Graph-Floyd-Warshall

1.0

Gerado por Doxygen 1.9.5

Capítulo 1

Índice dos Componentes

1.1 Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

Capítulo 2

Índice dos Arquivos

2.1 Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

Capítulo 3

Classes

3.1 Referência da Classe Graph

Estrutura do Grafo.

`#include <graph.hpp>`

Membros Públicos

- Graph (string name_file)
  Construtor da classe Graph.
- ~Graph ()
  Destrutor da classe Graph.
- Matrix getMatrixAdj ()
  get matrix_adj.
- Matrix getMatrixFinal ()
  get matrix_final
- int getEdgesSize ()
  get count_edges.
- string getNameVertex (int index)
  get Vertex::getNameVertex().
- float getWeightIndex (int i1, int i2)
  Busca o peso de uma aresta entre dois vértices na matriz de adjacência.
- void PrintVertex ()
  Imprime o vertor de Vértices (vertex).
- void PrintMatrix (Matrix m)
  Imprime uma matriz.
- void ReadFileConections (string name_file)
  Faz as conexões conforme o arquivo de entrada.
- void MakeFloydWarshall ()
  Executa as operações para gerar uma matriz de pesos e uma matriz de caminhos.
- void MakePath_Sequential (int index1, int index2)
  Faz o caminho.
- void MakeMinPath3_Sequential (string name1, string name2, string name3)
  Encontra o menor caminho para se ir a 3 vértices diferentes sequencialmente.

Membros Privados

- int getIndexVertex (string name)

  get do índice do vértice escolhido no vetor de vértices (vertex).
- void MatrixAdjNull (int size)

  Gera a Matriz de Adjacência nula para as futuras conexões.
- void MakeConection (string name1, string name2, float weight)

  Faz a conexão entre dois vértices.
- void UpdateGrade (int i1, int i2)

  Faz a atualização dos graus dos vértices.
- void ShortPath_Sequential (int index1, int index2)

  Encontra o menor caminho por meio da matriz de caminhos.

Atributos Privados

- int size_graph
- int count_edges
- vector< Vertex > vertex
- Matrix matrix_adj
- Matrix matrix_final

3.1.1 Descrição detalhada

Estrutura do Grafo.

3.1.2 Construtores e Destrutores

3.1.2.1 Graph()

```
Graph::Graph (
            string name_file )
```

Construtor da classe Graph.

- Faz a leitura do arquivo com o nome de todos os vertices.
- Armazena os vertices no vetor "vertex".
- Cria a matriz de adjacência com todas as posições nula.

Parâmetros

name_file	= Nome do aquivo com a extensão (.txt).

```
101                             {
102
```

```
103     string str_file = "./src/input/";
104     str_file.append(name_file);
105     string str_token;
106 
107     cout « str_file « endl;
108     ifstream file(str_file);
109 
110     if(file.is_open()){
111 
112         while(getline(file,str_token)){
113 
114             Vertex v = Vertex(str_token);
115             vertex.push_back(v);
116         }
117 
118         this->size_graph = vertex.size();
119         this->count_edges = 0;
120 
121     }else{
122 
123         cout « "ERRO - " « name_file « endl « endl;
124     }
125 
126     file.close();
127 
128     MatrixAdjNull(size_graph);
129 }
```

3.1.2.2 ~Graph()

```
Graph::~Graph ( )
```

Destrutor da classe Graph.

```
134             {
135     //Destrutor
136 }
```

3.1.3 Funções membros

3.1.3.1 getEdgesSize()

```
int Graph::getEdgesSize ( )
```

get count_edges.

Retorna

count_edges - Quantidade de arestas no grafo.

```
144                     {
145 
146     return count_edges;
147 }
```

3.1.3.2 getIndexVertex()

```
int Graph::getIndexVertex (
             string name )  [private]
```

get do índice do vértice escolhido no vetor de vértices (vertex).

Parâmetros

name	= Nome do vértice

Retorna

índice do vértice escolhido.

```
282                                     {
283
284     for(int i = 0; i < vertex.size(); i++){
285
286         if(vertex[i].getNameVertex() == name){
287
288             return i;
289         }
290     }
291
292     return -1;
293 }
```

3.1.3.3 getMatrixAdj()

```
Matrix Graph::getMatrixAdj ( )
```

get matrix_adj.

Retorna

matrix_adj

```
177                             {
178
179     return matrix_adj;
180 }
```

3.1.3.4 getMatrixFinal()

```
Matrix Graph::getMatrixFinal ( )
```

get matrix_final

Retorna

matrix_final

```
187                               {
188
189     return matrix_final;
190 }
```

3.1.3.5 getNameVertex()

```
string Graph::getNameVertex (
              int index )
```

get Vertex::getNameVertex().

