap.c . . . 40
map/vmap.h . . . 42

Chapter 4

Data Structure Documentation

4.1 _item Struct Reference

Data Fields

- void * data
- double priority

4.1.1 Field Documentation

4.1.1.1 void * _item::data

4.1.1.2 double _item::priority

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

- dpqueue/heap.c
- dpqueue/pqueue.c

4.2 _map Struct Reference

Collaboration diagram for _map:

Data Fields

- struct entry * data
- size_t size
- size_t capacity

4.2.1 Field Documentation

4.2.1.1 size_t _map::capacity

4.2.1.2 struct entry * _map::data

4.2.1.3 size_t _map::size

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

- map/hashtable.c
- map/map.c

4.3 _pqueue Struct Reference

Collaboration diagram for _pqueue:

Data Fields

- struct _item * data
- int(* cmp)(void *, void *)
- size_t size
- size_t capacity

4.3.1 Field Documentation

4.3.1.1 size_t _pqueue::capacity

4.3.1.2 int(* _pqueue::cmp)(void *, void *)

4.3.1.3 struct _item * _pqueue::data

4.3.1.4 size_t _pqueue::size

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

- dpqueue/heap.c
- dpqueue/pqueue.c

4.4 _vmap Struct Reference

Collaboration diagram for _vmap:

Data Fields

- struct entry * data
- size_t size
- size_t capacity

4.4.1 Field Documentation

4.4.1.1 size_t _vmap::capacity

4.4.1.2 struct entry* _vmap::data

4.4.1.3 size_t _vmap::size

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

- map/vmap.c

4.5 edge_ Struct Reference

Collaboration diagram for edge_:

Data Fields

- double weight
- struct node_ * to
- struct edge_ * next

4.5.1 Field Documentation

4.5.1.1 struct edge_ * edge_::next

4.5.1.2 struct node_ * edge_::to

4.5.1.3 double edge_::weight

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

- graph/adjlist.c
- graph/Graph.c

4.6 entry Struct Reference

Collaboration diagram for entry:

Data Fields

- char * key
- double value
- struct entry * next
- void * value

4.6.1 Field Documentation

4.6.1.1 char * entry::key

4.6.1.2 struct entry * entry::next

4.6.1.3 void* entry::value

4.6.1.4 double entry::value

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

- map/hashtable.c
- map/map.c
- map/vmap.c

4.7 graph Struct Reference

Collaboration diagram for graph:

Data Fields

- struct node_ * root

4.7.1 Field Documentation

4.7.1.1 struct node_ * graph::root

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

- graph/adjlist.c
- graph/Graph.c

4.8 node_ Struct Reference

Collaboration diagram for node_:

Data Fields

- char * name
- struct node_ * next
- struct edge_ * edges

4.8.1 Field Documentation

4.8.1.1 struct edge_ * node_::edges

4.8.1.2 char * node_::name

4.8.1.3 struct node_ * node_::next

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

- graph/adjlist.c
- graph/Graph.c

Chapter 5

File Documentation

5.1 Dijkstra.c File Reference

```
#include "Dijkstra.h"
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include "map/map.h"
#include "map/vmap.h"
#include "dpqueue/pqueue.h"
```

Include dependency graph for Dijkstra.c:

5.2 Dijkstra.h File Reference

```
#include <unistd.h>
#include "graph/Graph.h"
```

Include dependency graph for Dijkstra.h: This graph shows which files directly or indirectly include this file:

Functions

- ssize_t Dijkstra_path (const Graph *g, const char *start, const char *end, char ***path)

  Find path between start and end.
- void Dijkstra_solveMaze (char **mazeFromFile, char **route, size_t hops)

  Changes 2d array of the maze and changes the route to show the solved maze.

5.2.1 Function Documentation

5.2.1.1 ssize_t Dijkstra_path (const Graph * *g*, const char * *start*, const char * *end*, char *** *path*)

Find path between start and end.

Parameters

g	Graph to traverse
start	starting node in graph
end	ending node in graph
path	output parameter which is filled with shortest start-to-end path

Returns

number of nodes in path (negative if error or disconnected)

5.2.1.2 void Dijkstra_solveMaze (char ** *mazeFromFile,* char ** *route,* size_t *hops*)

Changes 2d array of the maze and changes the route to show the solved maze.

Parameters

mazeFromFile	2d array of maze
route	Nodes in graph that lead to end
hops	Number of hops for tracking in loop

5.3 dpqueue/driver.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "pqueue.h"
```

Include dependency graph for driver.c:

Functions

- void heap_print (const pqueue *pq)
- int main (void)

5.3.1 Function Documentation

5.3.1.1 void heap_print (const pqueue * *pq*)

5.3.1.2 int main (void)

5.4 graph/driver.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "Graph.h"
#include "GraphSerializer.h"
```

Include dependency graph for driver.c:

Functions

- int main (void)

5.4.1 Function Documentation

5.4.1.1 int main (void)

5.5 map/driver.c File Reference

```
#include <stdio.h>
#include "map.h"
```

Include dependency graph for driver.c:

Functions

- void hashtable_print (map *m)
- int main (void)

5.5.1 Function Documentation

5.5.1.1 void hashtable_print (map * *m*)

5.5.1.2 int main (void)

