

Bus Navigation Program

Generated by Doxygen 1.9.2

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Coordinates Struct Reference	5
3.2 DisjointSets< T > Class Template Reference	6
3.3 Edge Struct Reference	6
3.3.1 Detailed Description	7
3.4 EdgeKruskal Struct Reference	7
3.4.1 Detailed Description	8
3.5 Graph Class Reference	8
3.5.1 Member Function Documentation	9
3.5.1.1 addEdge() [1/3]	9
3.5.1.2 addEdge() [2/3]	9
3.5.1.3 addEdge() [3/3]	10
3.5.1.4 bfs() [1/2]	11
3.5.1.5 bfs() [2/2]	11
3.5.1.6 calculateChangesOfLine()	12
3.5.1.7 calculateDistance() [1/2]	13
3.5.1.8 calculateDistance() [2/2]	14
3.5.1.9 calculateWeightDijkstraLine()	14
3.5.1.10 dijkstra() [1/2]	15
3.5.1.11 dijkstra() [2/2]	15
3.5.1.12 dijkstraLine() [1/2]	16
3.5.1.13 dijkstraLine() [2/2]	17
3.5.1.14 getPathFromGraph() [1/2]	18
3.5.1.15 getPathFromGraph() [2/2]	18
3.5.1.16 insertTemporaryNode()	19
3.5.1.17 kruskal()	20
3.5.1.18 prim()	21
3.5.1.19 printNodes()	21
3.5.1.20 printPath()	21
3.5.1.21 printPathLinesAlgorithm()	22
3.5.1.22 removeTemporaryNodes()	22
3.5.1.23 setSwapDistance()	23
3.6 Menu Class Reference	24
3.6.1 Member Function Documentation	25
3.6.1.1 askMSTKruskal()	25
3.6.1.2 askMSTPrim()	26
3.6.1.3 askStartEnd()	26

3.6.1.4 askSwapDistance()	27
3.6.1.5 askSwapWalkDistance()	28
3.6.1.6 askWalkingDistance()	28
3.6.1.7 bestPathDijkstra()	29
3.6.1.8 clear()	30
3.6.1.9 createGraphLine()	31
3.6.1.10 createGraphLines()	32
3.6.1.11 createGraphStops()	32
3.6.1.12 pressEnterToContinue()	32
3.6.1.13 readDouble()	33
3.6.1.14 readInputBestAlgorithm()	33
3.6.1.15 readInputMenu()	34
3.6.1.16 readInputStopsOrLocation()	34
3.6.1.17 readInt()	35
3.6.1.18 readString()	35
3.6.1.19 run()	36
3.6.1.20 showAlgorithmOptions()	36
3.6.1.21 showMenu()	37
3.6.1.22 showStopsOrLocation()	37
3.6.1.23 split()	37
3.7 MinHeap< K, V > Class Template Reference	38
3.8 Node Struct Reference	39
3.8.1 Detailed Description	39
4 File Documentation	41
4.1 DisjointSets.h	41
4.2 Graph.h	42
4.3 Menu.h	42
4.4 MinHeap.h	43
4.5 Node.h	45
Index	47

Chapter 1

Class Index

1.1 Class List

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

Coordinates	5
DisjointSets< T >	6
Edge	6
EdgeKruskal	7
Graph	8
Menu	24
MinHeap< K, V >	38
Node	39

Chapter 2

File Index

2.1 File List

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

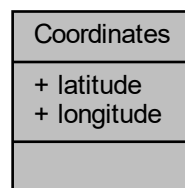
C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/ DisjointSets.h	41
C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/ Graph.h	42
C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/ Menu.h	42
C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/ MinHeap.h	43
C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/ Node.h	45

Chapter 3

Class Documentation

3.1 Coordinates Struct Reference

Collaboration diagram for Coordinates:



Public Attributes

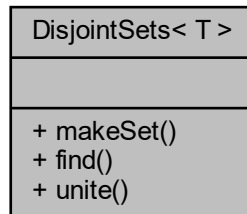
- double **latitude**
- double **longitude**

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

- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Node.h

3.2 DisjointSets< T > Class Template Reference

Collaboration diagram for DisjointSets< T >:



Public Member Functions

- void **makeSet** (const T &x)
- T **find** (const T &x)
- void **unite** (const T &x, const T &y)

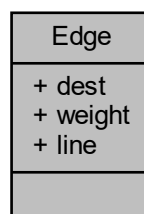
The documentation for this class was generated from the following file:

- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/DisjointSets.h

3.3 Edge Struct Reference

```
#include <Node.h>
```

Collaboration diagram for Edge:



Public Attributes

- int **dest**
- double **weight**
- string **line**

3.3.1 Detailed Description

Lines that connect the bus stops

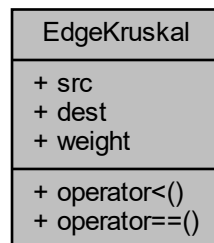
The documentation for this struct was generated from the following file:

- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Node.h

3.4 EdgeKruskal Struct Reference

```
#include <Node.h>
```

Collaboration diagram for EdgeKruskal:



Public Member Functions

- bool **operator<** (const [EdgeKruskal](#) &other) const
- bool **operator==** (const [EdgeKruskal](#) &other) const

Public Attributes

- int **src**
- int **dest**
- double **weight**

3.4.1 Detailed Description

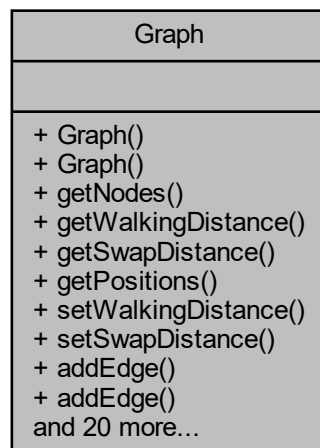
Struct only used for calculating Kruskal MST value

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

- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Node.h

3.5 Graph Class Reference

Collaboration diagram for Graph:



Public Member Functions

- **Graph** (int nodes=0, bool dir=true)
- **Graph** (vector< [Node](#) > nodes, unordered_map< string, int > positions, bool dir=true)
- vector< [Node](#) > & **getNodes** ()
- double **getWalkingDistance** ()
- double **getSwapDistance** ()
- unordered_map< string, int > **getPositions** ()
- void **setWalkingDistance** (double walkingDist)
- void **setSwapDistance** (double swapDist)
- void **addEdge** (string src, string dest, string line)
- void **addEdge** (int src, int dest, string line)
- void **addEdge** (int src, int dest, double weight=1.0)
- double **calculateDistance** ([Coordinates](#) c1, [Coordinates](#) c2)
- double **calculateDistance** (int src, int dest)
- int **calculateChangesOfLine** (map< string, pair< double, int > > lines, int numPrevChangesOfLines, string newLine)