- Procura o vértice no vetor vertex.

Parâmetros

index	= Indice do vértice.

Retorna

Nome do vertíce do indice de parâmetro.

```
155                                     {
156
157     return vertex[index].getNameVertex();
158 }
```

3.1.3.6 getWeightIndex()

```
float Graph::getWeightIndex (
            int i1,
            int i2 )
```

Busca o peso de uma aresta entre dois vértices na matriz de adjacência.

Parâmetros

i1	= Índice do primeiro vértice
i2	= Índice do segundo vértice.

Retorna

peso da aresta de conexção entre os dois vertíces.

```
167                                         {
168
169     return matrix_adj[i1][i2];
170 }
```

3.1.3.7 MakeConection()

```
void Graph::MakeConection (
            string name1,
            string name2,
            float weight )  [private]
```

Faz a conexão entre dois vértices.

- Gera uma aresta entre dois vértices.

Parâmetros

name1	= Nome do 1º vértice.
name2	= Nome do 2º vértice.
weight	= Peso da Aresta entre os dois vértices.

```
262                                                                 {
263
264     int index_v1 = getIndexVertex(name1);
265     int index_v2 = getIndexVertex(name2);
266
267     if(index_v1 != -1 && index_v2 != -1){
268
269         matrix_adj[index_v1][index_v2] = weight;
270         count_edges++;
271         UpdateGrade(index_v1, index_v2);
272     }
273
274 }
```

3.1.3.8 MakeFloydWarshall()

```
void Graph::MakeFloydWarshall ( )
```

Executa as operações para gerar uma matriz de pesos e uma matriz de caminhos.

Parâmetros

matrix_adj	= matriz de pesos.
matrix_final	= matriz de caminhos.

```
376                             {
377
378     int N = vertex.size();
379
380     //Nessa parte analisamos a ditância entre um ponto intermediario k, entre i e j
381     for(int k = 0; k < N; k++){
382         for(int i = 0; i < N; i++){
383             for(int j = 0; j < N; j++){
384
385                 if(matrix_adj[i][j] > matrix_adj[i][k]+matrix_adj[k][j]){
386
387                     matrix_adj[i][j] = matrix_adj[i][k] + matrix_adj[k][j];
388                     matrix_final[i][j] = k;
389                 }
390             }
391         }
392     }
393 }
```

3.1.3.9 MakeMinPath3_Sequential()

```
void Graph::MakeMinPath3_Sequential (
              string name1,
              string name2,
              string name3 )
```

Encontra o menor caminho para se ir a 3 vértices diferentes sequencialmente.

Parâmetros

name1	= Nome do 1º vértice.
name2	= Nome do 2º vértice.
name3	= Nome do 3º vértice.

```
436                                                                     {
437
438     int index1 = getIndexVertex(name1);
439     int index2 = getIndexVertex(name2);
440     int index3 = getIndexVertex(name3);
441     int index_loja = getIndexVertex(LOJA);
442
443     float custo_t = 0.0;
444
445     MakePath_Sequential(index_loja, index1);
446     cout « endl;
447     custo_t += getWeightIndex(index_loja,index1);
448     MakePath_Sequential(index1, index2);
449     cout « endl;
450     custo_t += getWeightIndex(index1,index2);
451     MakePath_Sequential(index2, index3);
452     cout « endl;
453     custo_t += getWeightIndex(index2,index3);
454
455     cout « "\nCusto Total = " « custo_t « " Km" « endl;
456 }
```

3.1.3.10 MakePath_Sequential()

```
void Graph::MakePath_Sequential (
            int index1,
            int index2 )
```

Faz o caminho.

- Função de auxilio de Graph::ShortPath_Sequential.

Parâmetros

name1	= Nome do vértice inicial.
name2	= = Nome do vértice inicial. • Do vértice de onde se está, para o vértice que quer ir.

```
422                                                 {
423
424     cout« "\n[" « getNameVertex(index1) « "] para [" «
425     getNameVertex(index2) « "]: " « getNameVertex(index1) « " --> ";
426     ShortPath_Sequential(index1, index2);
427 }
```

3.1.3.11 MatrixAdjNull()

```
void Graph::MatrixAdjNull (
            int size )  [private]
```

Gera a Matriz de Adjacência nula para as futuras conexões.