5.6 dpqueue/heap.c File Reference

```
#include "pqueue.h"
#include <math.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for heap.c:

Data Structures

- struct _item
- struct _pqueue

Functions

- pqueue * pqueue_create (int(*cmp)(void *, void *))
- size_t pqueue_size (const pqueue *pq)
- bool pqueue_enqueue (pqueue *pq, void *item, double priority)
- bool pqueue_reprioritize (pqueue *pq, void *item, double priority)
- double pqueue_dequeue (pqueue *pq, void **item)
- void * pqueue_search (const pqueue *pq, void *item)
- void pqueue_destroy (pqueue *pq)
- void heap_print (const pqueue *pq)

5.6.1 Function Documentation

5.6.1.1 void heap_print (const pqueue * pq)

5.6.1.2 pqueue* pqueue_create (int(*)(void *, void *) cmp)

5.6.1.3 double pqueue_dequeue (pqueue * pq, void ** item)

5.6.1.4 void pqueue_destroy (pqueue * pq)

5.6.1.5 bool pqueue_enqueue (pqueue * pq, void * item, double priority)

5.6.1.6 bool pqueue_reprioritize (pqueue * pq, void * item, double priority)

5.6.1.7 void* pqueue_search (const pqueue * pq, void * item)

5.6.1.8 size_t pqueue_size (const pqueue * pq)

5.7 dpqueue/pqueue.c File Reference

```
#include "pqueue.h"
#include <math.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for pqueue.c:

Data Structures

- struct _item
- struct _pqueue

Functions

- pqueue * pqueue_create (int(*cmp)(void *, void *))
- size_t pqueue_size (const pqueue *pq)
- bool pqueue_enqueue (pqueue *pq, void *item, double priority)
- bool pqueue_reprioritize (pqueue *pq, void *item, double priority)
- double pqueue_dequeue (pqueue *pq, void **item)
- void * pqueue_search (const pqueue *pq, void *item)
- void pqueue_destroy (pqueue *pq)
- void heap_print (const pqueue *pq)

5.7.1 Function Documentation

5.7.1.1 void heap_print (const pqueue * pq)

5.7.1.2 pqueue* pqueue_create (int(*)(void *, void *) cmp)

5.7.1.3 double pqueue_dequeue (pqueue * pq, void ** item)

5.7.1.4 void pqueue_destroy (pqueue * pq)

5.7.1.5 bool pqueue_enqueue (pqueue * pq, void * item, double priority)

5.7.1.6 bool pqueue_reprioritize (pqueue * pq, void * item, double priority)

5.7.1.7 void* pqueue_search (const pqueue * pq, void * item)

5.7.1.8 size_t pqueue_size (const pqueue * pq)

5.8 dpqueue/pqueue.h File Reference

```
#include <stdbool.h>
#include <stdlib.h>
```

Include dependency graph for pqueue.h: This graph shows which files directly or indirectly include this file:

Typedefs

- typedef struct _pqueue pqueue

Functions

- pqueue * pqueue_create (int(*cmp)(void *, void *))
- size_t pqueue_size (const pqueue *pq)
- bool pqueue_enqueue (pqueue *pq, void *item, double priority)
- bool pqueue_reprioritize (pqueue *pq, void *item, double priority)
- double pqueue_dequeue (pqueue *pq, void **item)
- void * pqueue_search (const pqueue *pq, void *item)
- void pqueue_destroy (pqueue *pq)

5.8.1 Typedef Documentation

5.8.1.1 typedef struct _pqueue pqueue

5.8.2 Function Documentation

5.8.2.1 pqueue* pqueue_create (int(*)(void *, void *) *cmp*)

5.8.2.2 double pqueue_dequeue (pqueue * *pq*, void ** *item*)

5.8.2.3 void pqueue_destroy (pqueue * *pq*)

5.8.2.4 bool pqueue_enqueue (pqueue * *pq*, void * *item*, double *priority*)

5.8.2.5 bool pqueue_reprioritize (pqueue * *pq*, void * *item*, double *priority*)

5.8.2.6 void* pqueue_search (const pqueue * *pq*, void * *item*)

5.8.2.7 size_t pqueue_size (const pqueue * *pq*)