- double [calculateWeightDijkstraLine](#) (int min, double edgeWeight, string newLine)
- long double [prim](#) (int r)
- long double [kruskal](#) ()
- double [bfs](#) (string src, string dest)
- double [bfs](#) (int a, int b)
- double [dijkstra](#) (string src, string dest)
- double [dijkstra](#) (int a, int b)
- double [dijkstraLine](#) (string src, string dest)
- double [dijkstraLine](#) (int a, int b)
- stack< int > [getPathFromGraph](#) (string src, string dest)
- stack< int > [getPathFromGraph](#) (int a, int b)
- void [printPath](#) (stack< int > path)
- void [printPathLinesAlgorithm](#) (stack< int > path)
- void [printNodes](#) ()
- void [insertTemporaryNode](#) ([Coordinates](#) c, bool startType)
- void [removeTemporaryNodes](#) ()

3.5.1 Member Function Documentation

3.5.1.1 [addEdge\(\)](#) [1/3]

```
void Graph::addEdge (
    int src,
    int dest,
    double weight = 1.0 )
```

Adds an edge to the list of outgoing edges to adjacent nodes of the origin node. It does the same for the destiny node, if the graph is undirected.

Parameters

<i>src</i>	The origin's position
<i>dest</i>	The destiny's position
<i>weight</i>	The predetermined weight

3.5.1.2 [addEdge\(\)](#) [2/3]

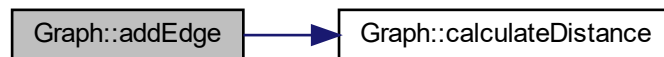
```
void Graph::addEdge (
    int src,
    int dest,
    string line )
```

Adds an edge to the list of outgoing edges to adjacent nodes of the origin node. It does the same for the destiny node, if the graph is undirected. It also calculates the weight of the edge (distance) based on the origin and destiny and includes the line that covers this edge.

Parameters

<i>src</i>	The origin's position
<i>dest</i>	The destiny's position
<i>line</i>	The line's code

Here is the call graph for this function:



3.5.1.3 addEdge() [3/3]

```

void Graph::addEdge (
    string src,
    string dest,
    string line )
  
```

Gets the position of an origin and destiny from their station names, and uses them on the other addEdge method.

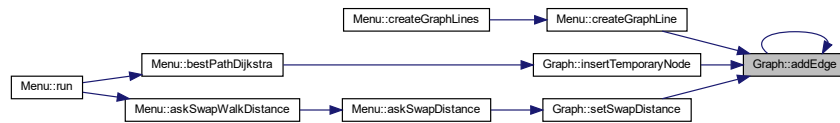
Parameters

<i>src</i>	The origin code
<i>dest</i>	The destiny code
<i>line</i>	The line's code

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.1.4 bfs() [1/2]

```
double Graph::bfs (
    int a,
    int b )
```

Time Complexity: $O(|V| + |E|)$

Parameters

<i>Beginning</i>	node a
<i>Ending</i>	node b

Returns

value of the distance of the path with the least amount of stops

3.5.1.5 bfs() [2/2]

```
double Graph::bfs (
    string src,
    string dest )
```

Determines the positions of an origin and destiny to use on the other bfs method.

Parameters

<i>src</i>	The origin's code
<i>dest</i>	The destiny's code

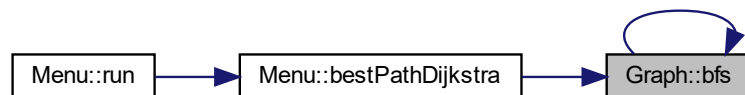
Returns

value of the distance of the path with the least amount of stops

Here is the call graph for this function:



Here is the caller graph for this function:

**3.5.1.6 calculateChangesOfLine()**

```

int Graph::calculateChangesOfLine (
    map< string, pair< double, int > > lines,
    int numPrevChangesOfLines,
    string newLine )

```

Calculates the number of line changes in the new node.

Parameters

<i>lines</i>	The bus lines that reached the previous node
<i>numPrevChangesOfLines</i>	The number of line changes of the previous node
<i>newLine</i>	The line of the edge that we are currently analysing

Returns

number of line changes

Here is the caller graph for this function:

**3.5.1.7 calculateDistance() [1/2]**

```
double Graph::calculateDistance (
    Coordinates c1,
    Coordinates c2 )
```

Uses the mathematical formula to calculate the distance between two points

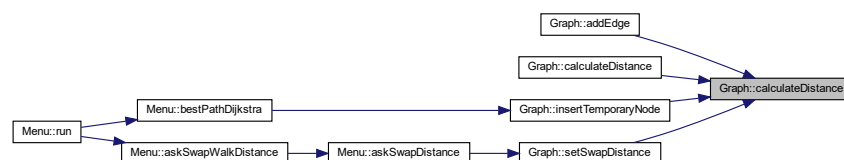
Parameters

<i>c1</i>	The origin's coordinates
<i>c2</i>	The destiny's coordinates

Returns

The distance between two nodes

Here is the caller graph for this function:



3.5.1.8 calculateDistance() [2/2]

```
double Graph::calculateDistance (
    int src,
    int dest )
```

Determines the coordinates of a the origin and destiny and uses them on the other method with this name

Parameters

<i>src</i>	The origin's position
<i>dest</i>	The destiny's position

Returns

The distance between two nodes

Here is the call graph for this function:



3.5.1.9 calculateWeightDijkstraLine()

```
double Graph::calculateWeightDijkstraLine (
    int min,
    double edgeWeight,
    string newLine )
```

Calculates the distance of the new node according to the line of the edge.

Parameters

<i>min</i>	previousNode
<i>edgeWeight</i>	
<i>newLine</i>	

Returns

distance

Here is the caller graph for this function:

**3.5.1.10 dijkstra()** [1/2]

```
double Graph::dijkstra (  
    int a,  
    int b )
```

Time Complexity: $O(|E| \log(|V|))$

Parameters

<i>Beginning</i>	node a
<i>Ending</i>	node b

Returns

value of the distance of the shortest path

3.5.1.11 dijkstra() [2/2]

```
double Graph::dijkstra (  
    string src,  
    string dest )
```

Determines the positions of an origin and destiny to use on the other dijkstra method.

