Graph

beta 1.1

Generated by Doxygen 1.10.0

1 Data Structure Index **1**

1.1 Data Structures . . . 1

2 File Index **3**

2.1 File List . . . 3

3 Data Structure Documentation **5**

3.1 edge Struct Reference . . . 5

3.1.1 Detailed Description . . . 5

3.1.2 Field Documentation . . . 5

3.1.2.1 end_vertex . . . 5

3.1.2.2 length . . . 6

3.1.2.3 start_vertex . . . 6

3.2 graph Struct Reference . . . 6

3.2.1 Detailed Description . . . 6

3.2.2 Field Documentation . . . 7

3.2.2.1 edges . . . 7

3.2.2.2 edges_amount . . . 7

3.2.2.3 vertices . . . 7

3.2.2.4 vertices_amount . . . 7

3.3 matrix Struct Reference . . . 7

3.3.1 Detailed Description . . . 7

3.3.2 Field Documentation . . . 8

3.3.2.1 columns . . . 8

3.3.2.2 rows . . . 8

3.3.2.3 values . . . 8

4 File Documentation **9**

4.1 D:/files from internet/important/learning/github/c-modules/Graph/code/inc/graph.h File Reference . . . 9

4.1.1 Macro Definition Documentation . . . 12

4.1.1.1 _GRAPH_EMPTY__ . . . 12

4.1.1.2 _GRAPH_EXIST__ . . . 12

4.1.1.3 _GRAPH_FORBIDDEN_SEPARATORS__ . . . 12

4.1.1.4 _GRAPH_INCORRECT_ARG__ . . . 12

4.1.1.5 _GRAPH_MEM__ . . . 12

4.1.1.6 _GRAPH_NOT_FOUND__ . . . 12

4.1.1.7 _GRAPH_OK__ . . . 13

4.1.1.8 _GRAPH_OS_ERROR__ . . . 13

4.1.1.9 _STRING__ . . . 13

4.1.2 Typedef Documentation . . . 13

4.1.2.1 graph_error_t . . . 13

4.1.3 Function Documentation . . . 13

4.1.3.1 graph_add_edge() . . . 13

4.1.3.2 graph_add_vertex() . . . 14
4.1.3.3 graph_adjacency_list_fill() . . . 15
4.1.3.4 graph_adjacency_list_size() . . . 16
4.1.3.5 graph_adjacency_matrix_create() . . . 16
4.1.3.6 graph_adjacency_matrix_free() . . . 17
4.1.3.7 graph_adjacency_matrix_show() . . . 17
4.1.3.8 graph_adjacency_matrix_to_dot() . . . 18
4.1.3.9 graph_delete_edge() . . . 19
4.1.3.10 graph_delete_vertex() . . . 20
4.1.3.11 graph_dfs() . . . 20
4.1.3.12 graph_floyd_warshall() . . . 21
4.1.3.13 graph_free() . . . 21
4.1.3.14 graph_has_edge() . . . 22
4.1.3.15 graph_has_vertex() . . . 23
4.1.3.16 graph_initialize() . . . 24
4.1.3.17 graph_is_empty() . . . 24
4.1.3.18 graph_show() . . . 25
4.1.3.19 graph_to_dot() . . . 26
4.2 graph.h . . . 27
4.3 D:/files from internet/important/learning/github/c-modules/Graph/code/src/graph.c File Reference . . . 28
4.3.1 Function Documentation . . . 29
4.3.1.1 graph_add_edge() . . . 29
4.3.1.2 graph_add_vertex() . . . 30
4.3.1.3 graph_adjacency_list_fill() . . . 31
4.3.1.4 graph_adjacency_list_size() . . . 31
4.3.1.5 graph_adjacency_matrix_create() . . . 32
4.3.1.6 graph_adjacency_matrix_free() . . . 33
4.3.1.7 graph_adjacency_matrix_show() . . . 33
4.3.1.8 graph_adjacency_matrix_to_dot() . . . 34
4.3.1.9 graph_delete_edge() . . . 35
4.3.1.10 graph_delete_vertex() . . . 36
4.3.1.11 graph_dfs() . . . 36
4.3.1.12 graph_floyd_warshall() . . . 37
4.3.1.13 graph_free() . . . 37
4.3.1.14 graph_has_edge() . . . 37
4.3.1.15 graph_has_vertex() . . . 39
4.3.1.16 graph_initialize() . . . 40
4.3.1.17 graph_is_empty() . . . 41
4.3.1.18 graph_show() . . . 42
4.3.1.19 graph_to_dot() . . . 43
4.4 graph.c . . . 44

Index 53

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

edge
Edge of graph 5
graph
Graph 6
matrix
Matrix 7

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

D:/files from internet/important/learning/github/c-modules/Graph/code/inc/ **graph.h** 9
D:/files from internet/important/learning/github/c-modules/Graph/code/src/ **graph.c** 28

Chapter 3

Data Structure Documentation

3.1 edge Struct Reference

Edge of graph.

`#include <graph.h>`

Data Fields

- char **start_vertex** [**_STRING__**+1]
- char **end_vertex** [**_STRING__**+1]
- size_t **length**

3.1.1 Detailed Description

Edge of graph.

Parameters

start_vertex	Name of start vertex
end_vertex	Name of end vertex
length	Length of edge

Definition at line **76** of file **graph.h**.

3.1.2 Field Documentation

3.1.2.1 end_vertex

`char end_vertex[ _STRING__+1]`

Definition at line **79** of file **graph.h**.

3.1.2.2 length

```
size_t length
```

Definition at line **80** of file **graph.h**.

3.1.2.3 start_vertex

```
char start_vertex[ _STRING__+1]
```

Definition at line **78** of file **graph.h**.

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

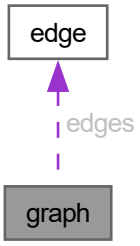
- D:/files from internet/important/learning/github/c-modules/Graph/code/inc/ **graph.h**

3.2 graph Struct Reference

Graph.

```
#include <graph.h>
```

Collaboration diagram for graph:



Data Fields

- char ** **vertices**
- size_t **vertices_amount**
- struct **edge** * **edges**
- size_t **edges_amount**

3.2.1 Detailed Description

Graph.

Parameters

vertices	Dynamic array of vertices names
vertices_amount	Length of vertices array
edges	Dynamic array of edges
edges_amount	Length of edges array

Definition at line **91** of file **graph.h**.

3.2.2 Field Documentation

3.2.2.1 edges

```
struct  edge* edges
```

Definition at line **95** of file **graph.h**.

3.2.2.2 edges_amount

```
size_t edges_amount
```

Definition at line **96** of file **graph.h**.

3.2.2.3 vertices

```
char** vertices
```

Definition at line **93** of file **graph.h**.

3.2.2.4 vertices_amount

```
size_t vertices_amount
```

Definition at line **94** of file **graph.h**.

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

- D:/files from internet/important/learning/github/c-modules/Graph/code/inc/ **graph.h**

3.3 matrix Struct Reference

Matrix.

```
#include <graph.h>
```

Data Fields

- int ** **values**
- size_t **rows**
- size_t **columns**

3.3.1 Detailed Description

Matrix.

Parameters

values	Matrix values
rows	Amount of rows in matrix
columns	Amount of columns in matrix

Definition at line **62** of file **graph.h**.

3.3.2 Field Documentation

3.3.2.1 columns

```
size_t columns
```

Definition at line **66** of file **graph.h**.

3.3.2.2 rows

```
size_t rows
```

Definition at line **65** of file **graph.h**.

3.3.2.3 values

```
int** values
```

Definition at line **64** of file **graph.h**.

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

- D:/files from internet/important/learning/github/c-modules/Graph/code/inc/ **graph.h**

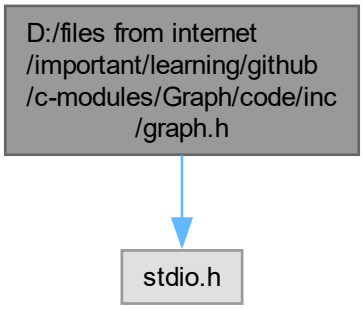
Chapter 4

File Documentation

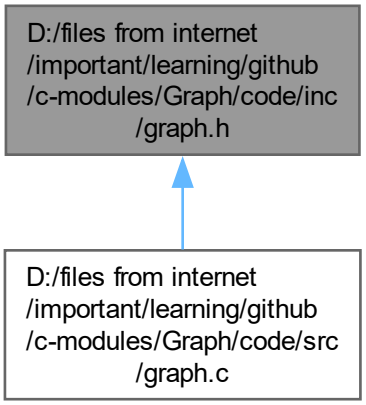
4.1 D:/files from internet/important/learning/github/c-modules/↩ Graph/code/inc/graph.h File Reference

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

Include dependency graph for graph.h:



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



Data Structures

- struct **matrix**

  Matrix.
- struct **edge**

  Edge of graph.
- struct **graph**

  Graph.

Macros

- #define **_STRING__** 256
- #define **_GRAPH_FORBIDDEN_SEPARATORS__** "\"\'#%()><{}-/\\|:;,"
- #define **_GRAPH_OK__** 0

  Positive return code.
- #define **_GRAPH_MEM__** -1

  Memory shortage error.
- #define **_GRAPH_INCORRECT_ARG__** -2

  Incorrect arguments in function.
- #define **_GRAPH_EMPTY__** -3

  Graph is empty.
- #define **_GRAPH_NOT_FOUND__** -4

  Object not in graph.
- #define **_GRAPH_EXIST__** -5

  Graph already has object.
- #define **_GRAPH_OS_ERROR__** -6

  Operating system error.

Typedefs

- typedef int **graph_error_t**

  Data type for errors that occur during the operation of functions.