5.9 graph/adjlist.c File Reference

```
#include "Graph.h"
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for adjlist.c:

Data Structures

- struct edge_
- struct node_
- struct graph

Macros

- #define _XOPEN_SOURCE 500

Typedefs

- typedef struct edge_ edge_
- typedef struct node_ node_

Functions

- Graph * Graph_create (void)
  Create an empty graph structure.
- bool Graph_addNode (Graph *g, const char *name)
  Adds a node to the graph (does not add duplicates)
- bool Graph_addEdge (Graph *g, const char *from, const char *to, double weight)
  Adds an edge to the graph (does not add duplicates)
- bool Graph_isAdjacent (const Graph *g, const char *from, const char *to)
  Checks if two nodes are adjacent.
- ssize_t Graph_getNodes (const Graph *g, char ***nodes)
  provide list of nodes of a graph
- ssize_t Graph_getNeighbors (const Graph *g, const char *name, char ***neighbors)
  provide list of neighbor's names for a given node
- double Graph_getEdgeWeight (const Graph *g, const char *from, const char *to)
  Provide edge weight between two nodes.
- void Graph_deleteNode (Graph *g, const char *name)
  Remove a node from the graph.
- void Graph_deleteEdge (Graph *g, const char *from, const char *to)
  Remove an edge from the graph.
- void Graph_print (const Graph *g)
  Prints graph to stdout.
- void Graph_disassemble (Graph *g)
  Destroy the graph scaffolding without affecting the underlying data.

5.9.1 Macro Definition Documentation

5.9.1.1 #define _XOPEN_SOURCE 500

5.9.2 Typedef Documentation

5.9.2.1 typedef struct edge_ edge_

5.9.2.2 typedef struct node_ node_

5.9.3 Function Documentation

5.9.3.1 bool Graph_addEdge (Graph * *g*, const char * *from*, const char * *to*, double *weight*)

Adds an edge to the graph (does not add duplicates)

Parameters

g	Graph to modify
from	Name of source node
to	Name of destination node
weight	Cost of the edge

Returns

true for successful add

5.9.3.2 bool Graph_addNode (Graph * g, const char * name)

Adds a node to the graph (does not add duplicates)

Parameters

g	Graph to modify
name	Name of new node

Returns

true for successful add

5.9.3.3 Graph* Graph_create (void)

Create an empty graph structure.

Returns

the graph structure, or NULL on error

5.9.3.4 void Graph_deleteEdge (Graph * g, const char * from, const char * to)

Remove an edge from the graph.

Parameters

g	Graph to alter
from	Starting node of edge
to	Ending node of edge

5.9.3.5 void Graph_deleteNode (Graph * g, const char * name)

Remove a node from the graph.

Parameters

g	Graph to alter
name	Name of node to remove

5.9.3.6 void Graph_disassemble (Graph * g)

Destroy the graph scaffolding without affecting the underlying data.

Parameters

g	Graph to disassemble

5.9.3.7 double Graph_getEdgeWeight (const Graph * g, const char * from, const char * to)

Provide edge weight between two nodes.

Parameters

g	Graph to inspect
from	Starting node
to	neighbor node

Returns

weight of edge (NAN if edge does not exist)

5.9.3.8 ssize_t Graph_getNeighbors (const Graph * g, const char * name, char *** neighbors)

provide list of neighbor's names for a given node

Parameters

g	Graph to inspect
name	node's name to find neighbors for
neighbors	input parameter to store array of neighbors' names

Returns

number of neighbors found (-1 for error)

5.9.3.9 ssize_t Graph_getNodes (const Graph * g, char *** nodes)

provide list of nodes of a graph

Parameters

g	Graph to inspect
nodes	input parameter to store array of nodes' names

Returns

number of nodes found (-1 for error)

5.9.3.10 bool Graph_isAdjacent (const Graph * g, const char * from, const char * to)

Checks if two nodes are adjacent.

Parameters

g	Graph to inspect
from	Name of source node
to	Name of destination node

Returns

True if nodes are adjacent, false otherwise

5.9.3.11 void Graph_print (const Graph * g)

Prints graph to stdout.

Parameters

g	Graph to print

5.10 graph/Graph.c File Reference

```
#include "Graph.h"
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for Graph.c:

Data Structures

- struct edge_
- struct node_
- struct graph

Macros

- #define _XOPEN_SOURCE 500

Typedefs

- typedef struct edge_ edge_
- typedef struct node_ node_

Functions

- Graph * Graph_create (void)

  Create an empty graph structure.
- bool Graph_addNode (Graph *g, const char *name)

  Adds a node to the graph (does not add duplicates)
- bool Graph_addEdge (Graph *g, const char *from, const char *to, double weight)

  Adds an edge to the graph (does not add duplicates)
- bool Graph_isAdjacent (const Graph *g, const char *from, const char *to)

  Checks if two nodes are adjacent.
- ssize_t Graph_getNodes (const Graph *g, char ***nodes)

  provide list of nodes of a graph
- ssize_t Graph_getNeighbors (const Graph *g, const char *name, char ***neighbors)

  provide list of neighbor's names for a given node
- double Graph_getEdgeWeight (const Graph *g, const char *from, const char *to)

  Provide edge weight between two nodes.
- void Graph_deleteNode (Graph *g, const char *name)

  Remove a node from the graph.
- void Graph_deleteEdge (Graph *g, const char *from, const char *to)

  Remove an edge from the graph.
- void Graph_print (const Graph *g)

  Prints graph to stdout.
- void Graph_disassemble (Graph *g)

  Destroy the graph scaffolding without affecting the underlying data.

5.10.1 Macro Definition Documentation

5.10.1.1 #define _XOPEN_SOURCE 500

5.10.2 Typedef Documentation

5.10.2.1 typedef struct edge_ edge_

5.10.2.2 typedef struct node_ node_

5.10.3 Function Documentation

5.10.3.1 bool Graph_addEdge (Graph * *g*, const char * *from*, const char * *to*, double *weight*)

Adds an edge to the graph (does not add duplicates)

Parameters

g	Graph to modify
from	Name of source node
to	Name of destination node
weight	Cost of the edge

Returns

true for successful add

5.10.3.2 bool Graph_addNode (Graph * g, const char * name)

Adds a node to the graph (does not add duplicates)

Parameters

g	Graph to modify
name	Name of new node

Returns

true for successful add

5.10.3.3 Graph* Graph_create (void)

Create an empty graph structure.

Returns

the graph structure, or NULL on error

5.10.3.4 void Graph_deleteEdge (Graph * g, const char * from, const char * to)

Remove an edge from the graph.

Parameters

g	Graph to alter
from	Starting node of edge
to	Ending node of edge

5.10.3.5 void Graph_deleteNode (Graph * g, const char * name)

Remove a node from the graph.

Parameters

g	Graph to alter
name	Name of node to remove

5.10.3.6 void Graph_disassemble (Graph * g)

Destroy the graph scaffolding without affecting the underlying data.

Parameters

g	Graph to disassemble

5.10.3.7 double Graph_getEdgeWeight (const Graph * g, const char * from, const char * to)

Provide edge weight between two nodes.