Parameters

<i>src</i>	The origin's code
<i>dest</i>	The destiny's code

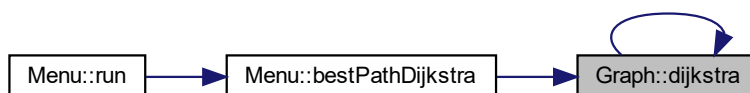
Returns

value of the distance of the shortest path

Here is the call graph for this function:



Here is the caller graph for this function:

**3.5.1.12 dijkstraLine()** [1/2]

```
double Graph::dijkstraLine (
    int a,
    int b )
```

Time Complexity: $O(|E| \log(|V|))$ (overlapping lines make it hard to tell time complexity)

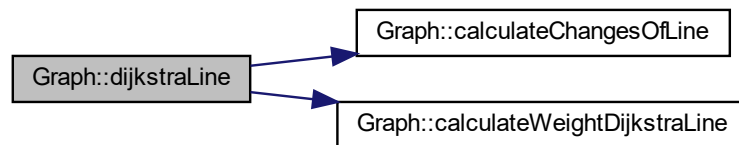
Parameters

<i>Beginning</i>	node a
<i>Ending</i>	node b

Returns

value of the distance of the path with the fewest changes in lines

Here is the call graph for this function:

**3.5.1.13 dijkstraLine() [2/2]**

```
double Graph::dijkstraLine (  
    string src,  
    string dest )
```

Determines the positions of an origin and destiny to use on the other `dijkstraLine` method.

Parameters

<i>src</i>	The origin's code
<i>dest</i>	The destiny's code

Returns

value of the distance of the path with the fewest changes in lines

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.1.14 getPathFromGraph() [1/2]

```

stack< int > Graph::getPathFromGraph (
    int a,
    int b )
  
```

Stores the path in a stack and then returns it.

Parameters

<i>a</i>	The origin's position
<i>b</i>	The destiny's position

Returns

The positions of every node on the way between two nodes

3.5.1.15 getPathFromGraph() [2/2]

```

stack< int > Graph::getPathFromGraph (
    string src,
    string dest )
  
```

Determines the positions of an origin and destiny to use them in the other method of this name.

Parameters

<i>src</i>	The origin's code
<i>dest</i>	The destiny's code

Returns

The positions of every node on the way between two nodes

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.1.16 insertTemporaryNode()

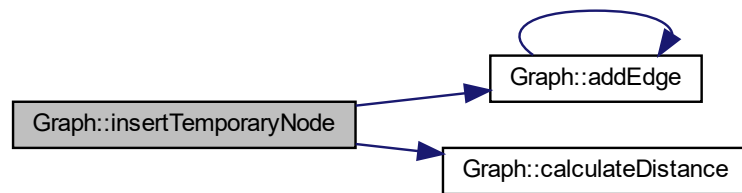
```
void Graph::insertTemporaryNode (
    Coordinates c,
    bool startType )
```

Inserts a temporary node that are at the coordinates.

Parameters

<i>c</i>	<code>Coordinates</code> of origin or destiny
<i>startType</i>	True if the coordinates refer to the origin, false if they refer to the destiny

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.1.17 kruskal()

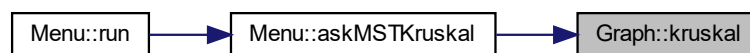
```
long double Graph::kruskal ( )
```

Time Complexity: $|E| \log(|E|)$

Returns

The value of the minimum spanning tree

Here is the caller graph for this function:



3.5.1.18 prim()

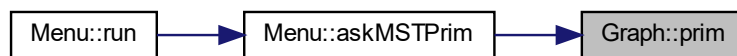
```
long double Graph::prim (
    int r )
```

Time Complexity: $|E| \log(|V|)$

Returns

The value of the minimum spanning tree

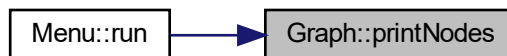
Here is the caller graph for this function:



3.5.1.19 printNodes()

```
void Graph::printNodes ( )
```

Prints the code, position, and local of every node. It also prints the code of every adjacent node and its distance from it, and line that links them. Here is the caller graph for this function:



3.5.1.20 printPath()

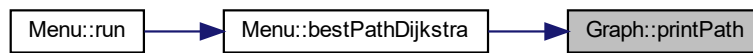
```
void Graph::printPath (
    stack< int > path )
```

Prints every edge in the path between two nodes. It also shows the line and distance of each edge.

Parameters

<i>path</i>	The positions of every node on the way between two nodes
-------------	--

Here is the caller graph for this function:

**3.5.1.21 printPathLinesAlgorithm()**

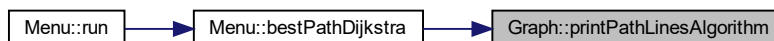
```
void Graph::printPathLinesAlgorithm (  
    stack< int > path )
```

Prints every edge in the path between two nodes. It also shows the line and distance of each edge. This works for the dijkstraLine algorithm

Parameters

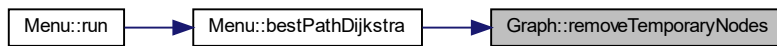
<i>path</i>	The positions of every node on the way between two nodes
-------------	--

Here is the caller graph for this function:

**3.5.1.22 removeTemporaryNodes()**

```
void Graph::removeTemporaryNodes ( )
```

Removes every temporary starting and ending nodes. Here is the caller graph for this function:



3.5.1.23 setSwapDistance()

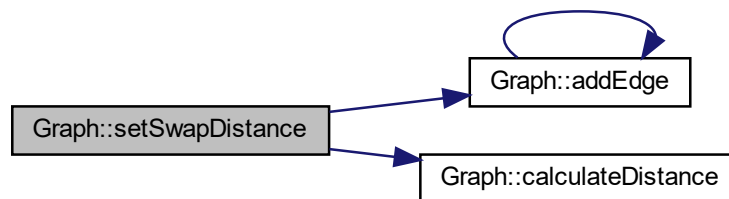
```
void Graph::setSwapDistance (
    double swapDist )
```

Sets a new value for swapDistance. But it also cleans every walking edge that existed before and creates new edges according to this new value.

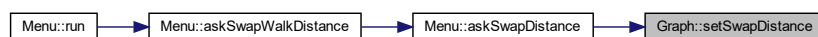
Parameters

<i>swapDist</i>	The distance the user is willing to walk to swap buses
-----------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



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

- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Graph.h
- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Graph.cpp

3.6 Menu Class Reference

Collaboration diagram for Menu:

Menu
<ul style="list-style-type: none"> + Menu() + getGraph() + run() + createGraphStops() + createGraphLines() + createGraphLine() + askStartEnd() + bestPathDijkstra() + askMSTPrim() + askMSTKruskal() + askSwapWalkDistance() + askWalkingDistance() + askSwapDistance() + showMenu() + showStopsOrLocation() + showAlgorithmOptions() + readInputMenu() + readInputStopsOrLocation() + readInputBestAlgorithm() + readDouble() + readInt() + readString() + pressEnterToContinue() + split() + clear()