Parâmetros

size	= Tamanho da matriz (size x size).

```
197                                 {
198
199     vector<float> aux1, aux2;
200     for(int i = 0; i < size; i++){
201         aux1.push_back(INF);
202         aux2.push_back(-1);
203     }
204
205     for(int i = 0; i < size; i++){
206
207         matrix_adj.push_back(aux1);
208         matrix_final.push_back(aux2);
209     }
210
211     for(int i = 0; i < size; i++){
212
213         matrix_adj[i][i] = 0.0;
214     }
215 }
```

3.1.3.12 PrintMatrix()

```
void Graph::PrintMatrix (
              Matrix m )
```

Imprime uma matriz.

Parâmetros

m	= Matriz que se deseja imprimir.

```
222                                 {
223
224     int size = vertex.size();
225
226     for(int i = 0; i < size; i++){
227
228         for(int j = 0; j < size; j++){
229
230             if(m[i][j] == INF){
231
232                 cout « 0.0 « "    ";
233
234             }else{
235
236                 cout « m[i][j] « "  ";
237             }
238         }
239         cout « endl;
240     }
241 }
```

3.1.3.13 PrintVertex()

```
void Graph::PrintVertex ( )
```

Imprime o vertor de Vértices (vertex).

```
246                         {
247
248     for(int i = 0; i < vertex.size(); i++){
249
250         cout « vertex[i].getNameVertex() « " - " « vertex[i].getGrade() « endl;
251     }
252 }
```

3.1.3.14 ReadFileConections()

```
void Graph::ReadFileConections (
            string name_file )
```

Faz as conexões conforme o arquivo de entrada.

- Formato: Vértice1/Vértice2/peso_da_aresta.

Parâmetros

name_file	= Nome do aquivo com a extensão (.txt).

```
317                                                 {
318
319     string str_file = "./src/input/";
320     str_file.append(name_file);
321
322     ifstream file(str_file);
323
324     if(file.is_open()){
325
326          string line_token, token;
327         char del = '/';
328
329         while(getline(file,line_token)){
330
331             stringstream sstream(line_token);
332             string str1, str2;
333             float w_ref;
334
335             int control = 0;
336
337             while(getline(sstream,token,del)){
338
339                 switch(control){
340
341                     case 0:
342                         str1 = token;
343                     break;
344
345                     case 1:
346                         str2 = token;
347                     break;
348
349                     case 2:
350                         w_ref = stof(token);
351                     break;
352                 }
353
354                 control++;
355             }
356
357             control = 0;
358
359             MakeConection(str1, str2, w_ref);
360             MakeConection(str2, str1, w_ref);
361         }
362
363     }else{
364
365         cout « "ERRO - " « name_file « endl;
366     }
367 }
```

3.1.3.15 ShortPath_Sequential()

```
void Graph::ShortPath_Sequential (
            int index1,
            int index2 )  [private]
```

Encontra o menor caminho por meio da matriz de caminhos.

- Método recursivo.

Parâmetros

index1	= Índice do vértice inicial.
index2	= Índice do vértice final.

```
402                                                     {
403
404     if(matrix_final[index1][index2] == -1){
405
406         cout « vertex[index2].getNameVertex();
407         return;
408     }
409     ShortPath_Sequential(index1,matrix_final[index1][index2]);
410     cout « " --> ";
411     ShortPath_Sequential(matrix_final[index1][index2],index2);
412 }
```

3.1.3.16 UpdateGrade()

```
void Graph::UpdateGrade (
              int i1,
              int i2 )  [private]
```

Faz a atualização dos graus dos vértices.

Parâmetros

i1	= Índice do vertice de saída.
i2	= Índice do vértice de entrada.

```
301                                     {
302
303     vertex[i1].setGradeOut(); // vertice do qual sai uma aresta.
304     vertex[i2].setGradeIn(); // vertice o qual chega uma aresta.
305
306     // Atualiza o grau total
307     vertex[i1].setGrade();
308     vertex[i2].setGrade();
309 }
```

3.1.4 Atributos

3.1.4.1 count_edges

```
int Graph::count_edges  [private]
```

3.1.4.2 matrix_adj

```
Matrix Graph::matrix_adj  [private]
```

3.1.4.3 matrix_final

```
Matrix Graph::matrix_final  [private]
```

3.1.4.4 size_graph

```
int Graph::size_graph  [private]
```

3.1.4.5 vertex

```
vector<Vertex> Graph::vertex  [private]
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- GitHub Repositories/Graph-Floyd-Warshall/code/include/graph.hpp
- GitHub Repositories/Graph-Floyd-Warshall/code/src/graph.cpp

3.2 Referência da Classe Vertex

Estrutura do Vértice.