Parameters

g	Graph to inspect
from	Starting node
to	neighbor node

Returns

weight of edge (NAN if edge does not exist)

5.10.3.8 ssize_t Graph_getNeighbors (const Graph * g, const char * name, char *** neighbors)

provide list of neighbor's names for a given node

Parameters

g	Graph to inspect
name	node's name to find neighbors for
neighbors	input parameter to store array of neighbors' names

Returns

number of neighbors found (-1 for error)

5.10.3.9 ssize_t Graph_getNodes (const Graph * g, char *** nodes)

provide list of nodes of a graph

Parameters

g	Graph to inspect
nodes	input parameter to store array of nodes' names

Returns

number of nodes found (-1 for error)

5.10.3.10 bool Graph_isAdjacent (const Graph * g, const char * from, const char * to)

Checks if two nodes are adjacent.

Parameters

g	Graph to inspect
from	Name of source node
to	Name of destination node

Returns

True if nodes are adjacent, false otherwise

5.10.3.11 void Graph_print (const Graph * g)

Prints graph to stdout.

Parameters

g	Graph to print

5.11 graph/Graph.h File Reference

```
#include <stdbool.h>
#include <unistd.h>
```

Include dependency graph for Graph.h: This graph shows which files directly or indirectly include this file:

Typedefs

- typedef struct graph Graph

Functions

- Graph * Graph_create (void)
  Create an empty graph structure.
- bool Graph_addNode (Graph *g, const char *name)
  Adds a node to the graph (does not add duplicates)
- bool Graph_addEdge (Graph *g, const char *from, const char *to, double weight)
  Adds an edge to the graph (does not add duplicates)
- bool Graph_isAdjacent (const Graph *g, const char *from, const char *to)
  Checks if two nodes are adjacent.
- ssize_t Graph_getNodes (const Graph *g, char ***nodes)
  provide list of nodes of a graph
- ssize_t Graph_getNeighbors (const Graph *g, const char *name, char ***neighbors)
  provide list of neighbor's names for a given node
- double Graph_getEdgeWeight (const Graph *g, const char *from, const char *to)
  Provide edge weight between two nodes.
- void Graph_deleteNode (Graph *g, const char *name)
  Remove a node from the graph.
- void Graph_deleteEdge (Graph *g, const char *from, const char *to)
  Remove an edge from the graph.
- void Graph_print (const Graph *g)
  Prints graph to stdout.
- void Graph_disassemble (Graph *g)
  Destroy the graph scaffolding without affecting the underlying data.

5.11.1 Typedef Documentation

5.11.1.1 typedef struct graph Graph

5.11.2 Function Documentation

5.11.2.1 bool Graph_addEdge (Graph * *g*, const char * *from*, const char * *to*, double *weight*)

Adds an edge to the graph (does not add duplicates)

Parameters

g	Graph to modify
from	Name of source node
to	Name of destination node
weight	Cost of the edge

Returns

true for successful add

5.11.2.2 bool Graph_addNode (Graph * *g*, const char * *name*)

Adds a node to the graph (does not add duplicates)

Parameters

g	Graph to modify
name	Name of new node

Returns

true for successful add

5.11.2.3 Graph* Graph_create (void)

Create an empty graph structure.

Returns

the graph structure, or NULL on error

5.11.2.4 void Graph_deleteEdge (Graph * *g*, const char * *from*, const char * *to*)

Remove an edge from the graph.

Parameters

g	Graph to alter
from	Starting node of edge
to	Ending node of edge

5.11.2.5 void Graph_deleteNode (Graph * *g*, const char * *name*)

Remove a node from the graph.

Parameters

g	Graph to alter
name	Name of node to remove

5.11.2.6 void Graph_disassemble (Graph * *g*)

Destroy the graph scaffolding without affecting the underlying data.

Parameters

g	Graph to disassemble

5.11.2.7 double Graph_getEdgeWeight (const Graph * g, const char * from, const char * to)

Provide edge weight between two nodes.

Parameters

g	Graph to inspect
from	Starting node
to	neighbor node

Returns

weight of edge (NAN if edge does not exist)

5.11.2.8 ssize_t Graph_getNeighbors (const Graph * g, const char * name, char *** neighbors)

provide list of neighbor's names for a given node

Parameters

g	Graph to inspect
name	node's name to find neighbors for
neighbors	input parameter to store array of neighbors' names

Returns

number of neighbors found (-1 for error)

5.11.2.9 ssize_t Graph_getNodes (const Graph * g, char *** nodes)

provide list of nodes of a graph

Parameters

g	Graph to inspect
nodes	input parameter to store array of nodes' names

Returns

number of nodes found (-1 for error)

5.11.2.10 bool Graph_isAdjacent (const Graph * g, const char * from, const char * to)

Checks if two nodes are adjacent.

Parameters

g	Graph to inspect
from	Name of source node
to	Name of destination node

Returns

True if nodes are adjacent, false otherwise

5.11.2.11 void Graph_print (const Graph * g)

Prints graph to stdout.

