Instituto Tecnológico y de Estudios Superiores de Monterrey

Programación de Estructuras de Datos y Algoritmos Fundamentales

Ivan Reyez Amezcua

Act 4.1 - Grafo: sus representaciones y sus recorridos

Arturo Ramos Martínez //A01643269

13 de Noviembre del 2023

Documentación de las funciones:

loadGraph	Descripción	Cargue los arcos del grafo y los almacene en una Matriz de Adyacencia y en una Lista de Adyacencia
	Entrada	Dos enteros positivos: - n (Cantidad de Vertices) - m (Cantidad de Arcos) La Matriz de Adjacencia (parámetro por ref) La Lista de Adjacencia (parámetro por ref)
	Salida	nada
	Precondición	n y m son enteros positivos.
	Postcondición	Grafo representado en la lista y matriz de adyacencia
	Complejidad	La complejidad de la creación del grafo dependerá del número de arcos que se hayan seleccionado entonces sería O(e), e = número de edges.

DFS (para la matriz de adyacencia)	Descripción	Imprima el recorrido de DFS (Profundidad) a partir del nodo inicial.
	Entrada	Recibe un valor entero (init_node), que

		representa el nodo por el cual se iniciará el recorrido
	Salida	nada
	Precondición	Que “init_node” sea un entero positivo
	Postcondición	nada
	Complejidad	Como se está usando una matriz de adyacencia, la complejidad incrementa, ya que en el peor de los casos se podría visitar literalmente toda la matriz, lo que nos daría una complejidad de: O(V^2)

BFS (para la lista de adyacencia)	Descripción	Imprima el Recorrido de BFS (Anchura) a partir de nodo inicial.o
	Entrada	Recibe un valor entero (init_node), que representa el nodo por el cual se iniciará el recorrido
	Salida	nada
	Precondición	Que “init_node” sea entero positivo
	Postcondición	nada
	Complejidad	O(V + E) v= vértices, e = edges o arco. Esto ya que se necesitará visitar por lo menos una vez cada vértice o nodo, y además,

		se le agregan la cantidad de enlaces o arcos, ya que no siempre se tendrá la misma configuración de conexiones o arcos.

Implementación en C++:

```
#include <iostream>
#include <list>
#include <stack>
#include <queue>
using namespace std;

class Graph {
public:
    int vertices;
    int edges;
    list<int>* adj;
    bool** adj_matrix;
    Graph(int v, int e);
    void loadGraph(int v, int e, list<int>*& adj, bool**&);
    void DFS(int);
    void BFS(int);
    void displayMatrix();
    void displayList();
};

Graph::Graph(int v, int e) {
    this->vertices = v;
    this->edges = e;
    this->adj = new list<int>[vertices];
    this->adj_matrix = new bool* [vertices];
    for (int i = 0; i < vertices; ++i) {
        this->adj_matrix[i] = new bool[vertices];
        for (int j = 0; j < vertices; ++j) {
            this->adj_matrix[i][j] = false;
        }
    }
    loadGraph(vertices, edges, adj, adj_matrix);
}
```

```
void Graph::loadGraph(int vertices, int edges, list<int>*& adj, bool**&
adj_matrix){
    int c = 0;
    int i = 0;
    int seguir;
    int arco;
    cout << "Elige las conexiones entre los: "<<vertices<<" vertices,
recuerda que tienes: "<<edges<<" arcos"<< endl;
    while(c < edges){
        seguir = 1;
        while(seguir == 1){
            cout << "Número de arcos: " << c << endl;
            cout << "vertice: "<<i<<endl;
            cout << "Tiene arco con (-1 si no quieres que tenga arco
con nigun nodo): "<<endl;
            cin>>arco;
            if(arco == -1){
                break;
            }
            adj[i].push_back(arco);
            //adj[arco].push_back(i);
            adj_matrix[i][arco] = true;
            c++;
            if(c == edges){
                return;
            }
            cout << "deseas seguir poniendo arcos al vertice: "<< i<<"
si = 1, no = 0"<<endl;
            cin>>seguir;
        }
        i++;
    }
}

void Graph::DFS(int init_node){
    cout << "Recorrido DFS con el gráfo reprensetado en una matriz de
adyacencia" << endl;
    bool* visited = new bool[vertices]; // nodos ya explorados
    // primeramente inicializar con todas las posiciones del array(que
a su vez representan cada nodo en el gráfo) en false.
    for(int i = 0; i< vertices; i++){
        visited[i] = false;
    }
```

```
    stack<int> stack; // nodos por visitar
    stack.push(init_node);
    cout << "[";
    while(!stack.empty()){
        init_node = stack.top(); // actualizar el nodo actual
        if(!visited[init_node]){
            cout << init_node <<" ";
            visited[init_node] = true;
        }
        for(int j = 0; j < vertices; j++){
            if(adj_matrix[init_node][j] && !visited[j]){
                stack.push(j);
            }
        }
        if(stack.top() == init_node){
            stack.pop();
        }
    }
    cout << "]" << endl;
}

void Graph::BFS(int init_node){
    cout << "Recorrido BFS con el gráfo reprensetado en una lista de
adyacencia" << endl;
    bool* visited = new bool[vertices]; // nodos ya explorados
    // primeramente inicializar con todas las posiciones del array(que
a su vez representan cada nodo en el gráfo) en false.
    for(int i = 0; i< vertices; i++){
        visited[i] = false;
    }
    queue<int> queue;
    queue.push(init_node);
    cout << "[";
    while(!queue.empty()){
        init_node = queue.front();

        if(!visited[init_node]){
            cout << init_node <<" ";
            visited[init_node] = true;
        }
```

```
        for(list<int>::iterator i = adj[init_node].begin(); i!=
adj[init_node].end() ; i++){
            if(!visited[*i]){
                queue.push(*i);
            }
        }
        queue.pop();
    }
    cout << "]"<< endl;
}

void Graph::displayMatrix(){
    cout << "--------------------------------" << endl;
    cout << "Matriz de adyacencia" << endl;
    for(int i = 0; i < vertices; i++){
        for(int j = 0; j < vertices ;j++){
            cout << adj_matrix[i][j] << " ";
        }
        cout << endl;
    }
    cout << "--------------------------------" << endl;
}
// iterator va accediendo a las direcciones de memoria de cada elemento
en la lista "adj", por eso
// para acceder al elemento como tal, se necesita indicar que queremos
lo que esta en esa dirección de
// memoria con "*"
void Graph::displayList(){
    cout << "--------------------------------" << endl;
    cout << "Lista de adyacencia" << endl;
    for(int i = 0; i < vertices; i++){
        cout << "vertice: " <<i<<" [";
        for(list<int>::iterator j = adj[i].begin(); j!= adj[i].end() ;
j++){
            cout << *j <<" ";
        }
        cout <<"]"<< endl;
    }
    cout << "--------------------------------" << endl;
}

int main (){
    int v, e, first_node;
```

```
    cout << "Indica la cantidad de nodos o vertices y de arcos" <<
endl;
    cout << "Vertices: "<<endl;
    cin>> v;
    cout << "Arcos: "<<endl;
    cin>>e;
    cout << "Por último indica el vertice por el cual se iniciarán los
recorridos" << endl;
    cout << "Recuerda que debe estar dentro de tu rango de vertices:
"<<v << endl;
    cin>>first_node;
    Graph g(v, e);
    g.displayMatrix();
    g.displayList();
    g.DFS(first_node);
    g.BFS(first_node);
    return 0;
}
```