Graph

beta

Generated by Doxygen 1.10.0

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

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

3.1.2 Field Documentation

3.1.2.1 end_vertex

```
char end_vertex[ _STRING__+1]
```

3.1.2.2 length

```
size_t length
```

3.1.2.3 start_vertex

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

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:

edge
edges
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

3.2.2 Field Documentation

3.2.2.1 edges

```
struct  edge* edges
```

3.2.2.2 edges_amount

```
size_t edges_amount
```

3.2.2.3 vertices

```
char** vertices
```

3.2.2.4 vertices_amount

```
size_t vertices_amount
```

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

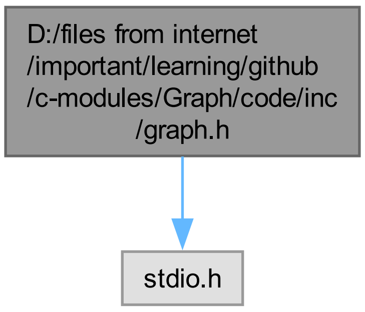
- D:/files from internet/important/learning/github/c-modules/Graph/code/inc/ **graph.h**

Chapter 4

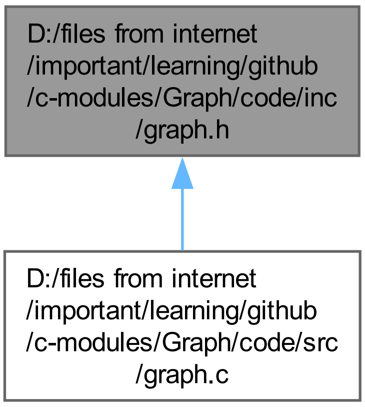
File Documentation

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:


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

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


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

Data Structures

- 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.
- 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.

4.1.1.2 _GRAPH_EXIST__

```
#define _GRAPH_EXIST__ -5
```

Graph already has object.

4.1.1.3 _GRAPH_FORBIDDEN_SEPARATORS__

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

Forbidden characters for vertex name

4.1.1.4 _GRAPH_INCORRECT_ARG__

```
#define _GRAPH_INCORRECT_ARG__ -2
```

Incorrect arguments in function.

4.1.1.5 _GRAPH_MEM__

```
#define _GRAPH_MEM__ -1
```

Memory shortage error.

4.1.1.6 _GRAPH_NOT_FOUND__

```
#define _GRAPH_NOT_FOUND__ -4
```

Object not in graph.

4.1.1.7 _GRAPH_OK__

```
#define _GRAPH_OK__ 0
```

Positive return code.

4.1.1.8 _GRAPH_OS_ERROR__

```
#define _GRAPH_OS_ERROR__ -6
```

Operating system error.

4.1.1.9 _STRING__

```
#define _STRING__ 256
```

Max length of string

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.

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

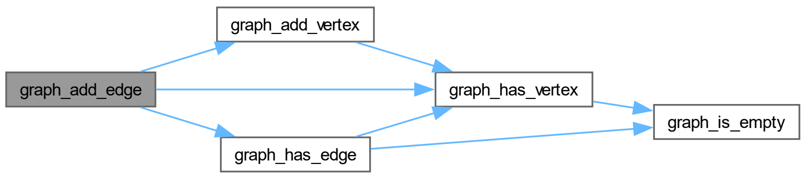
Returns

`__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

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

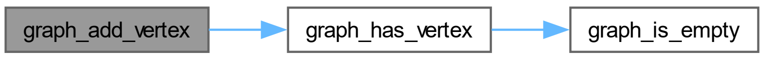
Returns

`__GRAPH_OK__`, `__GRAPH_MEM__`, `__GRAPH_INCORRECT_ARG__`, `__GRAPH_EXIST__`

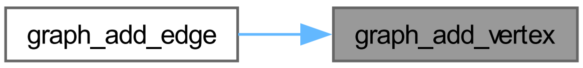
Note

- 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

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__`

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

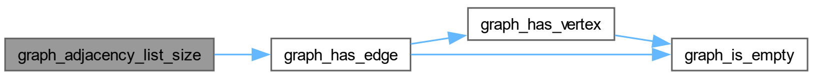
Returns

The number of adjacent vertices

Note

- If the arguments is incorrect, the function returns 0

Here is the call graph for this function:



4.1.3.5 graph_delete_edge()

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

Deleting edge from graph.

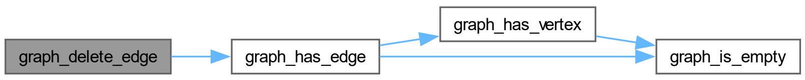
Parameters

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

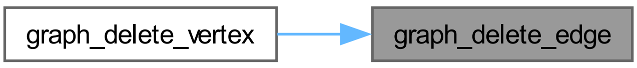
Returns

__GRAPH_OK__, __GRAPH_INCORRECT_ARG__, __GRAPH_EMPTY__, __GRAPH_NOT_FOUND__

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.6 graph_delete_vertex()

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

Deleting vertex from graph.