Parameters

g	Graph to print

5.12 graph/GraphSerializer.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Graph.h"
#include "GraphSerializer.h"
```

Include dependency graph for GraphSerializer.c:

Macros

- #define _GNU_SOURCE

Functions

- void GraphSerializer_toStdout (const Graph *g)

  Print out serialized version of graph.

- Graph * GraphSerializer_fromFile (FILE *fp, char ***mazeFromFile, size_t *maxLength, size_t *lineCount, char flags)

  Deserializes Graph object from input.

- void GraphSerializer_fromMaze (Graph **g, char **maze, size_t *maxLength, size_t *lineCount, char flags)

  Taking a 2d array will make nodes from each available space.

5.12.1 Macro Definition Documentation

5.12.1.1 #define _GNU_SOURCE

5.12.2 Function Documentation

5.12.2.1 Graph* GraphSerializer_fromFile (FILE * *fp*, char *** *mazeFromFile*, size_t * *maxLineLength*, size_t * *lineCount*, char *flags*)

Deserializes Graph object from input.

Parameters

fp	FILE * to read serialized Graph from
mazeFromFile	char *** to make the map read be available in main
maxLineLength	Tracking for mazeFromFile size (X axis)
lineCount	Tracking for mazeFromFile size (Y axis)
flags	Used to keep track of CLI arguments

Returns

Graph object (NULL on error)

5.12.2.2 void GraphSerializer_fromMaze (Graph ** *g*, char ** *maze*, size_t * *maxLength*, size_t * *lineCount*, char *flags*)

Taking a 2d array will make nodes from each available space.

Parameters

g	Graph ** to modify and produce graph from maze
mazeFromFile	char *** to parse and make the graph
maxLineLength	Tracking for mazeFromFile size (X axis)
lineCount	Tracking for mazeFromFile size (Y axis)
flags	Used to keep track of CLI arguments

5.12.2.3 void GraphSerializer_toStdout (const Graph * *g*)

Print out serialized version of graph.

Parameters

g	Graph object to print

5.13 graph/GraphSerializer.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void GraphSerializer_toStdout (const Graph *g)

  Print out serialized version of graph.
- Graph * GraphSerializer_fromFile (FILE *fp, char ***mazeFromFile, size_t *maxLineLength, size_t *lineCount, char flags)

  Deserializes Graph object from input.
- void GraphSerializer_fromMaze (Graph **g, char **maze, size_t *maxLength, size_t *lineCount, char flags)

  Taking a 2d array will make nodes from each available space.

5.13.1 Function Documentation

5.13.1.1 Graph* GraphSerializer_fromFile (FILE * *fp*, char *** *mazeFromFile*, size_t * *maxLineLength*, size_t * *lineCount*, char *flags*)

Deserializes Graph object from input.

Parameters

fp	FILE * to read serialized Graph from
mazeFromFile	char *** to make the map read be available in main
maxLineLength	Tracking for mazeFromFile size (X axis)
lineCount	Tracking for mazeFromFile size (Y axis)
flags	Used to keep track of CLI arguments

Returns

Graph object (NULL on error)

5.13.1.2 void GraphSerializer_fromMaze (Graph ** *g*, char ** *maze*, size_t * *maxLength*, size_t * *lineCount*, char *flags*)

Taking a 2d array will make nodes from each available space.

Parameters

g	Graph ** to modify and produce graph from maze
mazeFromFile	char *** to parse and make the graph
maxLineLength	Tracking for mazeFromFile size (X axis)
lineCount	Tracking for mazeFromFile size (Y axis)
flags	Used to keep track of CLI arguments

5.13.1.3 void GraphSerializer_toStdout (const Graph * g)

Print out serialized version of graph.

Parameters

g	Graph object to print

5.14 graph/test/test01.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "Graph.h"
#include "GraphSerializer.h"
```

Include dependency graph for test01.c:

Functions

- int main (void)

5.14.1 Function Documentation

5.14.1.1 int main (void)

5.15 graph/test/test02.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "Graph.h"
#include "GraphSerializer.h"
```

Include dependency graph for test02.c:

Functions

- int main (void)

5.15.1 Function Documentation

5.15.1.1 int main (void)

5.16 graph/test/test03.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "Graph.h"
#include "GraphSerializer.h"
```

Include dependency graph for test03.c:

Functions

- int main (void)

5.16.1 Function Documentation

5.16.1.1 int main (void)

5.17 map/hashtable.c File Reference

```
#include "map.h"
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/md5.h>
```

Include dependency graph for hashtable.c:

Data Structures

- struct entry
- struct _map

Macros

- #define _XOPEN_SOURCE 500

Functions

- map * map_create (void)

  Creates an empty map structure.
- bool map_insert (map *m, const char *key, double value)

  Inserts new key and value into map.
- bool map_exists (map *m, const char *key)

  Checks if key exists in map.
- double map_lookup (map *m, const char *key)

  Returns Value of specified key.
- void map_destroy (map *m)

  Breaks down map and frees memory.
- void hashtable_print (map *m)

5.17.1 Macro Definition Documentation

5.17.1.1 #define _XOPEN_SOURCE 500

5.17.2 Function Documentation

5.17.2.1 void hashtable_print (map * m)

5.17.2.2 map* map_create (void)

Creates an empty map structure.

Returns

Pointer to new map in memory

5.17.2.3 void map_destroy (map * m)

Breaks down map and frees memory.

Parameters

m	Map to destroy

5.17.2.4 bool map_exists (map * m, const char * key)

Checks if key exists in map.

Parameters

m	Map to inspect
key	Key value to look for in map

Returns

True if found successfully, False if not found

5.17.2.5 bool map_insert (map * m, const char * key, double value)

Inserts new key and value into map.

Parameters

m	Map to insert into
key	Key value to add to map
value	Value for specified key

Returns

True if added successfully, False if failed

5.17.2.6 double map_lookup (map * *m*, const char * *key*)

Returns Value of specified key.