Functions

- void **graph_initialize** (struct **graph** * **graph**)

  Initialization of graph by zero.
- int **graph_is_empty** (const struct **graph** * **graph**)

  Checking for graph emtiness.
- int **graph_has_vertex** (const struct **graph** * **graph**, const char *vertex)

  Checking for the presence of a vertex in the graph.
- int **graph_has_edge** (const struct **graph** * **graph**, const char *start_vertex, const char *end_vertex)

  Checking for the presence of a edge in the graph.
- **graph_error_t** **graph_add_vertex** (struct **graph** * **graph**, const char *vertex)

  Adding a vertex to a graph.
- **graph_error_t** **graph_delete_vertex** (struct **graph** * **graph**, const char *vertex)

  Deleting vertex from graph.
- **graph_error_t** **graph_add_edge** (struct **graph** * **graph**, const char *start_vertex, const char *end_vertex, size_t edge_length)

  Adding an edge to a graph.
- **graph_error_t** **graph_delete_edge** (struct **graph** * **graph**, const char *start_vertex, const char *end_↩ vertex)

  Deleting edge from graph.
- **graph_error_t** **graph_show** (const struct **graph** * **graph**)

  Draw graph using Graphviz and show it.
- **graph_error_t** **graph_to_dot** (const struct **graph** * **graph**, const char *folder, const char *filename)

  Creating a dot file by graph.
- size_t **graph_adjacency_list_size** (const struct **graph** * **graph**, const char *vertex)

  Counting the number of adjacent vertices (the size of the adjacency list)
- **graph_error_t** **graph_adjacency_list_fill** (const struct **graph** * **graph**, const char *vertex, int *adjacency_list)

  Filling in the adjacency list.
- void **graph_dfs** (struct **graph** * **graph**, void(*vertex_processing)(char *vertex_name))

  Graph traversal using a depth-first search algorithm.
- struct **matrix** * **graph_adjacency_matrix_create** (const struct **graph** * **graph**)

  Creating adjacency matrix by graph.
- **graph_error_t** **graph_adjacency_matrix_to_dot** (const struct **graph** * **graph**, const struct **matrix** *adjacency_matrix, const char *folder, const char *filename)

  Creating a dot file of adjacency matrix of graph.
- **graph_error_t** **graph_adjacency_matrix_show** (const struct **graph** * **graph**, const struct **matrix** *adjacency_matrix)

  Draw graph adjacency matrix using Graphviz and show it.
- void **graph_adjacency_matrix_free** (struct **matrix** *adjacency_matrix)

  Free adjacency matrix.
- struct **matrix** * **graph_floyd_warshall** (const struct **graph** * **graph**)

  Finding the shortest distance matrix using the Floyd-Warshall algorithm.
- void **graph_free** (struct **graph** * **graph**)

  Free graph.

4.1.1 Macro Definition Documentation

4.1.1.1 _GRAPH_EMPTY__

```
#define _GRAPH_EMPTY__ -3
```

Graph is empty.

Definition at line **36** of file **graph.h**.

4.1.1.2 _GRAPH_EXIST__

```
#define _GRAPH_EXIST__ -5
```

Graph already has object.

Definition at line **46** of file **graph.h**.

4.1.1.3 _GRAPH_FORBIDDEN_SEPARATORS__

```
#define _GRAPH_FORBIDDEN_SEPARATORS__ "\"\'#%()><{}-/\\|:;,"
```

Forbidden characters for vertex name

Definition at line **16** of file **graph.h**.

4.1.1.4 _GRAPH_INCORRECT_ARG__

```
#define _GRAPH_INCORRECT_ARG__ -2
```

Incorrect arguments in function.

Definition at line **31** of file **graph.h**.

4.1.1.5 _GRAPH_MEM__

```
#define _GRAPH_MEM__ -1
```

Memory shortage error.

Definition at line **26** of file **graph.h**.

4.1.1.6 _GRAPH_NOT_FOUND__

```
#define _GRAPH_NOT_FOUND__ -4
```

Object not in graph.

Definition at line **41** of file **graph.h**.

4.1.1.7 _GRAPH_OK__

```
#define _GRAPH_OK__ 0
```

Positive return code.

Definition at line **21** of file **graph.h**.

4.1.1.8 _GRAPH_OS_ERROR__

```
#define _GRAPH_OS_ERROR__ -6
```

Operating system error.

Definition at line **51** of file **graph.h**.

4.1.1.9 _STRING__

```
#define _STRING__ 256
```

Max length of string

Definition at line **11** of file **graph.h**.

4.1.2 Typedef Documentation

4.1.2.1 graph_error_t

```
typedef int  graph_error_t
```

Data type for errors that occur during the operation of functions.

Definition at line **102** of file **graph.h**.

4.1.3 Function Documentation

4.1.3.1 graph_add_edge()

```
graph_error_t graph_add_edge (
            struct  graph * graph,
            const char * start_vertex,
            const char * end_vertex,
            size_t edge_length )
```

Adding an edge to a graph.

Parameters

in	*graph*	Graph descriptor
in	*start_vertex*	Start vertex name
in	*end_vertex*	End vertex name

Returns

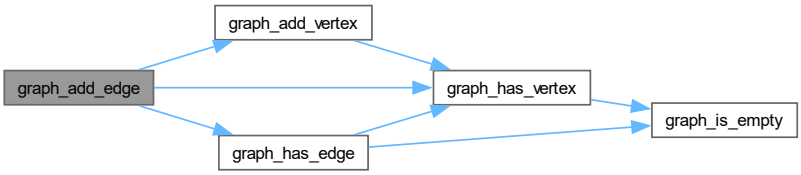
_GRAPH_OK__, _GRAPH_MEM__, _GRAPH_INCORRECT_ARG__, _GRAPH_EXIST__

Note

- You cannot add a copy of an existing edge
- When adding an edge consisting of new vertices, new vertices will be added to the graph

Definition at line **139** of file **graph.c**.

Here is the call graph for this function:



4.1.3.2 graph_add_vertex()

```
graph_error_t graph_add_vertex (
            struct  graph * graph,
            const char * vertex )
```

Adding a vertex to a graph.

Parameters

in	*graph*	Graph descriptor
in	*vertex*	Vertex name

Returns

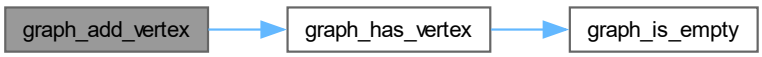
_GRAPH_OK__, _GRAPH_MEM__, _GRAPH_INCORRECT_ARG__, _GRAPH_EXIST__

Note

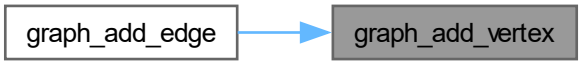
- You cannot add a copy of an existing vertex
- You cannot add a vertex with a name of zero length
- You cannot add a vertex with a name containing special characters - `#%()><{}-/\|:;,` and quotes

Definition at line **58** of file **graph.c**.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.3 graph_adjacency_list_fill()

```
graph_error_t graph_adjacency_list_fill (
            const struct  graph * graph,
            const char * vertex,
            int * adjacency_list )
```

Filling in the adjacency list.

Parameters

in	*graph*	Graph descriptor
in	*vertex*	Vertex name
in	*size*	Adjacency list size
out	*adjacency_list*	Adjacency list descriptor

Returns

`_GRAPH_OK__`, `_GRAPH_INCORRECT_ARG__`

Definition at line **362** of file **graph.c**.

4.1.3.4 graph_adjacency_list_size()

```
size_t graph_adjacency_list_size (
            const struct  graph * graph,
            const char * vertex )
```

Counting the number of adjacent vertices (the size of the adjacency list)

Parameters

in	*graph*	Graph descriptor
in	*vertex*	Vertex name

Returns

The number of adjacent vertices

Note

- If the arguments is incorrect, the function returns 0

Definition at line **346** of file **graph.c**.

Here is the call graph for this function:

4.1.3.5 graph_adjacency_matrix_create()

```
struct  matrix * graph_adjacency_matrix_create (
            const struct  graph * graph )
```

Creating adjacency matrix by graph.

Parameters

in	*graph*	Graph descriptor

Returns

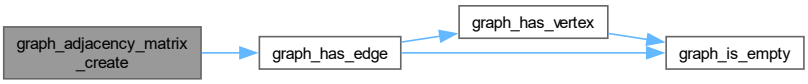
Adjacency matrix descriptor

Note

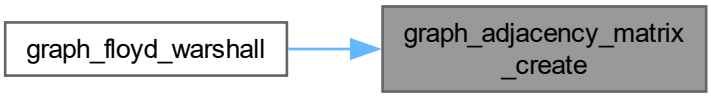
- If errors occur, the function returns NULL

Definition at line **391** of file **graph.c**.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.6 graph_adjacency_matrix_free()

```
void graph_adjacency_matrix_free (
            struct  matrix * adjacency_matrix )
```

Free adjacency matrix.

Parameters

in	*adjacency_matrix*	Adjacency matrix descriptor

Definition at line **671** of file **graph.c**.

4.1.3.7 graph_adjacency_matrix_show()

```
graph_error_t graph_adjacency_matrix_show (
            const struct  graph * graph,
            const struct  matrix * adjacency_matrix )
```

Draw graph adjacency matrix using Graphviz and show it.

Parameters

in	*graph*	Graph descriptor
in	*adjacency_matrix*	Adjacency matrix descriptor

Returns

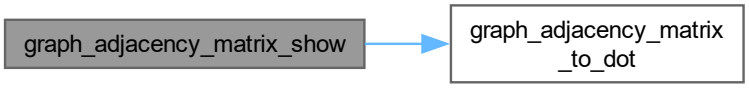
_GRAPH_OK__, _GRAPH_MEM__, _GRAPH_INCORRECT_ARGS__, _GRAPH_OS_ERROR__

Note

- Linux: the graph adjacency matrix is demonstrated using `eog`
- Windows: the graph adjacency matrix is demonstrated using `mspaint`
- The function creates a separate folder for temporary files and deletes it at the end of the work

Definition at line **548** of file **graph.c**.

Here is the call graph for this function:



4.1.3.8 graph_adjacency_matrix_to_dot()

```
graph_error_t graph_adjacency_matrix_to_dot (
            const struct  graph * graph,
            const struct  matrix * adjacency_matrix,
            const char * folder,
            const char * filename )
```

Creating a dot file of adjacency matrix of graph.

Parameters

in	*graph*	Graph descriptor
in	*adjacency_matrix*	Adjacency matrix descriptor
in	*folder*	Folder name
in	*filename*	File name

Returns

_GRAPH_OK__, _GRAPH_INCORRECT_ARG__,_GRAPH_MEM__, _GRAPH_OS_ERROR__

Note

- The pointer to the `folder` string can take the `NULL` value. In this case, the folder will not be created

Definition at line **456** of file **graph.c**.

