AdjGraph

0.3.0

Generated by Doxygen 1.8.17

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Graph Class Reference

Public Member Functions

- Graph (vector< int > a, int input)
- void addEdge (vector< int > adj, int u, int v)
- void delEdge (vector< int > adj, int u, int v)
- bool hasEdge (int i, int j)
- void outEdges (int i, vector< int > &edges)
- void inEdges (int i, vector< int > &edges)
- int nVertices ()
- void bfs (Graph &g, int r)
- void dfs2 (Graph &g, int r)
- void printGraph (vector< int > adj, int V)

Public Attributes

- vector< int > adj

3.1.1 Detailed Description

Definition at line 17 of file main.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Graph()

```
Graph::Graph (
            vector< int > a,
            int input )  [inline]
```

Definition at line 23 of file main.cpp.

```
23                                  {
24         n = input;
25         adj = a;
26     }
```

3.1.3 Member Function Documentation

3.1.3.1 addEdge()

```
void Graph::addEdge (
            vector< int > adj,
            int u,
            int v )  [inline]
```

Definition at line 28 of file main.cpp.

```
29 {
30     adj.push_back(v);
31     adj.push_back(u);
32 }
```

3.1.3.2 bfs()

```
void Graph::bfs (
            Graph & g,
            int r )  [inline]
```

Definition at line 82 of file main.cpp.

```
82                          {
83     bool *seen = new bool[g.nVertices()];
84     vector<int> q;
85     q.push_back(r);
86     seen[r] = true;
87     while (q.size() > 0) {
88         int i = q.back();
89         cout « endl « i « " > " « "This is BFS" « endl;
90         q.pop_back();
91         vector<int> edges;
92         g.outEdges(i, edges);
93         for (int k = 0; k < edges.size(); k++) {
94             int j = edges[k];
95             if (!seen[j]) {
96                 q.push_back(j);
97                 seen[j] = true;
98             }
99         }
100     }
101     delete[] seen;
102 }
```

3.1.3.3 delEdge()

```
void Graph::delEdge (
            vector< int > adj,
            int u,
            int v )  [inline]
```

Definition at line 33 of file main.cpp.

```
34 {
35     // Traversing through the first vector list
36     // and removing the second element from it
37     for (int i = 0; i < adj.size(); i++) {
38         if (adj[i] == v) {
```

```
39              adj.erase(adj.begin() + i);
40              break;
41          }
42      }
43
44      // Traversing through the second vector list
45      // and removing the first element from it
46      for (int i = 0; i < adj.size(); i++) {
47          if (adj[i] == u) {
48              adj.erase(adj.begin() + i);
49              break;
50          }
51      }
52  }
```

3.1.3.4 dfs2()

```
void Graph::dfs2 (
              Graph & g,
              int r )  [inline]
```

Definition at line 106 of file main.cpp.

```
106                                 {
107      bool *c = new bool[g.nVertices()];
108      vector<int> s;
109      s.push_back(r);
110      while (s.size() > 0) {
111          int i = s.back();
112          cout << endl << i << " > " << "This is DFS"<< endl;
113          s.pop_back();
114          if (c[i] == *c) {
115              c[i] = c;
116              vector<int> edges;
117              g.outEdges(i, edges);
118              for (int k = 0; k < edges.size(); k++)
119                  s.push_back(edges[k]);
120          }
121      }
122  delete[] c;
123  }
```

3.1.3.5 hasEdge()

```
bool Graph::hasEdge (
              int i,
              int j )  [inline]
```

Definition at line 55 of file main.cpp.

```
55                                  {
56      vector<int>::iterator it;
57      it = find (adj.begin(), adj.end(), i);
58      if (it != adj.end())
59          return adj[i];
60  }
```

3.1.3.6 inEdges()

```
void Graph::inEdges (
            int i,
            vector< int > & edges )  [inline]
```

Definition at line 68 of file main.cpp.

```
68                                          {
69     for (int j = 0; j < n; j++) {
70         vector<int>::iterator it;
71         it = find (adj.begin(), adj.end(), i);
72         if (it != adj.end())
73                 edges.push_back(j);
74     }
75 }
```

3.1.3.7 nVertices()

```
int Graph::nVertices ( )  [inline]
```

Definition at line 77 of file main.cpp.

```
77                  {
78     return n * n;
79 }
```

3.1.3.8 outEdges()

```
void Graph::outEdges (
            int i,
            vector< int > & edges )  [inline]
```

Definition at line 63 of file main.cpp.

```
63                                          {
64     for (int k = 0; k < adj.size(); k++)
65         edges.push_back(adj.at(k));
66 }
```

3.1.3.9 printGraph()

```
void Graph::printGraph (
            vector< int > adj,
            int V )  [inline]
```

Definition at line 125 of file main.cpp.

```
126 {
127     int x;
128     for (int v = 0; v < V; ++v) {
129         cout « "vertex " « v « " ";
130         for (auto x : adj){
131             cout « "-> " « x;
132         printf("\n");
133         }
134         break;
135     }
136
137     printf("\n");
138 }
```

3.1.4 Member Data Documentation

3.1.4.1 adj

```
vector<int> Graph::adj
```

Definition at line 21 of file main.cpp.

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

- /home/addis/graph2/listgraph/src/main.cpp

Chapter 4

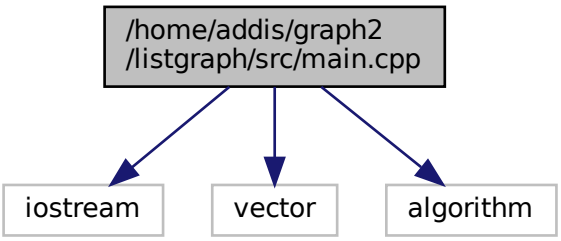
File Documentation

4.1 /home/addis/graph2/listgraph/src/main.cpp File Reference

This is a graph project.

```
#include <iostream>
#include <vector>
#include <algorithm>
```

Include dependency graph for main.cpp:



Classes

- class Graph

Functions

- int main ()

4.1.1 Detailed Description

This is a graph project.

This is the long brief at the top of main.cpp.

Author

Addis Bogale and Bona Tufa

Date

4/2/2021

4.1.2 Function Documentation

4.1.2.1 main()

```
int main ( )
```

Definition at line 141 of file main.cpp.

```
142 {
143     int v;
144     vector<int> value;
145 
146     Graph test = Graph(value, 10);
147 
148     // Adding edge as shown in the example figure
149 
150 
151     test.addEdge(value, 0, 4);
152     test.printGraph(value, v);
153     test.addEdge(value,1, 2);
154     test.printGraph(value, v);
155 
156     // Printing adjacency matrix
157     test.printGraph(value, v);
158 
159     // Deleting edge (1, 4)
160     // as shown in the example figure
161     test.delEdge(value, 1, 4);
162 
163     // Printing adjacency matrix
164     test.printGraph(value, v);
165 
166     return 0;
167 }
```

Index