Parameters

m	Map to inspect
key	Key value to look for in map

Returns

Value at key specified

5.18 map/map.c File Reference

```
#include "map.h"
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/md5.h>
```

Include dependency graph for map.c:

Data Structures

- struct entry
- struct _map

Macros

- #define _XOPEN_SOURCE 500

Functions

- map * map_create (void)
  Creates an empty map structure.
- bool map_insert (map *m, const char *key, double value)
  Inserts new key and value into map.
- bool map_exists (map *m, const char *key)
  Checks if key exists in map.
- double map_lookup (map *m, const char *key)
  Returns Value of specified key.
- void map_destroy (map *m)
  Breaks down map and frees memory.
- void hashtable_print (map *m)

5.18.1 Macro Definition Documentation

5.18.1.1 #define _XOPEN_SOURCE 500

5.18.2 Function Documentation

5.18.2.1 void hashtable_print (map * m)

5.18.2.2 map* map_create (void)

Creates an empty map structure.

Returns

Pointer to new map in memory

5.18.2.3 void map_destroy (map * m)

Breaks down map and frees memory.

Parameters

m	Map to destroy

5.18.2.4 bool map_exists (map * m, const char * key)

Checks if key exists in map.

Parameters

m	Map to inspect
key	Key value to look for in map

Returns

True if found successfully, False if not found

5.18.2.5 bool map_insert (map * m, const char * key, double value)

Inserts new key and value into map.

Parameters

m	Map to insert into
key	Key value to add to map
value	Value for specified key

Returns

True if added successfully, False if failed

5.18.2.6 double map_lookup (map * *m,* const char * *key*)

Returns Value of specified key.

Parameters

m	Map to inspect
key	Key value to look for in map

Returns

Value at key specified

5.19 map/map.h File Reference

`#include <stdbool.h>`
Include dependency graph for map.h: This graph shows which files directly or indirectly include this file:

Typedefs

- typedef struct _map map

Functions

- map * map_create (void)
  Creates an empty map structure.
- bool map_insert (map *m, const char *key, double value)
  Inserts new key and value into map.
- bool map_exists (map *m, const char *key)
  Checks if key exists in map.
- double map_lookup (map *m, const char *key)
  Returns Value of specified key.
- void map_destroy (map *m)
  Breaks down map and frees memory.

5.19.1 Typedef Documentation

5.19.1.1 typedef struct _map map

5.19.2 Function Documentation

5.19.2.1 map* map_create (void)

Creates an empty map structure.

Returns

Pointer to new map in memory

5.19.2.2 void map_destroy (map * m)

Breaks down map and frees memory.

Parameters

m	Map to destroy

5.19.2.3 bool map_exists (map * m, const char * key)

Checks if key exists in map.

Parameters

m	Map to inspect
key	Key value to look for in map

Returns

True if found successfully, False if not found

5.19.2.4 bool map_insert (map * m, const char * key, double value)

Inserts new key and value into map.

Parameters

m	Map to insert into
key	Key value to add to map
value	Value for specified key

Returns

True if added successfully, False if failed

5.19.2.5 double map_lookup (map * m, const char * key)

Returns Value of specified key.

Parameters

m	Map to inspect
key	Key value to look for in map

Returns

Value at key specified

5.20 map/vmap.c File Reference

```
#include "vmap.h"
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/md5.h>
```

Include dependency graph for vmap.c:

Data Structures

- struct entry
- struct _vmap

Macros

- #define _XOPEN_SOURCE 500

Functions

- vmap ∗ vmap_create (void)

  Creates an empty vmap structure.
- bool vmap_insert (vmap ∗m, const char ∗key, void ∗value)

  Inserts new key and value into vmap.
- bool vmap_exists (vmap ∗m, const char ∗key)

  Checks if key exists in vmap.
- void ∗ vmap_lookup (vmap ∗m, const char ∗key)

  Returns Value of specified key.
- void vmap_destroy (vmap ∗m)

  Breaks down vmap and frees memory.

5.20.1 Macro Definition Documentation

5.20.1.1 #define _XOPEN_SOURCE 500

5.20.2 Function Documentation

5.20.2.1 vmap∗ vmap_create (void)

Creates an empty vmap structure.

Returns

Pointer to new vmap in memory

5.20.2.2 void vmap_destroy (vmap ∗ *m*)

Breaks down vmap and frees memory.

Parameters

m	VMap to destroy

5.20.2.3 bool vmap_exists (vmap * m, const char * key)

Checks if key exists in vmap.

Parameters

m	VMap to inspect
key	Key value to look for in vmap

Returns

True if found successfully, False if not found

5.20.2.4 bool vmap_insert (vmap * m, const char * key, void * value)

Inserts new key and value into vmap.

Parameters

m	VMap to insert into
key	Key value to add to vmap
value	Value for specified key

Returns

True if added successfully, False if failed

5.20.2.5 void* vmap_lookup (vmap * m, const char * key)

Returns Value of specified key.

Parameters

m	VMap to inspect
key	Key value to look for in vmap

Returns

Value at key specified

5.21 map/vmap.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for vmap.h: This graph shows which files directly or indirectly include this file:

Typedefs

- typedef struct _vmap vmap

Functions

- vmap * vmap_create (void)

  Creates an empty vmap structure.
- bool vmap_insert (vmap *m, const char *key, void *value)

  Inserts new key and value into vmap.
- bool vmap_exists (vmap *m, const char *key)

  Checks if key exists in vmap.
- void * vmap_lookup (vmap *m, const char *key)

  Returns Value of specified key.
- void vmap_destroy (vmap *m)

  Breaks down vmap and frees memory.

5.21.1 Typedef Documentation

5.21.1.1 typedef struct _vmap vmap

5.21.2 Function Documentation

5.21.2.1 vmap* vmap_create (void)

Creates an empty vmap structure.