Here is the caller graph for this function:

4.1.3.9 graph_delete_edge()

```
graph_error_t graph_delete_edge (
            struct  graph * graph,
            const char * start_vertex,
            const char * end_vertex )
```

Deleting edge from graph.

Parameters

in	*graph*	Graph descriptor
in	*start_vertex*	Start vertex name
in	*end_vertex*	End vertex name

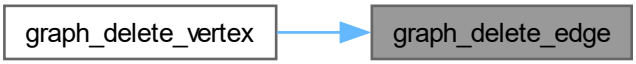
Returns

`_GRAPH_OK__`, `_GRAPH_INCORRECT_ARG__`, `_GRAPH_EMPTY__`, `_GRAPH_NOT_FOUND__`

Definition at line **185** of file **graph.c**.

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.3.10 graph_delete_vertex()

```
graph_error_t graph_delete_vertex (
            struct  graph * graph,
            const char * vertex )
```

Deleting vertex from graph.

Parameters

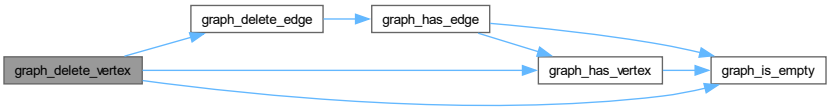
in	*graph*	Graph descriptor
in	*vertex*	Vertex name

Returns

_GRAPH_OK__, _GRAPH_INCORRECT_ARG__, _GRAPH_EMPTY__, _GRAPH_NOT_FOUND__

Definition at line **93** of file **graph.c**.

Here is the call graph for this function:



4.1.3.11 graph_dfs()

```
void graph_dfs (
            struct  graph * graph,
            void(*)(char *vertex_name) vertex_processing )
```

Graph traversal using a depth-first search algorithm.

Parameters

in	*graph*	Graph descriptor
in	*vertex_processing*	Vertex processing function

Note

- If the input arguments are incorrect, the function will not work

Definition at line **618** of file **graph.c**.

4.1.3.12 graph_floyd_warshall()

```
struct  matrix * graph_floyd_warshall (
            const struct  graph * graph )
```

Finding the shortest distance matrix using the Floyd-Warshall algorithm.

Parameters

graph	Graph descriptor

Returns

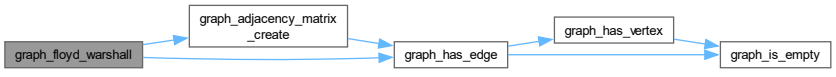
The shortest distance matrix

Note

- If errors occur, the function returns NULL

Definition at line **634** of file **graph.c**.

Here is the call graph for this function:



4.1.3.13 graph_free()

```
void graph_free (
            struct  graph * graph )
```

Free graph.

Parameters

in	*graph*	Graph descriptor

Definition at line **683** of file **graph.c**.

4.1.3.14 graph_has_edge()

```
int graph_has_edge (
            const struct  graph * graph,
            const char * start_vertex,
            const char * end_vertex )
```

Checking for the presence of a edge in the graph.

Parameters

in	*graph*	Graph descriptor
in	*start_vertex*	Start vertex name
in	*end_vertex*	End vertex name

Returns

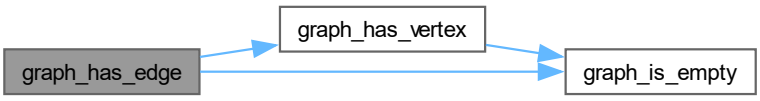
`1` - `True` / `0` - `False`

Note

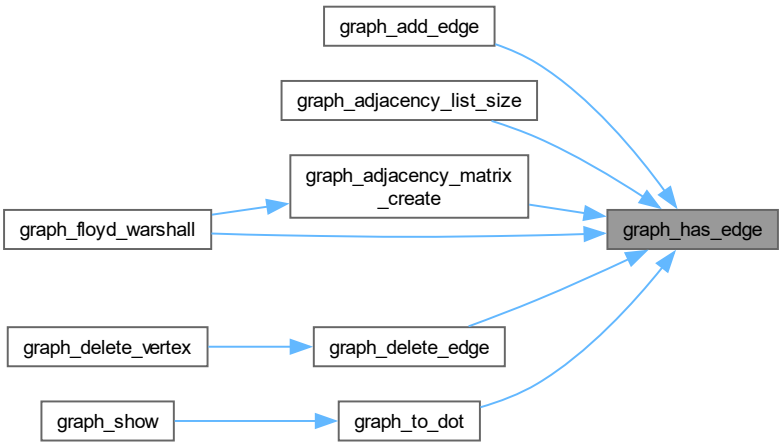
- If incorrect arguments are passed, the function returns `0` (`False`)

Definition at line **34** of file **graph.c**.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.15 graph_has_vertex()

```
int graph_has_vertex (
            const struct  graph * graph,
            const char * vertex )
```

Checking for the presence of a vertex in the graph.

Parameters

in	*graph*	Graph descriptor
in	*vertex*	Vertex name

Returns

`1 - True / 0 - False`

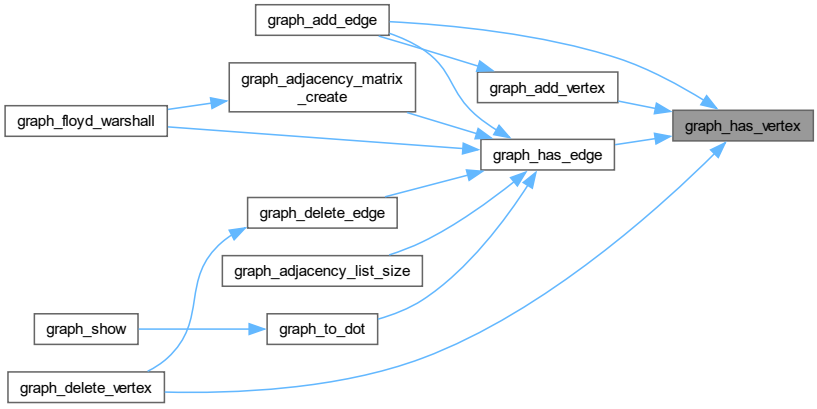
Note

- If incorrect arguments are passed, the function returns `0` (`False`)

Definition at line **17** of file **graph.c**.

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.3.16 graph_initialize()

```
void graph_initialize (
            struct  graph * graph )
```

Initialization of graph by zero.

Parameters

in	*graph*	Graph descriptor

Note

- If the graph descriptor is NULL, the function will not cause a segmentation error

Definition at line **5** of file **graph.c**.

4.1.3.17 graph_is_empty()

```
int graph_is_empty (
            const struct  graph * graph )
```

Checking for graph emtiness.

Parameters

in	*graph*	Graph descriptor

Returns

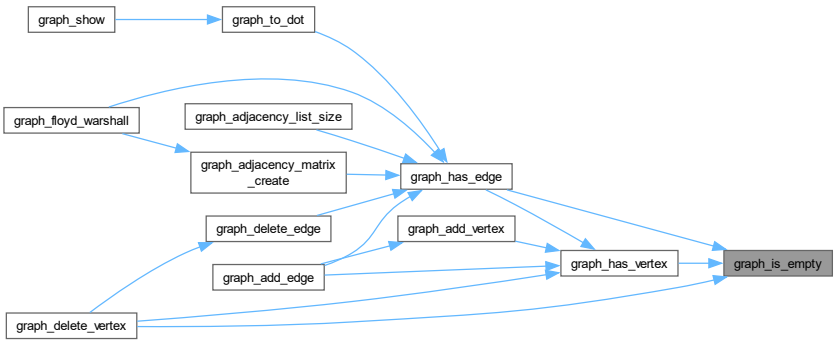
`1 - True / 0 - False`

Note

- If incorrect arguments are passed, the function returns `1` (`True`)

Definition at line **10** of file **graph.c**.

Here is the caller graph for this function:



4.1.3.18 graph_show()

```
graph_error_t graph_show (
            const struct  graph * graph )
```

Draw graph using Graphviz and show it.

Parameters

`in`	*graph*	Graph descriptor

Returns

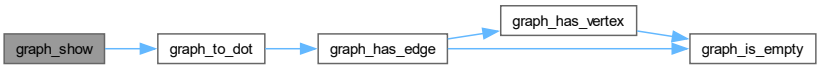
`_GRAPH_OK__`, `_GRAPH_MEM__`, `_GRAPH_INCORRECT_ARGS__`, `_GRAPH_OS_ERROR__`

Note

- Linux: the graph is demonstrated using `eog`
- Windows: the graph is demonstrated using `mspaint`
- The function creates a separate folder for temporary files and deletes it at the end of the work

Definition at line **313** of file **graph.c**.

Here is the call graph for this function:



4.1.3.19 graph_to_dot()

```
graph_error_t graph_to_dot (
            const struct  graph * graph,
            const char * folder,
            const char * filename )
```

Creating a dot file by graph.

Parameters

in	*graph*	Graph descriptor
in	*folder*	Folder name
in	*filename*	File name

Returns

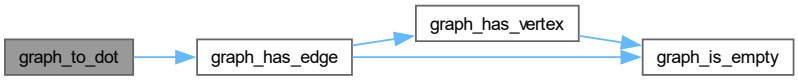
_GRAPH_OK__, _GRAPH_INCORRECT_ARG__,_GRAPH_MEM__, _GRAPH_OS_ERROR__

Note

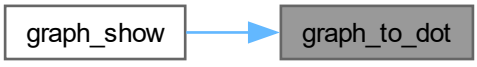
- The pointer to the `folder` string can take the `NULL` value. In this case, the folder will not be created

Definition at line **225** of file **graph.c**.

Here is the call graph for this function:



Here is the caller graph for this function:



4.2 graph.h

Go to the documentation of this file.