Public Member Functions

- **Menu** (string directory)
- [Graph](#) & **getGraph** ()
- void [run](#) ()
- void [createGraphStops](#) ()
- void [createGraphLines](#) ()
- void [createGraphLine](#) (string line)
- int [askStartEnd](#) (string &stopBegin, string &stopEnd, [Coordinates](#) &cBegin, [Coordinates](#) &cEnd)
- void [bestPathDijkstra](#) ()
- void [askMSTPrim](#) ()
- void [askMSTKruskal](#) ()
- void [askSwapWalkDistance](#) ()
- void [askWalkingDistance](#) ()
- void [askSwapDistance](#) ()

Static Public Member Functions

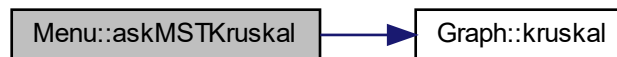
- static void [showMenu](#) ()
- static void [showStopsOrLocation](#) ()
- static void [showAlgorithmOptions](#) ()
- static int [readInputMenu](#) ()
- static int [readInputStopsOrLocation](#) ()
- static int [readInputBestAlgorithm](#) ()
- static double [readDouble](#) ()
- static int [readInt](#) ()
- static string [readString](#) ()
- static void [pressEnterToContinue](#) ()
- static vector< string > [split](#) (string line, string delimiter)
- static void [clear](#) ()

3.6.1 Member Function Documentation

3.6.1.1 askMSTKruskal()

```
void Menu::askMSTKruskal ( )
```

Shows the user the resultant MST from Kruskal's Algorithm. Here is the call graph for this function:



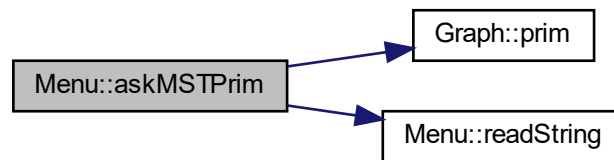
Here is the caller graph for this function:



3.6.1.2 askMSTPrim()

```
void Menu::askMSTPrim ( )
```

Asks the user the origin stop to start with Prim' Algorithm. It also shows the resultant MST from that algorithm. Here is the call graph for this function:



Here is the caller graph for this function:



3.6.1.3 askStartEnd()

```
int Menu::askStartEnd (
    string & stopBegin,
    string & stopEnd,
    Coordinates & cBegin,
    Coordinates & cEnd )
```

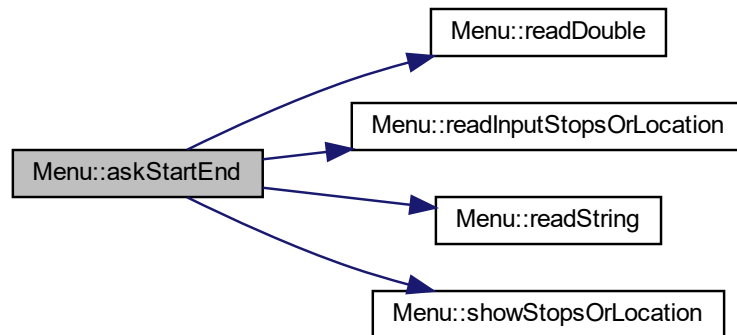
Asks the user the codes or coordinates of their wanted origin and destination. If the user inserts coordinates that don't belong to any stop and refuses to walk, the system warns them about it.

Parameters

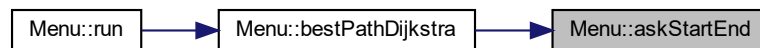
<i>stopBegin</i>	The origin's code
<i>stopEnd</i>	The destiny's code
<i>cBegin</i>	The origin's coordinates
<i>cEnd</i>	The destiny's coordinates

Returns

Here is the call graph for this function:



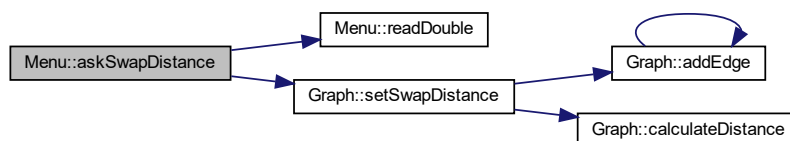
Here is the caller graph for this function:



3.6.1.4 askSwapDistance()

```
void Menu::askSwapDistance ( )
```

Asks the user how much they are willing to walk to swap buses in the middle of their journey. Here is the call graph for this function:



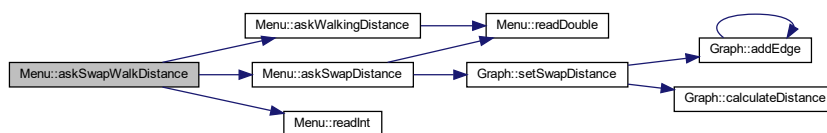
Here is the caller graph for this function:



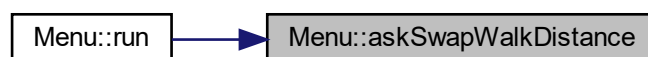
3.6.1.5 askSwapWalkDistance()

```
void Menu::askSwapWalkDistance ( )
```

Ask the user which of the two types of walking distances they want to set. Here is the call graph for this function:



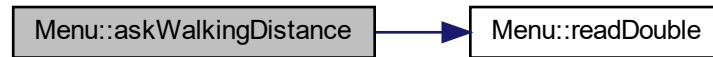
Here is the caller graph for this function:



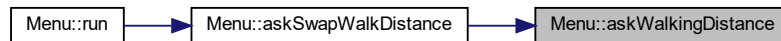
3.6.1.6 askWalkingDistance()

```
void Menu::askWalkingDistance ( )
```


Asks the user how much they are willing to walk to either their first stop or from their last stop to destiny if they give coordinates instead of the stops. Here is the call graph for this function:



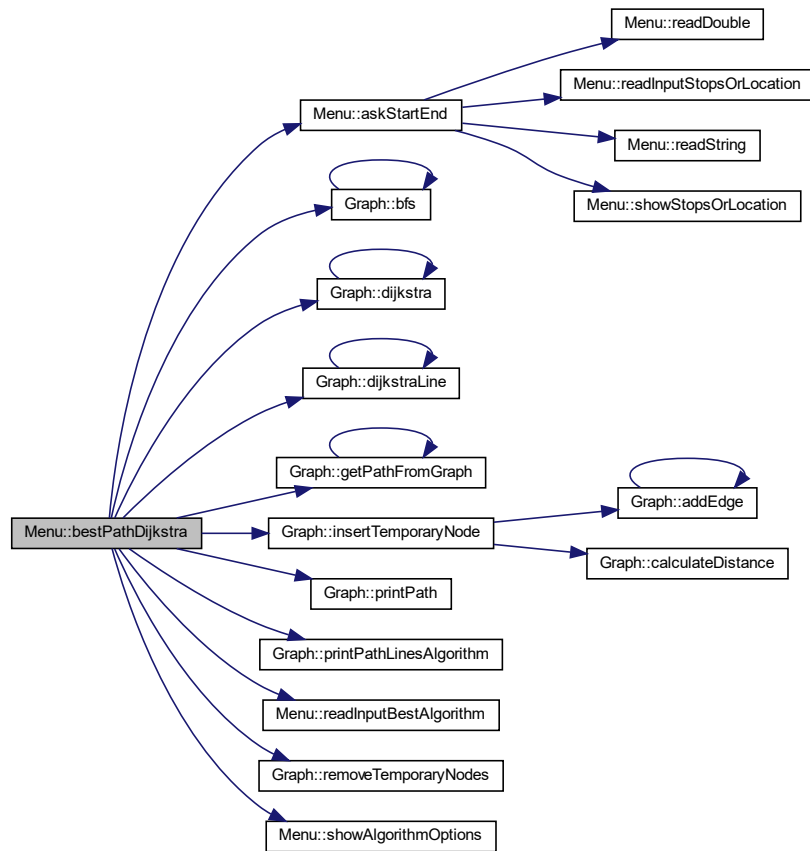
Here is the caller graph for this function:



3.6.1.7 bestPathDijkstra()

```
void Menu::bestPathDijkstra ( )
```

Method that deals with all the calculations of paths. Here is the call graph for this function:



Here is the caller graph for this function:



3.6.1.8 clear()

```
void Menu::clear ( ) [static]
```

It clears the console by calling `system()`. (It might have problems in Clion, but it works well in the terminal) Here is the caller graph for this function:



3.6.1.9 createGraphLine()

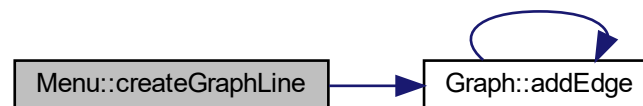
```
void Menu::createGraphLine (
    string line )
```

Reads a string representing the attributes of a line with them, creates a new line. It also adds this new line to every edges that it passes through.

Parameters

<i>line</i>	Attributes of a line
-------------	----------------------

Here is the call graph for this function:



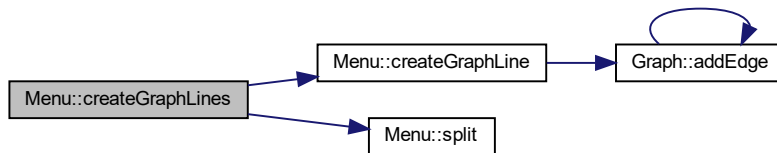
Here is the caller graph for this function:



3.6.1.10 createGraphLines()

```
void Menu::createGraphLines ( )
```

Read lines.csv and for every line on the document calls the method responsible for creating lines. Here is the call graph for this function:



3.6.1.11 createGraphStops()

```
void Menu::createGraphStops ( )
```

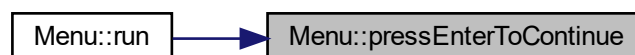
Reads stops.csv and for every line on the document it creates a node and adds it to the graph. Here is the call graph for this function:



3.6.1.12 pressEnterToContinue()

```
void Menu::pressEnterToContinue ( ) [static]
```

Waits for user to press enter to continue. Here is the caller graph for this function:



3.6.1.13 readDouble()

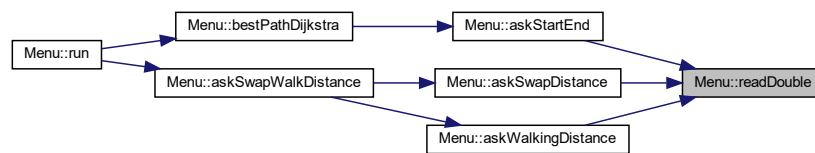
```
double Menu::readDouble ( ) [static]
```

Reads double from the user (General use).

Returns

Value given by the user

Here is the caller graph for this function:



3.6.1.14 readInputBestAlgorithm()

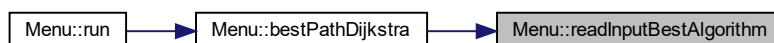
```
int Menu::readInputBestAlgorithm ( ) [static]
```

Read an input from the user regarding their algorithm of choice.

Returns

The chosen algorithm's number

Here is the caller graph for this function:



3.6.1.15 readInputMenu()

```
int Menu::readInputMenu ( ) [static]
```

Reads input from the user in the main menu.

Returns

choice of the user

Here is the caller graph for this function:



3.6.1.16 readInputStopsOrLocation()

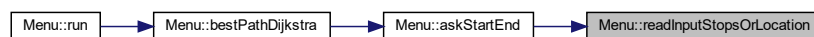
```
int Menu::readInputStopsOrLocation ( ) [static]
```

Reads an input from the user regarding if they want to tell the stops' code or coordinates.

Returns

The value of the user's choice.

Here is the caller graph for this function:



3.6.1.17 readInt()

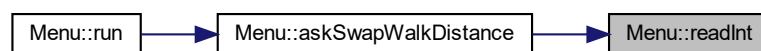
```
int Menu::readInt ( ) [static]
```

Reads integer from the user (General use).

Returns

Value given by the user

Here is the caller graph for this function:



3.6.1.18 readString()

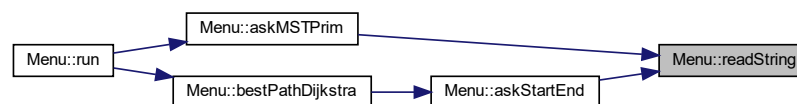
```
string Menu::readString ( ) [static]
```

Reads string from the user (General use).

Returns

Value given by the user

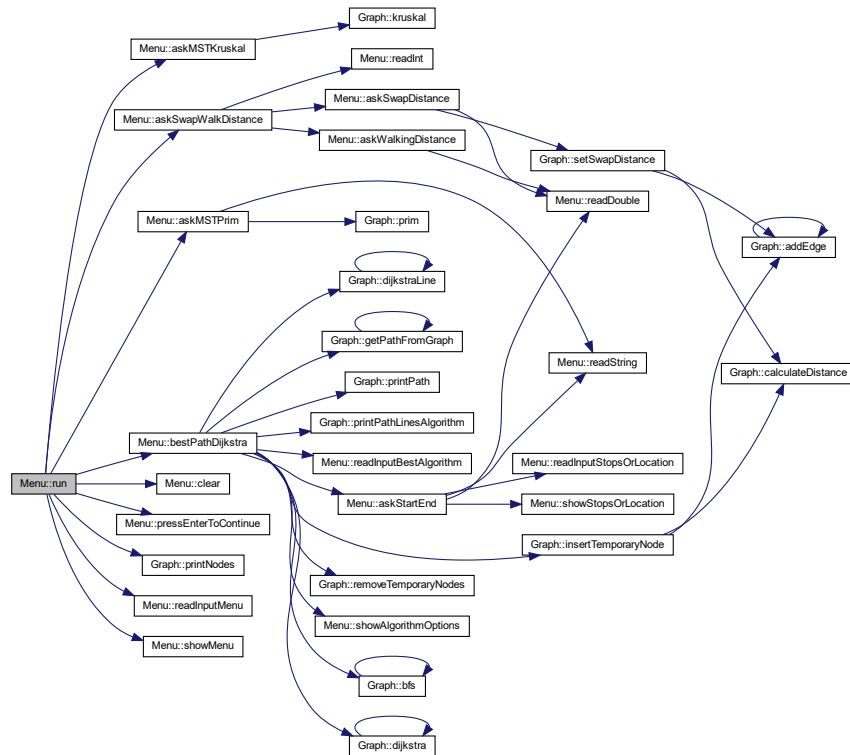
Here is the caller graph for this function:



3.6.1.19 run()

```
void Menu::run ( )
```

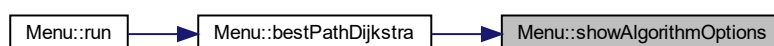
Takes the user's input to determine which functionality to execute. Here is the call graph for this function:



3.6.1.20 showAlgorithmOptions()

```
void Menu::showAlgorithmOptions ( ) [static]
```

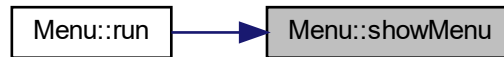
Prints the options of the priorities that can define "the best path" for the user. Here is the caller graph for this function:



3.6.1.21 showMenu()

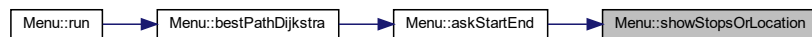
```
void Menu::showMenu ( ) [static]
```

Prints the main menu showing every functionality that the user can choose. Here is the caller graph for this function:

**3.6.1.22 showStopsOrLocation()**

```
void Menu::showStopsOrLocation ( ) [static]
```

Prints the options to indicate the stops' code or coordinates. Here is the caller graph for this function:

**3.6.1.23 split()**

```
vector< string > Menu::split (
    string line,
    string delimiter ) [static]
```

It separates a string in to a vector of strings by the delimiter.

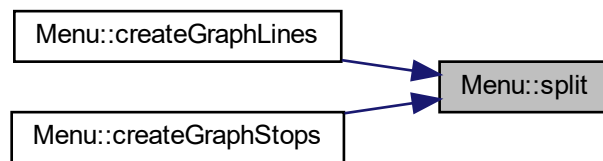
Parameters

<i>line</i>	The line's code
<i>delimiter</i>	The expression where the systems splits the string

Returns

Vector of strings

Here is the caller graph for this function:

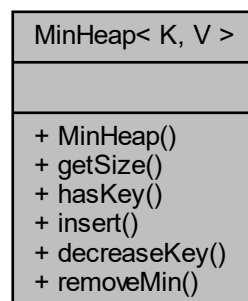


The documentation for this class was generated from the following files:

- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Menu.h
- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Menu.cpp

3.7 MinHeap< K, V > Class Template Reference

Collaboration diagram for MinHeap< K, V >:



Public Member Functions

- **MinHeap** (int n, const K ¬Found)
- int **getSize** ()
- bool **hasKey** (const K &key)
- void **insert** (const K &key, const V &value)
- void **decreaseKey** (const K &key, const V &value)
- K **removeMin** ()

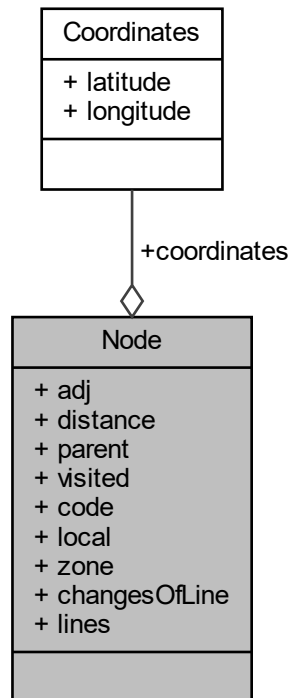
The documentation for this class was generated from the following file:

- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/MinHeap.h

3.8 Node Struct Reference

```
#include <Node.h>
```

Collaboration diagram for Node:



Public Attributes

- list< [Edge](#) > **adj**
- double **distance**
- int **parent**
- bool **visited**
- string **code**
- string **local**
- string **zone**
- [Coordinates](#) **coordinates**
- int **changesOfLine**
- map< string, pair< double, int > > **lines**

3.8.1 Detailed Description

A [Node](#) represents a Bus Stop

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

- C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Node.h

Chapter 4

File Documentation

4.1 DisjointSets.h

```
1 //
2 // Created by johnny on 29/01/22.
3 //
4
5 #ifndef _DISJOINTSETS_H_
6 #define _DISJOINTSETS_H_
7 #include <unordered_map>
8
9 template <class T> class DisjointSets {
10     struct Node {
11         T parent;
12         int myrank; // to use on union by rank
13     };
14
15     std::unordered_map<T, Node> a;
16
17 public:
18     void makeSet(const T &x); // Create a set with a single element x
19     T find(const T &x);      // Find the representative of the set of element x
20     void unite(const T &x, const T &y); // Merge the sets of elements x and y
21 };
22
23 // Create a set with a single element x
24 template <class T> void DisjointSets<T>::makeSet(const T &x) {
25     a[x].parent = x;
26     a[x].myrank = 0;
27 }
28
29 // Find the representative of the set of element x
30 template <class T> T DisjointSets<T>::find(const T &x) {
31     if (a[x].parent != x)
32         a[x].parent = find(a[x].parent);
33
34     return a[x].parent;
35 }
36
37 // Merge the sets of elements x and y
38 template <class T> void DisjointSets<T>::unite(const T &x, const T &y) {
39     T xRoot = find(x);
40     T yRoot = find(y);
41
42     if (xRoot == yRoot)
43         return;
44
45     if (a[xRoot].myrank < a[yRoot].myrank)
46         a[xRoot].parent = yRoot;
47     else if (a[xRoot].myrank > a[yRoot].myrank)
48         a[yRoot].parent = xRoot;
49     else {
50         a[yRoot].parent = xRoot;
51         ++(a[xRoot].myrank);
52     }
53 }
54
55 #endif
```