```
#include <graph.hpp>
```

Membros Públicos

- Vertex (string name)
  Construtor da classe Vertex.
- ~Vertex ()
  Destrutor da classe Vertex.
- void setGradeIn ()
  set grande_input
- int getGradeIn ()
  get grade_input
- void setGradeOut ()
  set grade_output
- int getGradeOut ()
  get grade_output
- void setGrade ()
  set grade
- int getGrade ()
  get grade
- string getNameVertex ()
  get name_vertex

Atributos Privados

- string name_vertex
- int grade_input
- int grade_output
- int grade

3.2.1 Descrição detalhada

Estrutura do Vértice.

3.2.2 Construtores e Destrutores

3.2.2.1 Vertex()

```
Vertex::Vertex (
              string name )
```

Construtor da classe Vertex.

Parâmetros

name	= Nome do Vértice.

```
21                          {
22
23     this->name_vertex = name;
24     this->grade_input = 0;
25     this->grade_output = 0;
26     this->grade = 0;
27 }
```

3.2.2.2 ~Vertex()

```
Vertex::~Vertex ( )
```

Destrutor da classe Vertex.

```
32                  {
33     //Destrutor!
34 }
```

3.2.3 Funções membros

3.2.3.1 getGrade()

```
int Vertex::getGrade ( )
```

get grade

Retorna

return grade

```
82 {return grade;}
```

3.2.3.2 getGradeIn()

```
int Vertex::getGradeIn ( )
```

get grade_input

Retorna

grade_input

```
50 {return grade_input;}
```

3.2.3.3 getGradeOut()

```
int Vertex::getGradeOut ( )
```

get grade_output

Retorna

grade_output

```
66 {return grade_output;}
```

3.2.3.4 getNameVertex()

```
string Vertex::getNameVertex ( )
```

get name_vertex

- Nome do vértice

  Retorna

  name_vertex

```
89 {return name_vertex;}
```

3.2.3.5 setGrade()

```
void Vertex::setGrade ( )
```

set grade

- Grau to vertice = grade_input + grade_output

```
72                      {
73
74     this->grade = getGradeIn()+getGradeOut();
75 }
```

3.2.3.6 setGradeIn()

```
void Vertex::setGradeIn ( )
```

set grande_input

- Incrementa grau de entrada

```
40                       {
41
42     this->grade_input++;
43 }
```

3.2.3.7 setGradeOut()

```
void Vertex::setGradeOut ( )
```

set grade_output

- Incrementa grau de saída

```
56                        {
57
58     this->grade_output++;
59 }
```

3.2.4 Atributos

3.2.4.1 grade

```
int Vertex::grade  [private]
```

3.2.4.2 grade_input

```
int Vertex::grade_input  [private]
```

3.2.4.3 grade_output

```
int Vertex::grade_output  [private]
```

3.2.4.4 name_vertex

```
string Vertex::name_vertex  [private]
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- GitHub Repositories/Graph-Floyd-Warshall/code/include/graph.hpp
- GitHub Repositories/Graph-Floyd-Warshall/code/src/graph.cpp

Capítulo 4

Arquivos

4.1 Referência do Arquivo GitHub Repositories/Graph-Floyd-Warshall/code/include/graph.hpp

Arquivo cabeçalho.

```
#include <bits/stdc++.h>
```

Componentes

- class Vertex

  Estrutura do Vértice.
- class Graph

  Estrutura do Grafo.

Definições e Macros

- #define INF (float)DBL_MAX
- #define Matrix vector<vector<float>>
- #define LOJA (string)"Loja-SUPREMA"

4.1.1 Descrição detalhada

Arquivo cabeçalho.

Autor

Gabriel Alves (https://github.com/Nerd100oculoS)