Returns

Pointer to new vmap in memory

5.21.2.2 void vmap_destroy (vmap * m)

Breaks down vmap and frees memory.

Parameters

m	VMap to destroy

5.21.2.3 bool vmap_exists (vmap * m, const char * key)

Checks if key exists in vmap.

Parameters

m	VMap to inspect
key	Key value to look for in vmap

Returns

True if found successfully, False if not found

5.21.2.4 bool vmap_insert (vmap * m, const char * key, void * value)

Inserts new key and value into vmap.

Parameters

m	VMap to insert into
key	Key value to add to vmap
value	Value for specified key

Returns

True if added successfully, False if failed

5.21.2.5 void* vmap_lookup (vmap * m, const char * key)

Returns Value of specified key.

Parameters

m	VMap to inspect
key	Key value to look for in vmap

Returns

Value at key specified

5.22 maze.c File Reference

```
#include <ctype.h>
```

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Dijkstra.h"
#include "graph/Graph.h"
#include "graph/GraphSerializer.h"
```

Include dependency graph for maze.c:

Functions

- void Maze_generate (char ***customMaze, size_t maxLineLength, size_t lineCount)

  Generates 2d array of a maze.
- int main (int argc, char *argv[ ])

5.22.1 Function Documentation

5.22.1.1 int main (int *argc,* char * *argv[ ]*)

5.22.1.2 void Maze_generate (char *** *customMaze,* size_t *maxLineLength,* size_t *lineCount*)

Generates 2d array of a maze.

Parameters

customMaze	char *** to modify into a maze
maxLineLength	Size passed in on CLI
lineCount	Size passed in on CLI

5.23 README.md File Reference

5.24 tests/door-water.txt File Reference

5.25 tests/door.txt File Reference

5.26 tests/map01.txt File Reference

5.27 tests/map02.txt File Reference

5.28 tests/map03.txt File Reference

5.29 tests/map04.txt File Reference

5.30 tests/map05.txt File Reference

5.31 tests/water.txt File Reference

Index

_GNU_SOURCE
    GraphSerializer.c, 31
_XOPEN_SOURCE
    adjlist.c, 19
    Graph.c, 23
    hashtable.c, 35
    map.c, 37
    vmap.c, 40
_item, 7
    data, 7
    priority, 7
_map, 7
    capacity, 8
    data, 8
    size, 8
_pqueue, 8
    capacity, 8
    cmp, 8
    data, 8
    size, 8
_vmap, 8
    capacity, 9
    data, 9
    size, 9

adjlist.c
    _XOPEN_SOURCE, 19
    edge_, 19
    Graph_addEdge, 19
    Graph_addNode, 20
    Graph_create, 20
    Graph_deleteEdge, 20
    Graph_deleteNode, 20
    Graph_disassemble, 20
    Graph_getEdgeWeight, 21
    Graph_getNeighbors, 21
    Graph_getNodes, 21
    Graph_isAdjacent, 22
    Graph_print, 22
    node_, 19

capacity
    _map, 8
    _pqueue, 8
    _vmap, 9
cmp
    _pqueue, 8

data
    _item, 7
    _map, 8
    _pqueue, 8
    _vmap, 9
Dijkstra.c, 13
Dijkstra.h, 13
    Dijkstra_path, 13
    Dijkstra_solveMaze, 14
Dijkstra_path
    Dijkstra.h, 13
Dijkstra_solveMaze
    Dijkstra.h, 14
dpqueue/driver.c, 14
    heap_print, 14
    main, 14
dpqueue/heap.c, 15
dpqueue/pqueue.c, 16
dpqueue/pqueue.h, 17

edge_, 9
    adjlist.c, 19
    Graph.c, 23
    next, 9
    to, 9
    weight, 9
edges
    node_, 11
entry, 9
    key, 10
    next, 10
    value, 10

Graph
    Graph.h, 27
graph, 10
    root, 10
Graph.c
    _XOPEN_SOURCE, 23
    edge_, 23
    Graph_addEdge, 23
    Graph_addNode, 24
    Graph_create, 24
    Graph_deleteEdge, 24
    Graph_deleteNode, 24
    Graph_disassemble, 25
    Graph_getEdgeWeight, 25
    Graph_getNeighbors, 25
    Graph_getNodes, 25
    Graph_isAdjacent, 26
    Graph_print, 26
    node_, 23

Graph.h
  Graph, 27
  Graph_addEdge, 27
  Graph_addNode, 27
  Graph_create, 28
  Graph_deleteEdge, 28
  Graph_deleteNode, 28
  Graph_disassemble, 28
  Graph_getEdgeWeight, 29
  Graph_getNeighbors, 29
  Graph_getNodes, 29
  Graph_isAdjacent, 29
  Graph_print, 30
graph/Graph.c, 22
graph/Graph.h, 26
graph/GraphSerializer.c, 30
graph/GraphSerializer.h, 32
graph/adjlist.c, 18
graph/driver.c, 14
  main, 15
graph/test/test01.c, 33
graph/test/test02.c, 33
graph/test/test03.c, 33