4.2 Graph.h

```

1 #pragma once
2 #ifndef BUSNAVIGATIONPROGRAM_GRAPH_H
3 #include <list>
4 #include <stack>
5 #include <vector>
6 #include <queue>
7 #include <iostream>
8 #include <unordered_map>
9 #include <cmath>
10 #include "Node.h"
11 #include "MinHeap.h"
12 #include "DisjointSets.h"
13
14 using namespace std;
15
16 class Graph {
17     int n; // Graph size (vertices are numbered from 1 to n)
18     double walkingDistance = 0.001; //km
19     double swapDistance = 0.0;
20     bool hasDir; // false: undirect; true: directed
21     vector<Node> nodes; // The list of nodes being represented
22     unordered_map<string, int> positions;
23
24 public:
25     // Constructor: nr nodes and direction (default: directed)
26     Graph(int nodes = 0, bool dir = true);
27     Graph(vector<Node> nodes, unordered_map<string, int> positions, bool dir = true);
28     vector<Node>& getNodes();
29     double getWalkingDistance();
30     double getSwapDistance();
31     unordered_map<string, int> getPositions();
32     void setWalkingDistance(double walkingDist);
33     void setSwapDistance(double swapDist);
34
35     // Add edge from source to destination with a certain weight
36     void addEdge(string src, string dest, string line);
37     void addEdge(int src, int dest, string line);
38     void addEdge(int src, int dest, double weight = 1.0);
39     double calculateDistance(Coordinates c1, Coordinates c2);
40     double calculateDistance(int src, int dest);
41     int calculateChangesOfLine(map<string, pair<double, int> lines, int numPrevChangesOfLines, string
42     newLine);
43     double calculateWeightDijkstraLine(int min, double edgeWeight, string newLine);
44
45     long double prim(int r);
46     long double kruskal();
47
48     double bfs(string src, string dest);
49     double bfs(int a, int b);
50     double dijkstra(string src, string dest);
51     double dijkstra(int a, int b);
52     double dijkstraLine(string src, string dest);
53     double dijkstraLine(int a, int b);
54
55     stack<int> getPathFromGraph(string src, string dest);
56     stack<int> getPathFromGraph(int a, int b);
57
58     void printPath(stack<int> path);
59     void printPathLinesAlgorithm(stack<int> path);
60     void printNodes();
61     void insertTemporaryNode(Coordinates c, bool startType);
62     void removeTemporaryNodes();
63 };
64
65
66 #endif //BUSNAVIGATIONPROGRAM_GRAPH_H

```

4.3 Menu.h

```

1 #ifndef BUSNAVIGATIONPROGRAM_MENU_H
2 #define BUSNAVIGATIONPROGRAM_MENU_H
3
4 #include <string>
5 #include <vector>
6 #include <unordered_map>
7 #include <fstream>
8 #include <iostream>
9 #include "Graph.h"
10
11 using namespace std;

```

```

12
13 class Menu {
14     string directory;
15     Graph graph;
16
17 public:
18     Menu(string directory);
19     Graph& getGraph();
20
21     void run();
22     static void showMenu();
23     static void showStopsOrLocation();
24     static void showAlgorithmOptions();
25
26     static int readInputMenu();
27     static int readInputStopsOrLocation();
28     static int readInputBestAlgorithm();
29     static double readDouble();
30     static int readInt();
31     static string readString();
32     static void pressEnterToContinue();
33
34     void createGraphStops();
35     void createGraphLines();
36     void createGraphLine(string line);
37
38     int askStartEnd(string& stopBegin, string& stopEnd, Coordinates& cBegin, Coordinates& cEnd);
39
40     void bestPathDijkstra();
41     void askMSTPrim();
42     void askMSTKruskal();
43
44     void askSwapWalkDistance();
45     void askWalkingDistance();
46     void askSwapDistance();
47
48     static vector<string> split(string line, string delimiter);
49     static void clear();
50 };
51
52
53 #endif //BUSNAVIGATIONPROGRAM_MENU_H

```

4.4 MinHeap.h

```

1 #ifndef BUSNAVIGATIONPROGRAM_MINHEAP_H
2 #define BUSNAVIGATIONPROGRAM_MINHEAP_H
3
4
5 #include <vector>
6 #include <unordered_map>
7
8 #define LEFT(i) (2*(i))
9 #define RIGHT(i) (2*(i)+1)
10 #define PARENT(i) ((i)/2)
11
12 using namespace std;
13
14 // Binary min-heap to represent integer keys of type K with values (priorities) of type V
15 template <class K, class V>
16 class MinHeap {
17     struct Node { // An element of the heap: a pair (key, value)
18         K key;
19         V value;
20     };
21
22     int size; // Number of elements in heap
23     int maxSize; // Maximum number of elements in heap
24     vector<Node> a; // The heap array
25     unordered_map<K, int> pos; // maps a key into its position on the array a
26     const K KEY_NOT_FOUND;
27
28     void upHeap(int i);
29     void downHeap(int i);
30     void swap(int i1, int i2);
31
32 public:
33     MinHeap(int n, const K& notFound); // Create a min-heap for a max of n pairs (K,V) with notFound
34     // returned when empty
35     int getSize(); // Return number of elements in the heap
36     bool hasKey(const K& key); // Heap has key?
37     void insert(const K& key, const V& value); // Insert (key, value) on the heap
38     void decreaseKey(const K& key, const V& value); // Decrease value of key

```

```

38     K removeMin(); // remove and return key with smaller value
39 };
40
41 // -----
42
43 // Make a value go "up the tree" until it reaches its position
44 template <class K, class V>
45 void MinHeap<K,V>::upHeap(int i) {
46     while (i>1 && a[i].value < a[PARENT(i)].value) { // while pos smaller than parent, keep swapping to
        upper position
47         swap(i, PARENT(i));
48         i = PARENT(i);
49     }
50 }
51
52 // Make a value go "down the tree" until it reaches its position
53 template <class K, class V>
54 void MinHeap<K,V>::downHeap(int i) {
55     while (LEFT(i) <= size) { // while within heap limits
56         int j = LEFT(i);
57         if (RIGHT(i)<=size && a[RIGHT(i)].value < a[j].value) j = RIGHT(i); // choose smaller child
58         if (a[i].value < a[j].value) break; // node already smaller than children, stop
59         swap(i, j); // otherwise, swap with smaller child
60         i = j;
61     }
62 }
63
64 // Swap two positions of the heap (update their positions)
65 template <class K, class V>
66 void MinHeap<K,V>::swap(int i1, int i2) {
67     Node tmp = a[i1]; a[i1] = a[i2]; a[i2] = tmp;
68     pos[a[i1].key] = i1;
69     pos[a[i2].key] = i2;
70 }
71
72 // -----
73
74 // Create a min-heap for a max of n pairs (K,V) with notFound returned when empty
75 template <class K, class V>
76 MinHeap<K,V>::MinHeap(int n, const K& notFound) : KEY_NOT_FOUND(notFound), size(0), maxSize(n), a(n+1) {
77 }
78
79 // Return number of elements in the heap
80 template <class K, class V>
81 int MinHeap<K,V>::getSize() {
82     return size;
83 }
84
85
86 // Heap has key?
87 template <class K, class V>
88 bool MinHeap<K, V>::hasKey(const K& key) {
89     return pos.find(key) != pos.end();
90 }
91
92 // Insert (key, value) on the heap
93 template <class K, class V>
94 void MinHeap<K,V>::insert(const K& key, const V& value) {
95     if (size == maxSize) return; // heap is full, do nothing
96     if (hasKey(key)) return; // key already exists, do nothing
97     a[++size] = {key, value};
98     pos[key] = size;
99     upHeap(size);
100 }
101
102 // Decrease value of key to the indicated value
103 template <class K, class V>
104 void MinHeap<K,V>::decreaseKey(const K& key, const V& value) {
105     if (!hasKey(key)) return; // key does not exist, do nothing
106     int i = pos[key];
107     if (value > a[i].value) return; // value would increase, do nothing
108     a[i].value = value;
109     upHeap(i);
110 }
111
112 // remove and return key with smaller value
113 template <class K, class V>
114 K MinHeap<K,V>::removeMin() {
115     if (size == 0) return KEY_NOT_FOUND;
116     K min = a[1].key;
117     pos.erase(min);
118     a[1] = a[size--];
119     downHeap(1);
120     return min;
121 }
122
123 #endif //BUSNAVIGATIONPROGRAM_MINHEAP_H