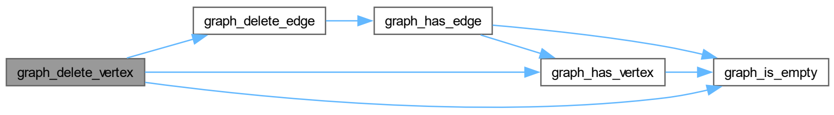
Parameters

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

Returns

`__GRAPH_OK__`, `__GRAPH_INCORRECT_ARG__`, `__GRAPH_EMPTY__`, `__GRAPH_NOT_FOUND__`

Here is the call graph for this function:



4.1.3.7 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

4.1.3.8 graph_free()

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

Free graph.

Parameters

in	*graph*	Graph descriptor

4.1.3.9 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

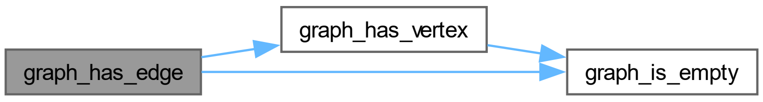
Returns

1 - True / 0 - False

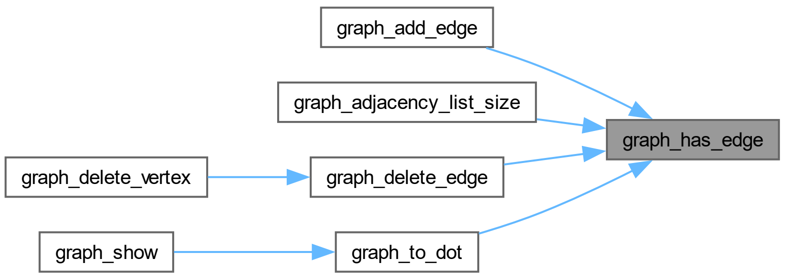
Note

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

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.10 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

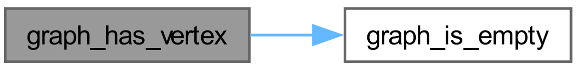
Returns

`1 - True / 0 - False`

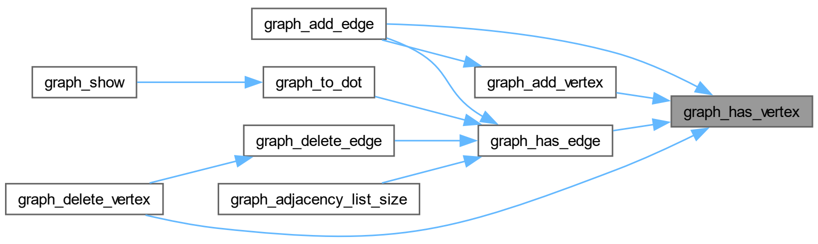
Note

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

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.11 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

4.1.3.12 graph_is_empty()

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

Checking for graph emtiness.

Parameters

in	*graph*	Graph descriptor

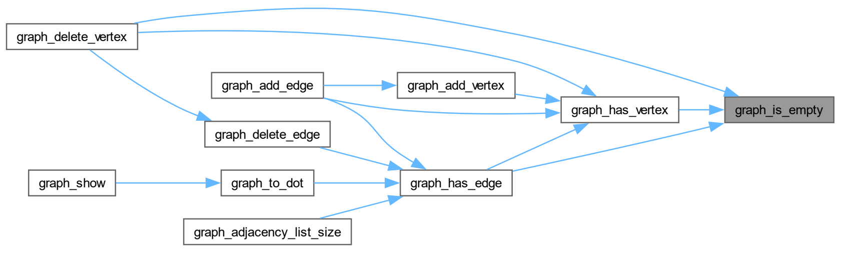
Returns

`1 - True / 0 - False`

Note

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

Here is the caller graph for this function:



4.1.3.13 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__`

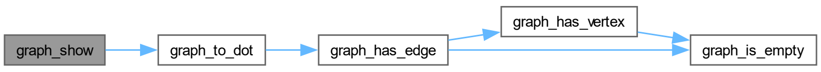
Note

- 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

Here is the call graph for this function:



4.1.3.14 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

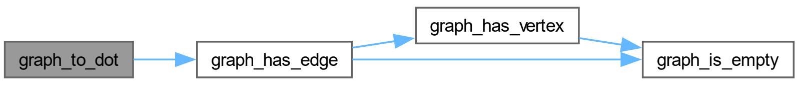
Returns

`__GRAPH_OK__`, `__GRAPH_INCORRECT_ARG__`,`__GRAPH_MEM__`, `__GRAPH_OS_ERROR__`

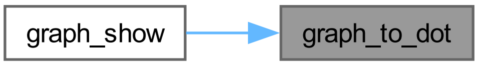
Note

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

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.