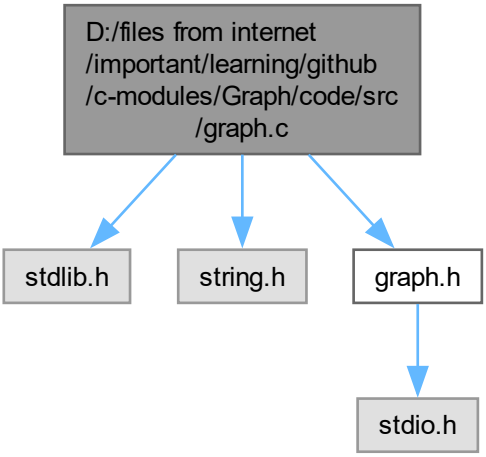
```
00001 #ifndef GRAPH_H__
00002 #define GRAPH_H__
00003
00004 #include <stdio.h>
00005
00006 // Macro
00007
00011 #define _STRING__ 256
00012
00016 #define _GRAPH_FORBIDDEN_SEPARATORS__ "\"\'#%()><{}-/\\|:;,"
00017
00021 #define _GRAPH_OK__ 0
00022
00026 #define _GRAPH_MEM__ -1
00027
00031 #define _GRAPH_INCORRECT_ARG__ -2
00032
00036 #define _GRAPH_EMPTY__ -3
00037
00041 #define _GRAPH_NOT_FOUND__ -4
00042
00046 #define _GRAPH_EXIST__ -5
00047
00051 #define _GRAPH_OS_ERROR__ -6
00052
00053 // Structs and functions
00054
00062 struct matrix
00063 {
00064     int **values;
00065     size_t rows;
00066     size_t columns;
00067 };
00068
00076 struct edge
00077 {
00078     char start_vertex[_STRING__ + 1];
00079     char end_vertex[_STRING__ + 1];
00080     size_t length;
00081 };
00082
00091 struct graph
00092 {
00093     char **vertices;
00094     size_t vertices_amount;
00095     struct edge *edges;
00096     size_t edges_amount;
00097 };
00098
00102 typedef int graph_error_t;
00103
00111 void graph_initialize(struct graph *graph);
00112
00122 int graph_is_empty(const struct graph *graph);
00123
00134 int graph_has_vertex(const struct graph *graph, const char *vertex);
00135
00147 int graph_has_edge(const struct graph *graph, const char *start_vertex, const char *end_vertex);
00148
00161 graph_error_t graph_add_vertex(struct graph *graph, const char *vertex);
00162
00171 graph_error_t graph_delete_vertex(struct graph *graph, const char *vertex);
00172
00185 graph_error_t graph_add_edge(struct graph *graph, const char *start_vertex, const char *end_vertex,
      size_t edge_length);
00186
00196 graph_error_t graph_delete_edge(struct graph *graph, const char *start_vertex, const char
      *end_vertex);
00197
00209 graph_error_t graph_show(const struct graph *graph);
00210
00222 graph_error_t graph_to_dot(const struct graph *graph, const char *folder, const char *filename);
00223
00234 size_t graph_adjacency_list_size(const struct graph *graph, const char *vertex);
00235
00247 graph_error_t graph_adjacency_list_fill(const struct graph *graph, const char *vertex, int
      *adjacency_list);
00248
00257 void graph_dfs(struct graph *graph, void (*vertex_processing)(char *vertex_name));
00258
00268 struct matrix *graph_adjacency_matrix_create(const struct graph *graph);
00269
```

```
00282 graph_error_t graph_adjacency_matrix_to_dot(const struct graph *graph, const struct matrix
      *adjacency_matrix, const char *folder, const char *filename);
00283
00296 graph_error_t graph_adjacency_matrix_show(const struct graph *graph, const struct matrix
      *adjacency_matrix);
00297
00303 void graph_adjacency_matrix_free(struct matrix *adjacency_matrix);
00304
00314 struct matrix *graph_floyd_warshall(const struct graph *graph);
00315
00321 void graph_free(struct graph *graph);
00322
00323 #endif // GRAPH_H__
```

4.3 D:/files from internet/important/learning/github/c-modules/↩ Graph/code/src/graph.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include "graph.h"
```

Include dependency graph for graph.c:



Functions

- void **graph_initialize** (struct **graph** ∗ **graph**)

  Initialization of graph by zero.
- int **graph_is_empty** (const struct **graph** ∗ **graph**)

  Checking for graph emtiness.
- int **graph_has_vertex** (const struct **graph** ∗ **graph**, const char ∗vertex)

  Checking for the presence of a vertex in the graph.
- int **graph_has_edge** (const struct **graph** ∗ **graph**, const char ∗start_vertex, const char ∗end_vertex)

  Checking for the presence of a edge in the graph.
- **graph_error_t graph_add_vertex** (struct **graph** ∗ **graph**, const char ∗vertex)

  Adding a vertex to a graph.

- **graph_error_t graph_delete_vertex** (struct **graph** * **graph**, const char *vertex)

  Deleting vertex from graph.
- **graph_error_t graph_add_edge** (struct **graph** * **graph**, const char *start_vertex, const char *end_vertex, size_t edge_length)

  Adding an edge to a graph.
- **graph_error_t graph_delete_edge** (struct **graph** * **graph**, const char *start_vertex, const char *end_↩ vertex)

  Deleting edge from graph.
- **graph_error_t graph_to_dot** (const struct **graph** * **graph**, const char *folder, const char *filename)

  Creating a dot file by graph.
- **graph_error_t graph_show** (const struct **graph** * **graph**)

  Draw graph using Graphviz and show it.
- size_t **graph_adjacency_list_size** (const struct **graph** * **graph**, const char *vertex)

  Counting the number of adjacent vertices (the size of the adjacency list)
- **graph_error_t graph_adjacency_list_fill** (const struct **graph** * **graph**, const char *vertex, int *adjacency_list)

  Filling in the adjacency list.
- struct **matrix** * **graph_adjacency_matrix_create** (const struct **graph** * **graph**)

  Creating adjacency matrix by graph.
- **graph_error_t graph_adjacency_matrix_to_dot** (const struct **graph** * **graph**, const struct **matrix** *adjacency_matrix, const char *folder, const char *filename)

  Creating a dot file of adjacency matrix of graph.
- **graph_error_t graph_adjacency_matrix_show** (const struct **graph** * **graph**, const struct **matrix** *adjacency_matrix)

  Draw graph adjacency matrix using Graphviz and show it.
- void **graph_dfs** (struct **graph** * **graph**, void(*vertex_processing)(char *vertex_name))

  Graph traversal using a depth-first search algorithm.
- struct **matrix** * **graph_floyd_warshall** (const struct **graph** * **graph**)

  Finding the shortest distance matrix using the Floyd-Warshall algorithm.
- void **graph_adjacency_matrix_free** (struct **matrix** *adjacency_matrix)

  Free adjacency matrix.
- void **graph_free** (struct **graph** * **graph**)

  Free graph.

4.3.1 Function Documentation

4.3.1.1 graph_add_edge()

```
graph_error_t graph_add_edge (
            struct  graph * graph,
            const char * start_vertex,
            const char * end_vertex,
            size_t edge_length )
```

Adding an edge to a graph.

Parameters

in	*graph*	Graph descriptor
in	*start_vertex*	Start vertex name
in	*end_vertex*	End vertex name

Returns

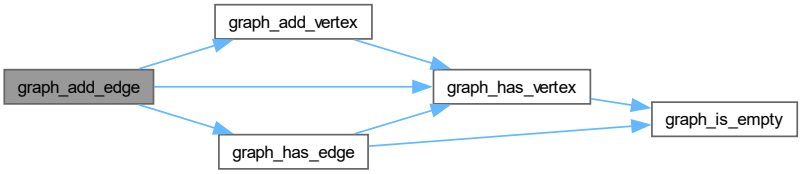
_GRAPH_OK__, _GRAPH_MEM__, _GRAPH_INCORRECT_ARG__, _GRAPH_EXIST__

Note

- You cannot add a copy of an existing edge
- When adding an edge consisting of new vertices, new vertices will be added to the graph

Definition at line **139** of file **graph.c**.

Here is the call graph for this function:



4.3.1.2 graph_add_vertex()

```
graph_error_t graph_add_vertex (
            struct  graph * graph,
            const char * vertex )
```

Adding a vertex to a graph.

Parameters

in	*graph*	Graph descriptor
in	*vertex*	Vertex name

Returns

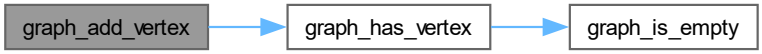
_GRAPH_OK__, _GRAPH_MEM__, _GRAPH_INCORRECT_ARG__, _GRAPH_EXIST__

Note

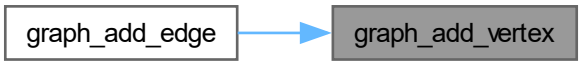
- You cannot add a copy of an existing vertex
- You cannot add a vertex with a name of zero length
- You cannot add a vertex with a name containing special characters - #%()><{}-/\|:;, and quotes

Definition at line **58** of file **graph.c**.

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.3 graph_adjacency_list_fill()

```
graph_error_t graph_adjacency_list_fill (
            const struct  graph * graph,
            const char * vertex,
            int * adjacency_list )
```

Filling in the adjacency list.

Parameters

in	*graph*	Graph descriptor
in	*vertex*	Vertex name
in	*size*	Adjacency list size
out	*adjacency_list*	Adjacency list descriptor

Returns

_GRAPH_OK__, _GRAPH_INCORRECT_ARG__

Definition at line **362** of file **graph.c**.

4.3.1.4 graph_adjacency_list_size()

```
size_t graph_adjacency_list_size (
            const struct  graph * graph,
            const char * vertex )
```

Counting the number of adjacent vertices (the size of the adjacency list)

Parameters

in	*graph*	Graph descriptor
in	*vertex*	Vertex name

Returns

The number of adjacent vertices

Note

- If the arguments is incorrect, the function returns 0

Definition at line **346** of file **graph.c**.

Here is the call graph for this function:

4.3.1.5 graph_adjacency_matrix_create()

```
struct  matrix * graph_adjacency_matrix_create (
            const struct  graph * graph )
```

Creating adjacency matrix by graph.

Parameters

in	*graph*	Graph descriptor

Returns

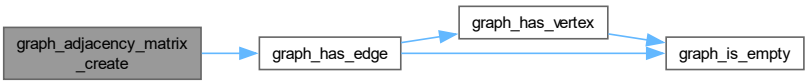
Adjacency matrix descriptor

Note

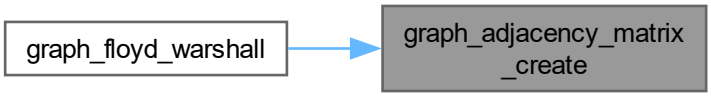
- If errors occur, the function returns NULL

Definition at line **391** of file **graph.c**.

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.6 graph_adjacency_matrix_free()