Data

2022-11-19

Copyright

Copyright (c) 2022

4.1.2 Definições e macros

4.1.2.1 INF

```
#define INF (float)DBL_MAX
```

4.1.2.2 LOJA

```
#define LOJA (string)"Loja-SUPREMA"
```

4.1.2.3 Matrix

```
#define Matrix vector<vector<float>>
```

4.2 graph.hpp

Vá para a documentação desse arquivo.

```
1
11 #ifndef GRAPH_HPP
12 #define GRAPH_HPP
13
14 #define INF (float)DBL_MAX
15 #define Matrix vector<vector<float»
16 #define LOJA (string)"Loja-SUPREMA"
17
18 #include<bits/stdc++.h>
19
20 using namespace std;
21
25 class Vertex{
26
27     private:
28
29     //Attributes
30         string name_vertex;
31         int grade_input;
32         int grade_output;
33         int grade;
34
35     public:
36
37     // Constructor and Destructor
38         Vertex(string name);
39         ~Vertex();
40
41     //set's and get's
42         void setGradeIn();
43         int getGradeIn();
44
45         void setGradeOut();
46         int getGradeOut();
47
48         void setGrade();
49         int getGrade();
50         string getNameVertex();
51 };
52
53
```

```
57 class Graph{
58
59     private:
60
61     //Attributes
62         int size_graph; //quantidade de vertices no grafo.
63         int count_edges; // Quantidade de arestas no grafo todo
64         vector<Vertex> vertex; // Lista de vertices para se obter indice
65
66         Matrix matrix_adj; //Matriz de Adjacência
67         Matrix matrix_final; //Matriz de Pesos
68
69         int getIndexVertex(string name);
70
71     //Private methods
72         void MatrixAdjNull(int size);
73         void MakeConection(string name1, string name2, float weight);
74         void UpdateGrade(int i1, int i2);
75         void ShortPath_Sequential(int index1, int index2);
76
77     public:
78
79     //Constructor and Destructor
80         Graph(string name_file);
81         ~Graph();
82
83     //set's and get's
84         Matrix getMatrixAdj();
85         Matrix getMatrixFinal();
86
87         int getEdgesSize();
88         string getNameVertex(int index); //Procura o indice dentro do vector
89         float getWeightIndex(int i1, int i2);
90
91     //Public methods
92         void PrintVertex();
93         void PrintMatrix(Matrix m);
94         void ReadFileConections(string name_file);
95
96         void MakeFloydWarshall();
97
98         void MakePath_Sequential(int index1, int index2);
99         void MakeMinPath3_Sequential(string name1, string name2, string name3);
100 };
101
102 #endif
```

4.3 Referência do Arquivo GitHub Repositories/Graph-Floyd-Warshall/code/src/graph.cpp

Arquivo de Implementação.

```
#include "../include/graph.hpp"
```

4.3.1 Descrição detalhada

Arquivo de Implementação.

Autor

Gabriel Alves (https://github.com/Nerd100oculoS)

Versão

0.1

Data

2022-11-19

Copyright

Copyright (c) 2022

4.4 Referência do Arquivo GitHub Repositories/Graph-Floyd-Warshall/code/src/main.cpp

```
#include "../include/graph.hpp"
```

Funções

- int main ()

4.4.1 Funções

4.4.1.1 main()

```
int main ( )
3           {
4
5      Graph g = Graph("vertex_inputs.txt");
6      // g.PrintVertex(); //Para mostrar todos os bairros
7      g.ReadFileConections("edges_inputs.txt");
8      cout « "------------------------> Matriz de adjacencia inicialmente:\n";
9      g.PrintMatrix(g.getMatrixAdj());
10      cout « endl « endl;
11
12      g.MakeFloydWarshall();
13      cout « "------------------------> Matriz de adjacencia após FLOYD-WARSHALL:\n";
14      g.PrintMatrix(g.getMatrixAdj());
15
16      cout « "\n\n------------------------> Matriz de caminhos:\n";
17      g.PrintMatrix(g.getMatrixFinal());
18
19      cout « "\n\n\n-----------------> Aôp Pimenta!!\n\n";
20      cout « "<---------------------- Menor Caminho para atender clientes em 3 bairros diferentes
        sequencialmente ---------------------->\n\n";
21
22      string str_file = "./src/input/input_teste.txt";
23      ifstream file(str_file);
24
25      if(file.is_open()){
26
27           string line_token, token;
28          char del = ',';
29
30          while(getline(file,line_token)){
31
32              stringstream sstream(line_token);
33              string str1, str2, str3;
34
35              int control = 0;
36
37              while(getline(sstream,token,del)){
38
39                  switch(control){
```

```
40
41                      case 0:
42                          str1 = token;
43                      break;
44
45                      case 1:
46                          str2 = token;
47                      break;
48
49                      case 2:
50                          str3 = token;
51                      break;
52                  }
53
54                  control++;
55              }
56
57              control = 0;
58
59              cout « "\nClientes nos bairros: " « str1 « ", " « str2 « ", " « str3 « endl;
60              g.MakeMinPath3_Sequential(str1,str2,str3);
61              cout « "\n--------------------------------------------------------------------------------\n";
62
63          }
64
65      }else{
66
67          cout « "ERRO - " « str_file « endl;
68      }
69
70      return 0;
71  }
```

Índice Remissivo