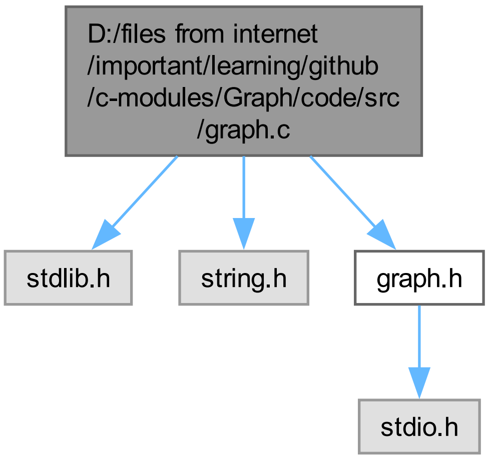
```
#ifndef GRAPH_H__
#define GRAPH_H__

#include <stdio.h>

// Macro

#define _STRING__ 256

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

#define _GRAPH_OK__ 0

#define _GRAPH_MEM__ -1

#define _GRAPH_INCORRECT_ARG__ -2

#define _GRAPH_EMPTY__ -3

#define _GRAPH_NOT_FOUND__ -4

#define _GRAPH_EXIST__ -5

#define _GRAPH_OS_ERROR__ -6

// Structs and functions

struct edge
{
    char start_vertex[_STRING__ + 1];
    char end_vertex[_STRING__ + 1];
    size_t length;
};

struct graph
{
    char **vertices;
    size_t vertices_amount;
    struct edge *edges;
    size_t edges_amount;
};

typedef int graph_error_t;

void graph_initialize(struct graph *graph);

int graph_is_empty(const struct graph *graph);

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

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

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

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

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

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

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

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

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

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

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

void graph_free(struct graph *graph);

#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:


D:/files from internet /important/learning/github /c-modules/Graph/code/src /graph.c
stdlib.h
string.h
graph.h
stdio.h

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.
- void **graph_dfs** (struct **graph** ∗ **graph**, void(∗vertex_processing)(char ∗vertex_name))

  Graph traversal using a depth-first search algorithm.
- 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

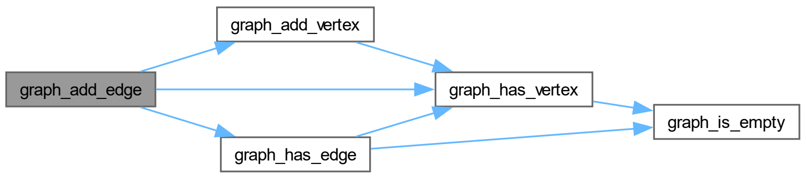
`__GRAPH_OK__`, `__GRAPH_MEM__`, `__GRAPH_INCORRECT_ARG__`, `__GRAPH_EXIST__`

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

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

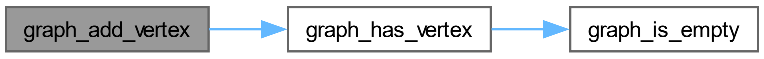
Returns

__GRAPH_OK__, __GRAPH_MEM__, __GRAPH_INCORRECT_ARG__, __GRAPH_EXIST__

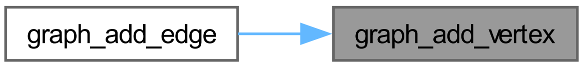
Note

- 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

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__`

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

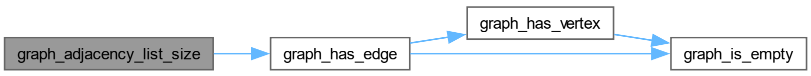
Returns

The number of adjacent vertices

Note

- If the arguments is incorrect, the function returns 0

Here is the call graph for this function:



4.3.1.5 graph_delete_edge()

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

Deleting edge from graph.

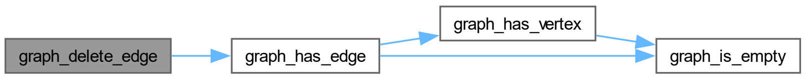
Parameters

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

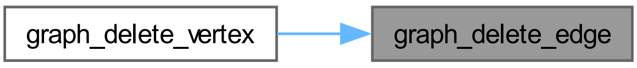
Returns

__GRAPH_OK__, __GRAPH_INCORRECT_ARG__, __GRAPH_EMPTY__, __GRAPH_NOT_FOUND__

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.6 graph_delete_vertex()

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

Deleting vertex from graph.

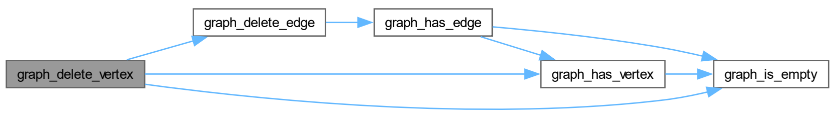
Parameters

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

Returns

__GRAPH_OK__, __GRAPH_INCORRECT_ARG__, __GRAPH_EMPTY__, __GRAPH_NOT_FOUND__

Here is the call graph for this function:



4.3.1.7 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

4.3.1.8 graph_free()

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

Free graph.

Parameters

in	*graph*	Graph descriptor

4.3.1.9 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

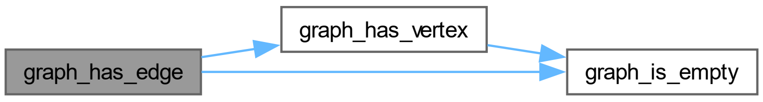
Returns

1 - True / 0 - False

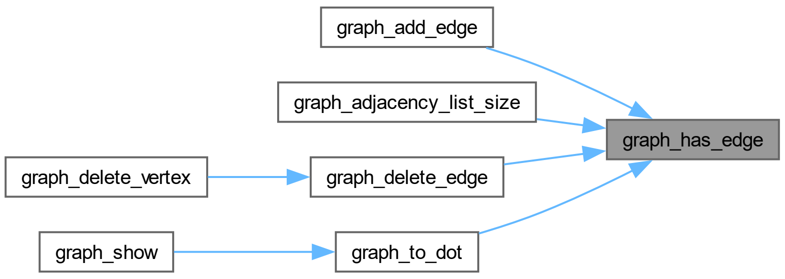
Note

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

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.10 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

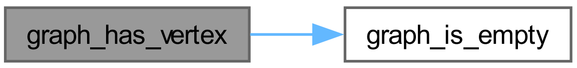
Returns

`1 - True / 0 - False`

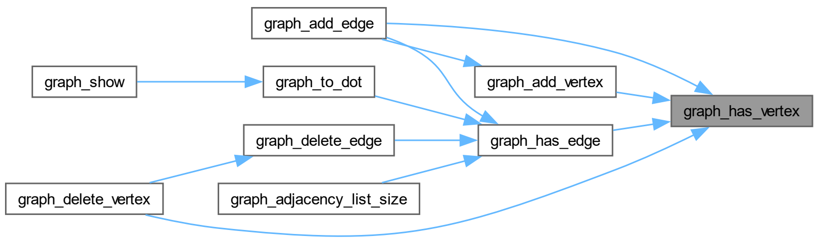
Note

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

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.11 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

4.3.1.12 graph_is_empty()

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

Checking for graph emtiness.

Parameters

in	*graph*	Graph descriptor

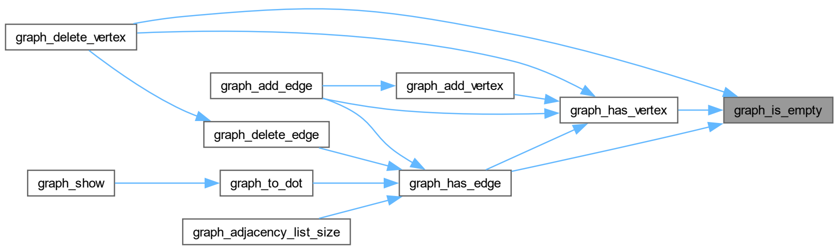
Returns

`1 - True / 0 - False`

Note

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

Here is the caller graph for this function:



4.3.1.13 graph_show()

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

Draw graph using Graphviz and show it.

Parameters

in	*graph*	Graph descriptor

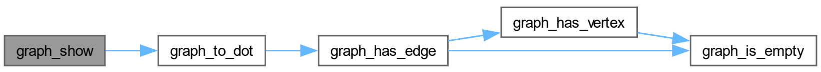
Returns

`__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

Here is the call graph for this function:



4.3.1.14 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

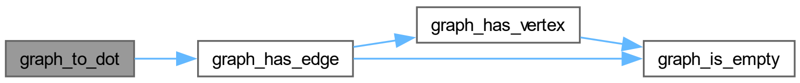
Returns

`__GRAPH_OK__`, `__GRAPH_INCORRECT_ARG__`,`__GRAPH_MEM__`, `__GRAPH_OS_ERROR__`

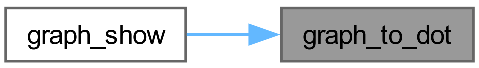
Note

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

Here is the call graph for this function:



Here is the caller graph for this function:



Index