```
void graph_adjacency_matrix_free (
            struct  matrix * adjacency_matrix )
```

Free adjacency matrix.

Parameters

in	*adjacency_matrix*	Adjacency matrix descriptor

Definition at line **671** of file **graph.c**.

4.3.1.7 graph_adjacency_matrix_show()

```
graph_error_t graph_adjacency_matrix_show (
            const struct  graph * graph,
            const struct  matrix * adjacency_matrix )
```

Draw graph adjacency matrix using Graphviz and show it.

Parameters

in	*graph*	Graph descriptor
in	*adjacency_matrix*	Adjacency matrix descriptor

Returns

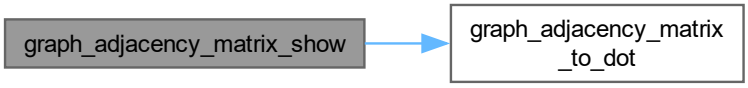
_GRAPH_OK__, _GRAPH_MEM__, _GRAPH_INCORRECT_ARGS__, _GRAPH_OS_ERROR__

Note

- Linux: the graph adjacency matrix is demonstrated using `eog`
- Windows: the graph adjacency matrix is demonstrated using `mspaint`
- The function creates a separate folder for temporary files and deletes it at the end of the work

Definition at line **548** of file **graph.c**.

Here is the call graph for this function:



4.3.1.8 graph_adjacency_matrix_to_dot()

```
graph_error_t graph_adjacency_matrix_to_dot (
            const struct  graph * graph,
            const struct  matrix * adjacency_matrix,
            const char * folder,
            const char * filename )
```

Creating a dot file of adjacency matrix of graph.

Parameters

in	*graph*	Graph descriptor
in	*adjacency_matrix*	Adjacency matrix descriptor
in	*folder*	Folder name
in	*filename*	File name

Returns

_GRAPH_OK__, _GRAPH_INCORRECT_ARG__,_GRAPH_MEM__, _GRAPH_OS_ERROR__

Note

- The pointer to the `folder` string can take the `NULL` value. In this case, the folder will not be created

Definition at line **456** of file **graph.c**.

Here is the caller graph for this function:

4.3.1.9 graph_delete_edge()

```
graph_error_t graph_delete_edge (
            struct  graph * graph,
            const char * start_vertex,
            const char * end_vertex )
```

Deleting edge from graph.

Parameters

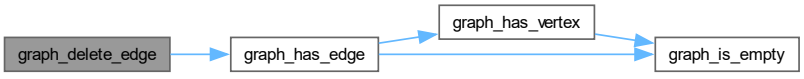
in	*graph*	Graph descriptor
in	*start_vertex*	Start vertex name
in	*end_vertex*	End vertex name

Returns

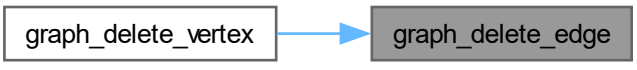
_GRAPH_OK__, _GRAPH_INCORRECT_ARG__, _GRAPH_EMPTY__, _GRAPH_NOT_FOUND__

Definition at line **185** of file **graph.c**.

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.10 graph_delete_vertex()

```
graph_error_t graph_delete_vertex (
            struct  graph * graph,
            const char * vertex )
```

Deleting vertex from graph.

Parameters

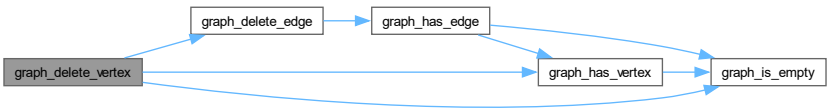
in	*graph*	Graph descriptor
in	*vertex*	Vertex name

Returns

_GRAPH_OK__, _GRAPH_INCORRECT_ARG__, _GRAPH_EMPTY__, _GRAPH_NOT_FOUND__

Definition at line **93** of file **graph.c**.

Here is the call graph for this function:



4.3.1.11 graph_dfs()

```
void graph_dfs (
            struct  graph * graph,
            void(*)(char *vertex_name) vertex_processing )
```

Graph traversal using a depth-first search algorithm.

Parameters

in	*graph*	Graph descriptor
in	*vertex_processing*	Vertex processing function

Note

- If the input arguments are incorrect, the function will not work

Definition at line **618** of file **graph.c**.

4.3.1.12 graph_floyd_warshall()

```
struct  matrix * graph_floyd_warshall (
            const struct  graph * graph )
```

Finding the shortest distance matrix using the Floyd-Warshall algorithm.

Parameters

graph	Graph descriptor

Returns

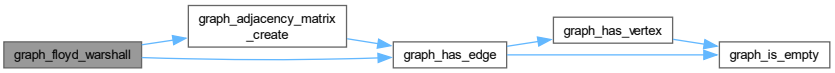
The shortest distance matrix

Note

- If errors occur, the function returns NULL

Definition at line **634** of file **graph.c**.

Here is the call graph for this function:



4.3.1.13 graph_free()

```
void graph_free (
            struct  graph * graph )
```

Free graph.

Parameters

in	*graph*	Graph descriptor

Definition at line **683** of file **graph.c**.

4.3.1.14 graph_has_edge()

```
int graph_has_edge (
            const struct  graph * graph,
```

```
const char * start_vertex,
const char * end_vertex )
```

Checking for the presence of a edge in the graph.

Parameters

in	*graph*	Graph descriptor
in	*start_vertex*	Start vertex name
in	*end_vertex*	End vertex name

Returns

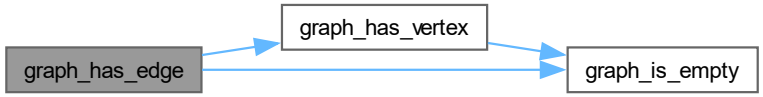
`1 - True / 0 - False`

Note

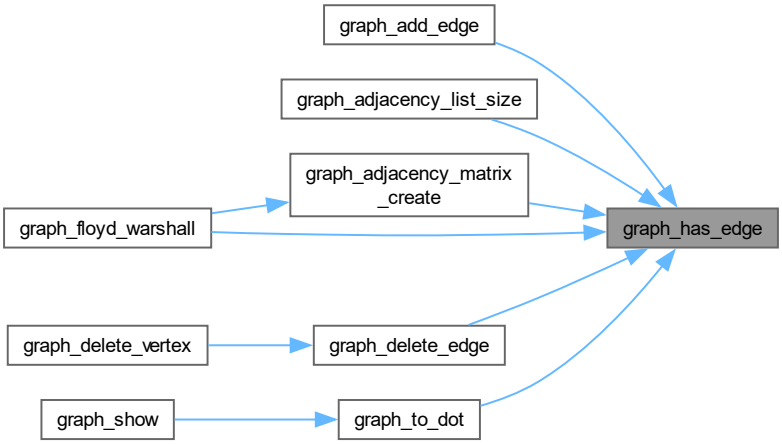
- If incorrect arguments are passed, the function returns `0` (`False`)

Definition at line **34** of file **graph.c**.

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.15 graph_has_vertex()

```
int graph_has_vertex (
            const struct  graph * graph,
            const char * vertex )
```

Checking for the presence of a vertex in the graph.

Parameters

in	*graph*	Graph descriptor
in	*vertex*	Vertex name

Returns

`1 - True / 0 - False`

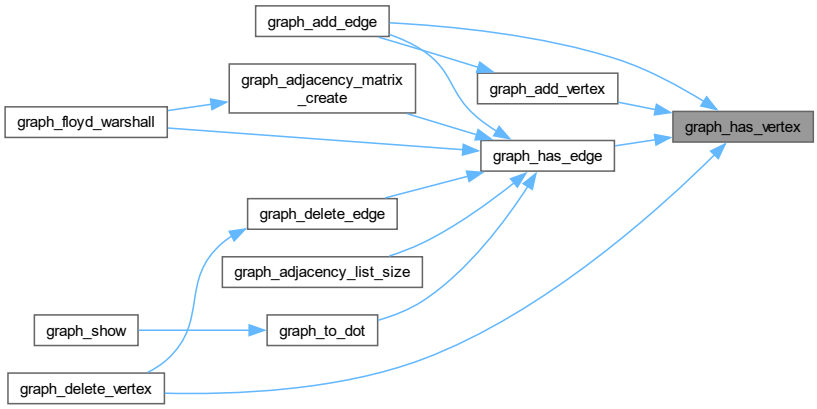
Note

- If incorrect arguments are passed, the function returns `0` (`False`)

Definition at line **17** of file **graph.c**.

Here is the call graph for this function:

Here is the caller graph for this function:



4.3.1.16 graph_initialize()

```
void graph_initialize (
            struct  graph * graph )
```

Initialization of graph by zero.

Parameters

in	*graph*	Graph descriptor

Note

- If the graph descriptor is NULL, the function will not cause a segmentation error

Definition at line **5** of file **graph.c**.

4.3.1.17 graph_is_empty()

```
int graph_is_empty (
            const struct  graph * graph )
```

Checking for graph emtiness.

Parameters

in	*graph*	Graph descriptor

Returns

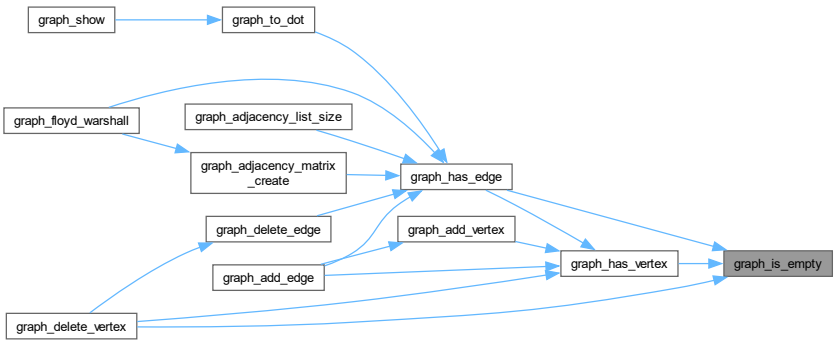
`1 - True / 0 - False`

Note

- If incorrect arguments are passed, the function returns `1` (`True`)

Definition at line **10** of file **graph.c**.

Here is the caller graph for this function:



4.3.1.18 graph_show()