Graph_addEdge
  adjlist.c, 19
  Graph.c, 23
  Graph.h, 27
Graph_addNode
  adjlist.c, 20
  Graph.c, 24
  Graph.h, 27
Graph_create
  adjlist.c, 20
  Graph.c, 24
  Graph.h, 28
Graph_deleteEdge
  adjlist.c, 20
  Graph.c, 24
  Graph.h, 28
Graph_deleteNode
  adjlist.c, 20
  Graph.c, 24
  Graph.h, 28
Graph_disassemble
  adjlist.c, 20
  Graph.c, 25
  Graph.h, 28
Graph_getEdgeWeight
  adjlist.c, 21
  Graph.c, 25
  Graph.h, 29
Graph_getNeighbors
  adjlist.c, 21
  Graph.c, 25
  Graph.h, 29
Graph_getNodes
  adjlist.c, 21
  Graph.c, 25
  Graph.h, 29
Graph_isAdjacent
  adjlist.c, 22
  Graph.c, 26
  Graph.h, 29
Graph_print
  adjlist.c, 22
  Graph.c, 26
  Graph.h, 30
GraphSerializer.c
  _GNU_SOURCE, 31
  GraphSerializer_fromFile, 31
  GraphSerializer_fromMaze, 31
  GraphSerializer_toStdout, 31
GraphSerializer.h
  GraphSerializer_fromFile, 32
  GraphSerializer_fromMaze, 32
  GraphSerializer_toStdout, 32
GraphSerializer_fromFile
  GraphSerializer.c, 31
  GraphSerializer.h, 32
GraphSerializer_fromMaze
  GraphSerializer.c, 31
  GraphSerializer.h, 32
GraphSerializer_toStdout
  GraphSerializer.c, 31
  GraphSerializer.h, 32

hashtable.c
  _XOPEN_SOURCE, 35
  hashtable_print, 35
  map_create, 35
  map_destroy, 35
  map_exists, 35
  map_insert, 35
  map_lookup, 36
hashtable_print
  hashtable.c, 35
  map.c, 37
  map/driver.c, 15
heap.c
  heap_print, 16
  pqueue_create, 16
  pqueue_dequeue, 16
  pqueue_destroy, 16
  pqueue_enqueue, 16
  pqueue_reprioritize, 16
  pqueue_search, 16
  pqueue_size, 16
heap_print
  dpqueue/driver.c, 14
  heap.c, 16
  pqueue.c, 17

key
  entry, 10

main
  dpqueue/driver.c, 14

graph/driver.c, 15
map/driver.c, 15
maze.c, 44
test01.c, 33
test02.c, 33
test03.c, 34
map
map.h, 38
map.c
_XOPEN_SOURCE, 37
hashtable_print, 37
map_create, 37
map_destroy, 37
map_exists, 37
map_insert, 37
map_lookup, 38
map.h
map, 38
map_create, 38
map_destroy, 38
map_exists, 39
map_insert, 39
map_lookup, 39
map/driver.c, 15
hashtable_print, 15
main, 15
map/hashtable.c, 34
map/map.c, 36
map/map.h, 38
map/vmap.c, 40
map/vmap.h, 42
map_create
hashtable.c, 35
map.c, 37
map.h, 38
map_destroy
hashtable.c, 35
map.c, 37
map.h, 38
map_exists
hashtable.c, 35
map.c, 37
map.h, 39
map_insert
hashtable.c, 35
map.c, 37
map.h, 39
map_lookup
hashtable.c, 36
map.c, 38
map.h, 39
maze.c, 43
main, 44
Maze_generate, 44
Maze_generate
maze.c, 44

name
node_, 11
next
edge_, 9
entry, 10
node_, 11
node_, 10
adjlist.c, 19
edges, 11
Graph.c, 23
name, 11
next, 11

pqueue
pqueue.h, 18
pqueue.c
heap_print, 17
pqueue_create, 17
pqueue_dequeue, 17
pqueue_destroy, 17
pqueue_enqueue, 17
pqueue_reprioritize, 17
pqueue_search, 17
pqueue_size, 17
pqueue.h
pqueue, 18
pqueue_create, 18
pqueue_dequeue, 18
pqueue_destroy, 18
pqueue_enqueue, 18
pqueue_reprioritize, 18
pqueue_search, 18
pqueue_size, 18
pqueue_create
heap.c, 16
pqueue.c, 17
pqueue.h, 18
pqueue_dequeue
heap.c, 16
pqueue.c, 17
pqueue.h, 18
pqueue_destroy
heap.c, 16
pqueue.c, 17
pqueue.h, 18
pqueue_enqueue
heap.c, 16
pqueue.c, 17
pqueue.h, 18
pqueue_reprioritize
heap.c, 16
pqueue.c, 17
pqueue.h, 18
pqueue_search
heap.c, 16
pqueue.c, 17
pqueue.h, 18
pqueue_size
heap.c, 16
pqueue.c, 17
pqueue.h, 18

priority
_item, 7

README.md, 44
root
graph, 10

size
_map, 8
_pqueue, 8
_vmap, 9

test01.c
main, 33
test02.c
main, 33
test03.c
main, 34
tests/door-water.txt, 44
tests/door.txt, 44
tests/map01.txt, 44
tests/map02.txt, 44
tests/map03.txt, 44
tests/map04.txt, 45
tests/map05.txt, 45
tests/water.txt, 45
to
edge_, 9

value
entry, 10
vmap
vmap.h, 42
vmap.c
_XOPEN_SOURCE, 40
vmap_create, 40
vmap_destroy, 40
vmap_exists, 41
vmap_insert, 41
vmap_lookup, 41
vmap.h
vmap, 42
vmap_create, 42
vmap_destroy, 42
vmap_exists, 42
vmap_insert, 43
vmap_lookup, 43
vmap_create
vmap.c, 40
vmap.h, 42
vmap_destroy
vmap.c, 40
vmap.h, 42
vmap_exists
vmap.c, 41
vmap.h, 42
vmap_insert
vmap.c, 41
vmap.h, 43
vmap_lookup
vmap.c, 41
vmap.h, 43

weight
edge_, 9