

```

1
2 #include <iostream>
3 #include <list>
4 #include "graph.h"
5 /*****
6 Author: Jose Eduardo Morales
7 graph.cpp
8 Date: march 22, 2023
9 *****/
10
11 //empty construct
12 Graph::Graph(){};
13 //construct that creates an empty graph with size as a parameter
14 Graph::Graph (int size = 5):m_size(size) { //constructor
15
16     if (m_size == 0){
17         std::cout << "error initializing graph at 0" << std::endl;
18     }
19
20     graph = new int*[m_size];
21     for (int i = 0; i < m_size; i++)
22         graph[i] = new int[m_size];
23
24     //std::cout << "graph created" << std::endl;
25
26 }
27 Graph::~Graph() { // destructor
28     //std::cout << "destroying graph" << std::endl;
29 }
30
31 template <typename T>
32 void Graph::printGraph(T** graph, int rows, int cols) {
33     //print titles
34     std::cout << "\t";
35     for(int i = 0; i < cols; ++i)
36     {
37         std::cout << "n" << i << "\t" ;
38     }
39     //print content
40     std::cout << std::endl;
41     for(int i = 0; i < rows; ++i) {
42         std::cout << "n" << i << "\t";
43         for(int j = 0; j < cols; ++j) {
44             std::cout << graph[i][j] << " \t";
45         }
46         std::cout << std::endl;
47     }
48 }
49 // public print
50 void Graph::print() {
51     printGraph(graph, m_size, m_size);
52 }
53
54
55 int Graph::V() { //return the # of vertices in the Graph
56     return m_size;
57 }
58
59 int Graph::E() { //return the # of edges in the Graph
60     int count_E = 0;
61     for(int i = 0; i < m_size; ++i) {
62         for(int j = 0; j < m_size; ++j) {
63             if ( graph[i][j] > 0)
64                 count_E ++;
65         }
66     }
67     return count_E;
68 }
69
70 bool Graph::adjacent(int x, int y){ //test if there is an edge from x to y
71     return (graph[x][y] > 0);
72 }
73

```

```

74  std::list<int> Graph::neighbors(int x){ // list all nodes y with edges to x
75      std::list<int> neighborsToX = {};
76
77      for(int i = 0; i < m_size; ++i) {
78          if ( graph[x][i] > 0)
79              neighborsToX.push_back(i);
80      }
81      return neighborsToX;
82  }
83
84  void Graph::add(int x, int y, int value){ //add the edge from x to y
85      graph[x][y] = value;
86      graph[y][x] = value;    // make bidirectional edge
87  }
88
89  void Graph::remove(int x, int y){ //removes edge from x to y
90      graph[x][y] = 0;
91      graph[y][x] = 0;        // bidirectional edge
92  }
93
94  int Graph::getEdge(int x, int y){
95      return graph[x][y];
96  }
97
98  float Graph::getDensity(){
99      return static_cast<float>(E())*100.0/(V()*(V()-1));
100 }
101 /*
102 int Graph::get_node_value(int x){return x} //return node name
103 void Graph::set_node_value(int x, char a){} //change node name
104
105 int Graph::get_edge_value(int x, int y);
106 void Graph::set_edge_value(int x, int y, int value);
107
108     */

```