```
graph_error_t graph_show (
            const struct  graph * graph )
```

Draw graph using Graphviz and show it.

Parameters

in	*graph*	Graph descriptor

Returns

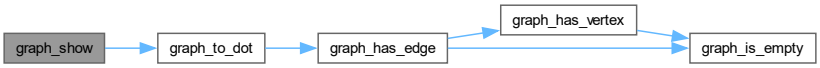
_GRAPH_OK__, _GRAPH_MEM__, _GRAPH_INCORRECT_ARGS__, _GRAPH_OS_ERROR__

Note

- Linux: the graph is demonstrated using `eog`
- Windows: the graph is demonstrated using `mspaint`
- The function creates a separate folder for temporary files and deletes it at the end of the work

Definition at line **313** of file **graph.c**.

Here is the call graph for this function:



4.3.1.19 graph_to_dot()

```
graph_error_t graph_to_dot (
            const struct  graph * graph,
            const char * folder,
            const char * filename )
```

Creating a dot file by graph.

Parameters

in	*graph*	Graph descriptor
in	*folder*	Folder name
in	*filename*	File name

Returns

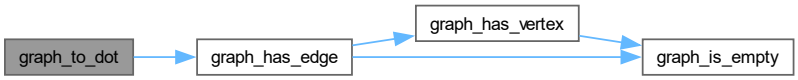
_GRAPH_OK__, _GRAPH_INCORRECT_ARG__,_GRAPH_MEM__, _GRAPH_OS_ERROR__

Note

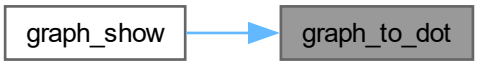
- The pointer to the `folder` string can take the `NULL` value. In this case, the folder will not be created

Definition at line **225** of file **graph.c**.

Here is the call graph for this function:



Here is the caller graph for this function:



4.4 graph.c

Go to the documentation of this file.

```
00001 #include <stdlib.h>
00002 #include <string.h>
00003 #include "graph.h"
00004
00005 void graph_initialize(struct graph *graph)
00006 {
00007     *graph = (struct graph) {0};
00008 }
00009
00010 int graph_is_empty(const struct graph *graph)
00011 {
00012     if (graph)
00013         return graph->vertices_amount == 0;
00014     return 1;
00015 }
00016
00017 int graph_has_vertex(const struct graph *graph, const char *vertex)
00018 {
00019     if (graph && vertex)
00020     {
00021         if (!graph_is_empty(graph))
00022         {
00023             for (size_t i = 0; i < graph->vertices_amount; i++)
00024             {
00025                 if (!strcmp(vertex, graph->vertices[i]))
00026                     return 1;
00027             }
00028         }
00029     }
00030
00031     return 0;
00032 }
00033
00034 int graph_has_edge(const struct graph *graph, const char *start_vertex, const char *end_vertex)
00035 {
00036     if (graph && start_vertex && strlen(start_vertex) && end_vertex && strlen(end_vertex))
00037     {
00038         if (!graph_is_empty(graph))
00039         {
00040             if (graph_has_vertex(graph, start_vertex) && graph_has_vertex(graph, end_vertex))
```

```
00041             {
00042                 for (size_t i = 0; i < graph->edges_amount; i++)
00043                 {
00044                     struct edge current_edge = graph->edges[i];
00045 
00046                     if (!strcmp(start_vertex, current_edge.start_vertex) \
00047                         && !strcmp(end_vertex, current_edge.end_vertex))
00048 
00049                         return 1;
00050                 }
00051             }
00052         }
00053     }
00054 
00055     return 0;
00056 }
00057 
00058 graph_error_t graph_add_vertex(struct graph *graph, const char *vertex)
00059 {
00060     if (!graph || !vertex || !strlen(vertex))
00061         return _GRAPH_INCORRECT_ARG__;
00062 
00063     for (size_t i = 0; vertex[i] != '\0'; i++)
00064     {
00065         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, vertex[i]))
00066             return _GRAPH_INCORRECT_ARG__;
00067     }
00068 
00069     if (graph_has_vertex(graph, vertex))
00070         return _GRAPH_EXIST__;
00071 
00072     // expanding a dynamic array of vertices
00073 
00074     char **tmp = (char **) realloc(graph->vertices, (graph->vertices_amount + 1) * sizeof(char *));
00075 
00076     if (!tmp)
00077         return _GRAPH_MEM__;
00078     else
00079         graph->vertices = tmp;
00080 
00081     // creating a dynamic copy of the vertex name
00082 
00083     graph->vertices[graph->vertices_amount] = strdup(vertex);
00084 
00085     if (!graph->vertices[graph->vertices_amount])
00086         return _GRAPH_MEM__;
00087     else
00088         graph->vertices_amount++;
00089 
00090     return _GRAPH_OK__;
00091 }
00092 
00093 graph_error_t graph_delete_vertex(struct graph *graph, const char *vertex)
00094 {
00095     if (!graph || !vertex || !strlen(vertex))
00096         return _GRAPH_INCORRECT_ARG__;
00097 
00098     for (size_t i = 0; vertex[i] != '\0'; i++)
00099     {
00100         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, vertex[i]))
00101             return _GRAPH_INCORRECT_ARG__;
00102     }
00103 
00104     if (graph_is_empty(graph))
00105         return _GRAPH_EMPTY__;
00106 
00107     if (!graph_has_vertex(graph, vertex))
00108         return _GRAPH_NOT_FOUND__;
00109 
00110     // removing edges from a given vertex
00111 
00112     for (size_t i = 0; i < graph->vertices_amount; i++)
00113     {
00114         graph_delete_edge(graph, vertex, graph->vertices[i]);
00115         graph_delete_edge(graph, graph->vertices[i], vertex);
00116     }
00117 
00118     // removing a pointer to a vertex using sequential displacement of elements
00119 
00120     for (size_t i = 0; i < graph->vertices_amount; i++)
00121     {
00122         if (!strcmp(vertex, graph->vertices[i]))
00123         {
00124             free(graph->vertices[i]);
00125 
00126             for (size_t j = i; j < graph->vertices_amount - 1; j++)
00127                 graph->vertices[j] = graph->vertices[j + 1];
```

```
00128
00129             break;
00130         }
00131     }
00132
00133     graph->vertices = (char **) realloc(graph->vertices, (graph->vertices_amount - 1) * sizeof(char
      *));
00134     graph->vertices_amount--;
00135
00136     return _GRAPH_OK__;
00137 }
00138
00139 graph_error_t graph_add_edge(struct graph *graph, const char *start_vertex, const char *end_vertex,
      size_t edge_length)
00140 {
00141     if (!graph || !start_vertex || !strlen(start_vertex) || strlen(start_vertex) > _STRING__ \
00142         || !end_vertex || !strlen(end_vertex) || strlen(end_vertex) > _STRING__)
00143         return _GRAPH_INCORRECT_ARG__;
00144
00145     for (size_t i = 0; start_vertex[i] != '\0'; i++)
00146     {
00147         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, start_vertex[i]))
00148             return _GRAPH_INCORRECT_ARG__;
00149     }
00150
00151     for (size_t i = 0; end_vertex[i] != '\0'; i++)
00152     {
00153         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, end_vertex[i]))
00154             return _GRAPH_INCORRECT_ARG__;
00155     }
00156
00157     if (graph_has_edge(graph, start_vertex, end_vertex))
00158         return _GRAPH_EXIST__;
00159
00160     struct edge edge_to_add = {0};
00161
00162     strcpy(edge_to_add.start_vertex, start_vertex);
00163     strcpy(edge_to_add.end_vertex, end_vertex);
00164     edge_to_add.length = edge_length;
00165
00166     struct edge *tmp = (struct edge *) realloc(graph->edges, (graph->edges_amount + 1) * sizeof(struct
      edge));
00167     if (!tmp)
00168         return _GRAPH_MEM__;
00169     else
00170     {
00171         graph->edges = tmp;
00172         graph->edges[graph->edges_amount] = edge_to_add;
00173         graph->edges_amount++;
00174     }
00175
00176     if (!graph_has_vertex(graph, start_vertex))
00177         graph_add_vertex(graph, start_vertex);
00178
00179     if (!graph_has_vertex(graph, end_vertex))
00180         graph_add_vertex(graph, end_vertex);
00181
00182     return _GRAPH_OK__;
00183 }
00184
00185 graph_error_t graph_delete_edge(struct graph *graph, const char *start_vertex, const char *end_vertex)
00186 {
00187     if (!graph || !start_vertex || !strlen(start_vertex) || strlen(start_vertex) > _STRING__ \
00188         || !end_vertex || !strlen(end_vertex) || strlen(end_vertex) > _STRING__)
00189         return _GRAPH_INCORRECT_ARG__;
00190
00191     for (size_t i = 0; start_vertex[i] != '\0'; i++)
00192     {
00193         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, start_vertex[i]))
00194             return _GRAPH_INCORRECT_ARG__;
00195     }
00196
00197     for (size_t i = 0; end_vertex[i] != '\0'; i++)
00198     {
00199         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, end_vertex[i]))
00200             return _GRAPH_INCORRECT_ARG__;
00201     }
00202
00203     if (!graph_has_edge(graph, start_vertex, end_vertex))
00204         return _GRAPH_NOT_FOUND__;
00205
00206     for (size_t i = 0; i < graph->edges_amount; i++)
00207     {
00208         struct edge current_edge = graph->edges[i];
00209
00210         if (!strcmp(start_vertex, current_edge.start_vertex) && !strcmp(end_vertex,
      current_edge.end_vertex))
```