```


4.5 Node.h

```
1 #ifndef BUSNAVIGATIONPROGRAM_NODE_H
2 #define BUSNAVIGATIONPROGRAM_NODE_H
3
4 #include <list>
5 #include <string>
6 #include <map>
7
8 using namespace std;
9
10
11 struct Coordinates {
12     double latitude;
13     double longitude;
14 };
15
16 struct Edge {
17     int dest; // Destination node
18     double weight; // An integer weight
19     string line;
20 };
21
22 struct EdgeKruskal {
23     int src; // Source node
24     int dest; // Destination node
25     double weight; // An integer weight
26
27     // For Kruskal's
28     bool operator<(const EdgeKruskal &other) const {
29         return weight < other.weight;
30     }
31
32     bool operator==(const EdgeKruskal &other) const {
33         return ((src == other.src && dest == other.dest) ||
34             (src == other.dest && dest == other.src)) &&
35             weight == other.weight;
36     }
37 };
38
39 struct Node {
40     list<Edge> adj; // The list of outgoing edges (to adjacent nodes)
41     double distance;
42     int parent; // previous node of the path
43     bool visited;
44     string code; //unique code of the bus stop
45     string local;
46     string zone;
47     Coordinates coordinates;
48     int changesOfLine; //used in dijkstraLine algorithm calculation
49     map<string, pair<double, int> lines; // used for dijkstraLine algorithm calculations
50 };
51
52 #endif //BUSNAVIGATIONPROGRAM_NODE_H
```


Index

addEdge
 Graph, [9](#), [10](#)
askMSTKruskal
 Menu, [25](#)
askMSTPrim
 Menu, [25](#)
askStartEnd
 Menu, [26](#)
askSwapDistance
 Menu, [27](#)
askSwapWalkDistance
 Menu, [28](#)
askWalkingDistance
 Menu, [28](#)

bestPathDijkstra
 Menu, [29](#)
bfs
 Graph, [11](#)

C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/DisjointSets.h,
 [41](#)
C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/DisjointSets.h,
 [42](#)
C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Menu.h,
 [42](#)
C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/MinHeap.h,
 [43](#)
C:/clion-projetos/BusNavigationProgram/BusNavigationProgram/src/Node.h,
 [45](#)

calculateChangesOfLine
 Graph, [12](#)
calculateDistance
 Graph, [13](#)
calculateWeightDijkstraLine
 Graph, [14](#)
clear
 Menu, [30](#)
Coordinates, [5](#)
createGraphLine
 Menu, [31](#)
createGraphLines
 Menu, [31](#)
createGraphStops
 Menu, [32](#)

dijkstra
 Graph, [15](#)
dijkstraLine
 Graph, [16](#), [17](#)

DisjointSets< T >, [6](#)

Edge, [6](#)
EdgeKruskal, [7](#)

getPathFromGraph
 Graph, [18](#)
Graph, [8](#)
 addEdge, [9](#), [10](#)
 bfs, [11](#)
 calculateChangesOfLine, [12](#)
 calculateDistance, [13](#)
 calculateWeightDijkstraLine, [14](#)
 dijkstra, [15](#)
 dijkstraLine, [16](#), [17](#)
 getPathFromGraph, [18](#)
 insertTemporaryNode, [19](#)
 kruskal, [20](#)
 prim, [20](#)
 printNodes, [21](#)
 printPath, [21](#)
 printPathLinesAlgorithm, [22](#)
 removeTemporaryNodes, [22](#)
 setSwapDistance, [23](#)
 insertTemporaryNode
 Graph, [19](#)
 kruskal
 Graph, [20](#)

Menu, [24](#)
 askMSTKruskal, [25](#)
 askMSTPrim, [25](#)
 askStartEnd, [26](#)
 askSwapDistance, [27](#)
 askSwapWalkDistance, [28](#)
 askWalkingDistance, [28](#)
 bestPathDijkstra, [29](#)
 clear, [30](#)
 createGraphLine, [31](#)
 createGraphLines, [31](#)
 createGraphStops, [32](#)
 pressEnterToContinue, [32](#)
 readDouble, [32](#)
 readInputBestAlgorithm, [33](#)
 readInputMenu, [33](#)
 readInputStopsOrLocation, [34](#)
 readInt, [34](#)
 readString, [35](#)
 run, [35](#)

- showAlgorithmOptions, [36](#)
 - showMenu, [36](#)
 - showStopsOrLocation, [37](#)
 - split, [37](#)
- MinHeap< K, V >, [38](#)
- Node, [39](#)
- pressEnterToContinue
 - Menu, [32](#)
- prim
 - Graph, [20](#)
- printNodes
 - Graph, [21](#)
- printPath
 - Graph, [21](#)
- printPathLinesAlgorithm
 - Graph, [22](#)
- readDouble
 - Menu, [32](#)
- readInputBestAlgorithm
 - Menu, [33](#)
- readInputMenu
 - Menu, [33](#)
- readInputStopsOrLocation
 - Menu, [34](#)
- readInt
 - Menu, [34](#)
- readString
 - Menu, [35](#)
- removeTemporaryNodes
 - Graph, [22](#)
- run
 - Menu, [35](#)
- setSwapDistance
 - Graph, [23](#)
- showAlgorithmOptions
 - Menu, [36](#)
- showMenu
 - Menu, [36](#)
- showStopsOrLocation
 - Menu, [37](#)
- split
 - Menu, [37](#)