Reporte de Laboratorio 6

Isaac Gómez Sánchez- B32919
José Fernando González Salas - B43023

13 de noviembre de 2016

Índice

---

1. Introducción

En el siguiente reporte se presenta una implementación de la estructura abstracta de grafo usando una matriz de adyacencia y búsquedas en profundidad y ancho, con el fin de comprender dicha estructura y percibir diferentes circunstancias en las que se pueda usar.

1.1. Objetivos

- Comprender la estructura abstracta grafo.
- Implementar la estructura de datos abstracta grafo usando una matriz de adyacencia y las búsquedas en profundidad y ancho.

2. Código

El código empleado para implementar y diseñar la estructura de grafo se presenta a continuación.

```
//main.cpp
#include "Graph.h"

int main()
{

    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
```

```
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    g.DFS();
    cout << endl;
    g.BFS(2);
    cout << endl;

    return 0;
}
```

```
//Graph.h

#ifndef GRAPH_H
#define GRAPH_H

#include<iostream>
#include<list>

using namespace std;

class Graph
{
    int V;
    list<int> *adj;
    void DFSUtil(int v, bool visited[]);
public:
    Graph(int V);
    void addEdge(int v, int w);
    void DFS();
    void BFS(int s);
};

#endif /* GRAPH_H */
```

```
//Graph.cpp
#include "Graph.h"

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}


void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}


void Graph::DFSUtil(int v, bool visited[])
```

```
{
    visited[v] = true;
    cout << v << " ";

    list<int>::iterator i;
    for(i = adj[v].begin(); i != adj[v].end(); ++i)
        if(!visited[*i])
            DFSUtil(*i, visited);
}


void Graph::DFS()
{
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            DFSUtil(i, visited);
}

void Graph::BFS(int s)
{
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;

    list<int> queue;

    visited[s] = true;
    queue.push_back(s);

    list<int>::iterator i;

    while(!queue.empty())
    {
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        for(i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if(!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}
```

```
#Makefile
```

```
compile:
   g++ -Wall main.cpp Graph.cpp -o a.out

clean:
   rm a.out

debugg:
   g++ -g main.cpp Graph.cpp -o debuger

execute: compile
   ./a.out
```

---

3. Conclusiones

- Se comprendió la estructura abstrata de grafo.
- Se implementa la estructura de datos abstracta grafo usando una matriz de adyacencia y las búsquedas en profundidad y ancho.