```
00211         {
00212             for (size_t j = i; j < graph->edges_amount - 1; j++)
00213                 graph->edges[j] = graph->edges[j + 1];
00214 
00215             break;
00216         }
00217     }
00218 
00219     graph->edges = (struct edge *) realloc(graph->edges, (graph->edges_amount - 1) * sizeof(struct
      edge));
00220     graph->edges_amount--;
00221 
00222     return _GRAPH_OK__;
00223 }
00224 
00225 graph_error_t graph_to_dot(const struct graph *graph, const char *folder, const char *filename)
00226 {
00227     if (!graph || !filename || (folder && !strlen(folder)) || !strlen(filename))
00228         return _GRAPH_INCORRECT_ARG__;
00229 
00230     for (size_t i = 0; folder && folder[i] != '\0'; i++)
00231     {
00232         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, folder[i]))
00233             return _GRAPH_INCORRECT_ARG__;
00234     }
00235 
00236     for (size_t i = 0; filename[i] != '\0'; i++)
00237     {
00238         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, filename[i]))
00239             return _GRAPH_INCORRECT_ARG__;
00240     }
00241 
00242     int rc = 0;
00243 
00244     FILE *dot_file = NULL;
00245 
00246     char buffer[_STRING__ + 1];
00247 
00248     // creating folder
00249 
00250     if (folder)
00251     {
00252         sprintf(buffer, "%s %s", "mkdir", folder);
00253 
00254         rc = system(buffer);
00255         if (rc)
00256             return _GRAPH_OS_ERROR__;
00257     }
00258 
00259     // creating file
00260 
00261     folder ? sprintf(buffer, "%s/%s", folder, filename) : sprintf(buffer, "%s", filename);
00262 
00263     dot_file = fopen(buffer, "w");
00264     if (!dot_file)
00265     {
00266         if (folder)
00267         {
00268             sprintf(buffer, "%s %s", "rm -r -f", folder);
00269             system(buffer);
00270         }
00271 
00272         return _GRAPH_MEM__;
00273     }
00274 
00275     // file processing
00276 
00277     fprintf(dot_file, "digraph picture {\n");
00278 
00279     // edges to dot
00280 
00281     for (size_t i = 0; i < graph->edges_amount; i++)
00282     {
00283         struct edge current_edge = graph->edges[i];
00284 
00285         fprintf(dot_file, "\"%s\" -> \"%s\" [label=  %zu];\n", current_edge.start_vertex,
      current_edge.end_vertex, current_edge.length);
00286     }
00287 
00288     // vertices (not in edges) to dot
00289 
00290     for (size_t i = 0; i < graph->vertices_amount; i++)
00291     {
00292         int vertex_drawed = 0;
00293 
00294         for (size_t j = 0; j < graph->vertices_amount && !vertex_drawed; j++)
00295         {
```

```
00296             if (graph_has_edge(graph, graph->vertices[i], graph->vertices[j]))
00297                 vertex_drawed = 1;
00298             else if (graph_has_edge(graph, graph->vertices[j], graph->vertices[i]))
00299                 vertex_drawed = 1;
00300         }
00301
00302         if (!vertex_drawed)
00303             fprintf(dot_file, "\"%s\";\n", graph->vertices[i]);
00304     }
00305
00306     fprintf(dot_file, "}");
00307
00308     fclose(dot_file);
00309
00310     return _GRAPH_OK__;
00311 }
00312
00313 graph_error_t graph_show(const struct graph *graph)
00314 {
00315     int rc = _GRAPH_OK__;
00316
00317     if (!graph)
00318         rc = _GRAPH_INCORRECT_ARG__;
00319
00320     if (rc == _GRAPH_OK__)
00321         rc = graph_to_dot(graph, ".graph_cash", "graph_dependencies.dot");
00322
00323     if (rc == _GRAPH_OK__)
00324     {
00325         rc = system("dot -Tpng .graph_cash/graph_dependencies.dot -o graph.png");
00326         if (rc)
00327             return _GRAPH_OS_ERROR__;
00328     }
00329
00330     if (rc == _GRAPH_OK__)
00331     {
00332         #if defined(__WIN32__)
00333             system("mspaint graph.png");
00334         #elif defined(__linux__)
00335             system("eog graph.png");
00336         #else
00337             #error "Unsupported operating system!"
00338         #endif
00339     }
00340
00341     system("rm -f -r .graph_cash graph.png");
00342
00343     return rc;
00344 }
00345
00346 size_t graph_adjacency_list_size(const struct graph *graph, const char *vertex)
00347 {
00348     if (!graph || !vertex)
00349         return 0;
00350
00351     size_t adjacency_list_size = 0;
00352
00353     for (size_t i = 0; i < graph->vertices_amount; i++)
00354     {
00355         if (graph_has_edge(graph, vertex, graph->vertices[i]))
00356             adjacency_list_size++;
00357     }
00358
00359     return adjacency_list_size;
00360 }
00361
00362 graph_error_t graph_adjacency_list_fill(const struct graph *graph, const char *vertex, int
      *adjacency_list)
00363 {
00364     if (!graph || !vertex || !adjacency_list)
00365         return _GRAPH_INCORRECT_ARG__;
00366
00367     for (size_t i = 0, k = 0; i < graph->edges_amount; i++)
00368     {
00369         struct edge current_edge = graph->edges[i];
00370
00371         if (!strcmp(vertex, current_edge.start_vertex))
00372         {
00373             int end_vertex_finded = 0;
00374
00375             for (size_t j = 0; j < graph->vertices_amount && !end_vertex_finded; j++)
00376             {
00377                 if (!strcmp(graph->vertices[j], current_edge.end_vertex))
00378                 {
00379                     adjacency_list[k++] = j;
00380                     end_vertex_finded = 1;
00381                 }
```

```
00382             }
00383         }
00384
00385         printf("\n");
00386     }
00387
00388     return _GRAPH_OK__;
00389 }
00390
00391 struct matrix *graph_adjacency_matrix_create(const struct graph *graph)
00392 {
00393     if (!graph)
00394         return NULL;
00395
00396     // memory allocation for matrix
00397
00398     struct matrix *matrix = malloc(sizeof(struct matrix));
00399     if (!matrix)
00400         return NULL;
00401
00402     matrix->values = calloc(graph->vertices_amount, sizeof(int *));
00403     if (!matrix->values)
00404     {
00405         free(matrix);
00406         return NULL;
00407     }
00408
00409     for (size_t i = 0; i < graph->vertices_amount; i++)
00410     {
00411         matrix->values[i] = malloc(sizeof(int) * graph->vertices_amount);
00412         if (!matrix->values[i])
00413         {
00414             for (size_t j = 0; j < i; j++)
00415                 free(matrix->values[j]);
00416             free(matrix->values);
00417             free(matrix);
00418
00419             return NULL;
00420         }
00421     }
00422
00423     // matrix fill
00424
00425     for (size_t i = 0; i < graph->vertices_amount; i++)
00426     {
00427         for (size_t j = 0; j < graph->vertices_amount; j++)
00428         {
00429             if (graph_has_edge(graph, graph->vertices[i], graph->vertices[j]))
00430             {
00431                 int edge_is_finded = 0;
00432
00433                 for (size_t k = 0; k < graph->edges_amount && !edge_is_finded; k++)
00434                 {
00435                     struct edge current_edge = graph->edges[k];
00436
00437                     if (!strcmp(current_edge.start_vertex, graph->vertices[i]) \
00438                         && !strcmp(current_edge.end_vertex, graph->vertices[j]))
00439                     {
00440                         matrix->values[i][j] = current_edge.length;
00441                         edge_is_finded = 1;
00442                     }
00443                 }
00444             }
00445             else
00446                 matrix->values[i][j] = INT_MAX;
00447         }
00448     }
00449
00450     matrix->rows = graph->vertices_amount;
00451     matrix->columns = graph->vertices_amount;
00452
00453     return matrix;
00454 }
00455
00456 graph_error_t graph_adjacency_matrix_to_dot(const struct graph *graph, const struct matrix
      *adjacency_matrix, const char *folder, const char *filename)
00457 {
00458     if (!adjacency_matrix || !folder || !filename)
00459         return _GRAPH_INCORRECT_ARG__;
00460
00461     for (size_t i = 0; folder && folder[i] != '\0'; i++)
00462     {
00463         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, folder[i]))
00464             return _GRAPH_INCORRECT_ARG__;
00465     }
00466
00467     for (size_t i = 0; filename[i] != '\0'; i++)
```

```
00468     {
00469         if (strchr(_GRAPH_FORBIDDEN_SEPARATORS__, filename[i]))
00470             return _GRAPH_INCORRECT_ARG__;
00471     }
00472 
00473     int rc = 0;
00474 
00475     FILE *dot_file = NULL;
00476 
00477     char buffer[_STRING__ + 1];
00478 
00479     // creating folder
00480 
00481     if (folder)
00482     {
00483         sprintf(buffer, "%s %s", "mkdir", folder);
00484 
00485         rc = system(buffer);
00486         if (rc)
00487             return _GRAPH_OS_ERROR__;
00488     }
00489 
00490     // creating file
00491 
00492     folder ? sprintf(buffer, "%s/%s", folder, filename) : sprintf(buffer, "%s", filename);
00493 
00494     dot_file = fopen(buffer, "w");
00495     if (!dot_file)
00496     {
00497         if (folder)
00498         {
00499             sprintf(buffer, "%s %s", "rm -r -f", folder);
00500             system(buffer);
00501         }
00502 
00503         return _GRAPH_MEM__;
00504     }
00505 
00506     // creating dot file
00507 
00508     fprintf(dot_file, "digraph picture {\n");
00509     fprintf(dot_file, "  node [shape=plaintext]\n");
00510     fprintf(dot_file, "  \"Adjacency matrix\" [label=<\n");
00511     fprintf(dot_file, "    <table border='0' cellborder='1' cellspacing='0'>\n");
00512 
00513 
00514     fprintf(dot_file, "      <tr>\n");
00515     fprintf(dot_file, "      <td></td>\n");
00516 
00517     for (size_t j = 0; j < graph->vertices_amount; j++)
00518         fprintf(dot_file, "        <td>%s</td>\n", graph->vertices[j]);
00519 
00520 
00521     fprintf(dot_file, "      </tr>\n");
00522 
00523     for(size_t i = 0; i < adjacency_matrix->rows; i++)
00524     {
00525         fprintf(dot_file, "      <tr>\n");
00526         fprintf(dot_file, "      <td>%s</td>\n", graph->vertices[i]);
00527         for (size_t j = 0; j < adjacency_matrix->columns; j++)
00528         {
00529             if (adjacency_matrix->values[i][j] != INT_MAX)
00530                 fprintf(dot_file, "        <td>%d</td>\n", adjacency_matrix->values[i][j]);
00531             else
00532                 fprintf(dot_file, "        <td></td>\n");
00533         }
00534 
00535         fprintf(dot_file, "      </tr>\n");
00536     }
00537 
00538 
00539     fprintf(dot_file, "    </table>\n");
00540     fprintf(dot_file, "  >]\n");
00541     fprintf(dot_file, "}\n");
00542 
00543     fclose(dot_file);
00544 
00545     return _GRAPH_OK__;
00546 }
00547 
00548 graph_error_t graph_adjacency_matrix_show(const struct graph *graph, const struct matrix
    *adjacency_matrix)
00549 {
00550     int rc = _GRAPH_OK__;
00551 
00552     if (!graph || !adjacency_matrix)
00553         rc = _GRAPH_INCORRECT_ARG__;
```

```
00554
00555     if (rc == _GRAPH_OK__)
00556         rc = graph_adjacency_matrix_to_dot(graph, adjacency_matrix, ".graph_cash",
      "graph_adjacency_matrix_dependencies.dot");
00557
00558     if (rc == _GRAPH_OK__)
00559     {
00560         rc = system("dot -Tpng .graph_cash/graph_adjacency_matrix_dependencies.dot -o
      graph_adjacency_matrix.png");
00561         if (rc)
00562             return _GRAPH_OS_ERROR__;
00563     }
00564
00565     if (rc == _GRAPH_OK__)
00566     {
00567         #if defined(__WIN32__)
00568             system("mspaint graph_adjacency_matrix.png");
00569         #elif defined(__linux__)
00570             system("eog graph_adjacency_matrix.png");
00571         #else
00572             #error "Unsupported operating system!"
00573         #endif
00574     }
00575
00576     system("rm -f -r .graph_cash graph_adjacency_matrix.png");
00577
00578     return rc;
00579 }
00580
00581 static inline void __graph_dfs_step(struct graph *graph, void (*vertex_processing)(char *vertex_name),
      int vertex_index, int *new)
00582 {
00583     if (new[vertex_index] == 0)
00584         return;
00585
00586     new[vertex_index] = 0;
00587
00588     // defining an adjacency list
00589
00590     size_t adjacent_vertices = graph_adjacency_list_size(graph, graph->vertices[vertex_index]);
00591
00592     int adjacent_vertices_indexes[adjacent_vertices];
00593
00594     graph_adjacency_list_fill(graph, graph->vertices[vertex_index], adjacent_vertices_indexes);
00595
00596     // processing current vertex
00597
00598     char vertex_copy[_STRING__ + 1];
00599     strcpy(vertex_copy, graph->vertices[vertex_index]);
00600
00601     vertex_processing(graph->vertices[vertex_index]);
00602
00603     for (size_t i = 0; i < graph->edges_amount; i++)
00604     {
00605         if (!strcmp(vertex_copy, graph->edges[i].start_vertex))
00606             strcpy(graph->edges[i].start_vertex, graph->vertices[vertex_index]);
00607
00608         if (!strcmp(vertex_copy, graph->edges[i].end_vertex))
00609             strcpy(graph->edges[i].end_vertex, graph->vertices[vertex_index]);
00610     }
00611
00612     // processing vertices from the adjacency list
00613
00614     for (size_t i = 0, index = adjacent_vertices_indexes[i]; i < adjacent_vertices; i++, index =
      adjacent_vertices_indexes[i])
00615         __graph_dfs_step(graph, vertex_processing, index, new);
00616 }
00617
00618 void graph_dfs(struct graph *graph, void (*vertex_processing)(char *vertex_name))
00619 {
00620     if (!graph || !vertex_processing)
00621         return;
00622
00623     size_t vertices_amount = graph->vertices_amount;
00624
00625     int new[vertices_amount];
00626
00627     for (size_t i = 0; i < vertices_amount; i++)
00628         new[i] = 1;
00629
00630     for (size_t i = 0; i < vertices_amount; i++)
00631         __graph_dfs_step(graph, vertex_processing, i, new);
00632 }
00633
00634 struct matrix *graph_floyd_warshall(const struct graph *graph)
00635 {
00636     if (!graph)
```

```
00637         return NULL;
00638
00639     struct matrix *matrix = graph_adjacency_matrix_create(graph);
00640     if (!matrix)
00641         return NULL;
00642
00643     for (size_t i = 0; i < graph->vertices_amount; i++)
00644         matrix->values[i][i] = 0;
00645
00646     for (size_t i = 0; i < graph->vertices_amount; i++)
00647     {
00648         for (size_t u = 0; u < graph->vertices_amount; u++)
00649         {
00650             for (size_t v = 0; v < graph->vertices_amount; v++)
00651             {
00652                 if (graph_has_edge(graph, graph->vertices[u], graph->vertices[i]) \
00653                     && graph_has_edge(graph, graph->vertices[i], graph->vertices[v]))
00654                 {
00655                     if (matrix->values[u][v] != 0)
00656                         matrix->values[u][v] = matrix->values[u][v] > (matrix->values[u][i] +
      matrix->values[i][v]) ? \
00657                             (matrix->values[u][i] + matrix->values[i][v]) : matrix->values[u][v];
00658                     else
00659                     {
00660                         if (strcmp(graph->vertices[u], graph->vertices[v]) && matrix->values[u][i] &&
      matrix->values[i][v])
00661                             matrix->values[u][v] = matrix->values[u][i] + matrix->values[i][v];
00662                     }
00663                 }
00664             }
00665         }
00666     }
00667
00668     return matrix;
00669 }
00670
00671 void graph_adjacency_matrix_free(struct matrix *adjacency_matrix)
00672 {
00673     if (adjacency_matrix)
00674     {
00675         for (size_t i = 0; i < adjacency_matrix->rows; i++)
00676             free(adjacency_matrix->values[i]);
00677         free(adjacency_matrix->values);
00678     }
00679
00680     free(adjacency_matrix);
00681 }
00682
00683 void graph_free(struct graph *graph)
00684 {
00685     for (size_t i = 0; i < graph->vertices_amount; i++)
00686         free(graph->vertices[i]);
00687
00688     free(graph->vertices);
00689     free(graph->edges);
00690 }
```

Index

_GRAPH_EMPTY__
    graph.h, 12
_GRAPH_EXIST__
    graph.h, 12
_GRAPH_FORBIDDEN_SEPARATORS__
    graph.h, 12
_GRAPH_INCORRECT_ARG__
    graph.h, 12
_GRAPH_MEM__
    graph.h, 12
_GRAPH_NOT_FOUND__
    graph.h, 12
_GRAPH_OK__
    graph.h, 12
_GRAPH_OS_ERROR__
    graph.h, 13
_STRING__
    graph.h, 13

columns
    matrix, 8

D:/files from internet/important/learning/github/c-modules/Graph/code/inc/graph.h, 9, 27
D:/files from internet/important/learning/github/c-modules/Graph/code/src/graph.c, 28, 44

edge, 5
    end_vertex, 5
    length, 5
    start_vertex, 6
edges
    graph, 7
edges_amount
    graph, 7
end_vertex
    edge, 5

graph, 6
    edges, 7
    edges_amount, 7
    vertices, 7
    vertices_amount, 7
graph.c
    graph_add_edge, 29
    graph_add_vertex, 30
    graph_adjacency_list_fill, 31
    graph_adjacency_list_size, 31
    graph_adjacency_matrix_create, 32
    graph_adjacency_matrix_free, 33
    graph_adjacency_matrix_show, 33
    graph_adjacency_matrix_to_dot, 34
    graph_delete_edge, 35
    graph_delete_vertex, 35
    graph_dfs, 36
    graph_floyd_warshall, 36
    graph_free, 37
    graph_has_edge, 37
    graph_has_vertex, 39
    graph_initialize, 40
    graph_is_empty, 41
    graph_show, 41
    graph_to_dot, 43
graph.h
    _GRAPH_EMPTY__, 12
    _GRAPH_EXIST__, 12
    _GRAPH_FORBIDDEN_SEPARATORS__, 12
    _GRAPH_INCORRECT_ARG__, 12
    _GRAPH_MEM__, 12
    _GRAPH_NOT_FOUND__, 12
    _GRAPH_OK__, 12
    _GRAPH_OS_ERROR__, 13
    _STRING__, 13
    graph_add_edge, 13
    graph_add_vertex, 14
    graph_adjacency_list_fill, 15
    graph_adjacency_list_size, 15
    graph_adjacency_matrix_create, 16
    graph_adjacency_matrix_free, 17
    graph_adjacency_matrix_show, 17
    graph_adjacency_matrix_to_dot, 18
    graph_delete_edge, 19
    graph_delete_vertex, 20
    graph_dfs, 20
    graph_error_t, 13
    graph_floyd_warshall, 21
    graph_free, 21
    graph_has_edge, 22
    graph_has_vertex, 23
    graph_initialize, 24
    graph_is_empty, 24
    graph_show, 25
    graph_to_dot, 26
graph_add_edge
    graph.c, 29
    graph.h, 13
graph_add_vertex
    graph.c, 30
    graph.h, 14

graph_adjacency_list_fill
graph.c, 31
graph.h, 15
graph_adjacency_list_size
graph.c, 31
graph.h, 15
graph_adjacency_matrix_create
graph.c, 32
graph.h, 16
graph_adjacency_matrix_free
graph.c, 33
graph.h, 17
graph_adjacency_matrix_show
graph.c, 33
graph.h, 17
graph_adjacency_matrix_to_dot
graph.c, 34
graph.h, 18
graph_delete_edge
graph.c, 35
graph.h, 19
graph_delete_vertex
graph.c, 35
graph.h, 20
graph_dfs
graph.c, 36
graph.h, 20
graph_error_t
graph.h, 13
graph_floyd_warshall
graph.c, 36
graph.h, 21
graph_free
graph.c, 37
graph.h, 21
graph_has_edge
graph.c, 37
graph.h, 22
graph_has_vertex
graph.c, 39
graph.h, 23
graph_initialize
graph.c, 40
graph.h, 24
graph_is_empty
graph.c, 41
graph.h, 24
graph_show
graph.c, 41
graph.h, 25
graph_to_dot
graph.c, 43
graph.h, 26

length
edge, 5

matrix, 7
columns, 8
rows, 8
values, 8

rows
matrix, 8

start_vertex
edge, 6

values
matrix, 8
vertices
graph, 7
vertices_amount
graph, 7