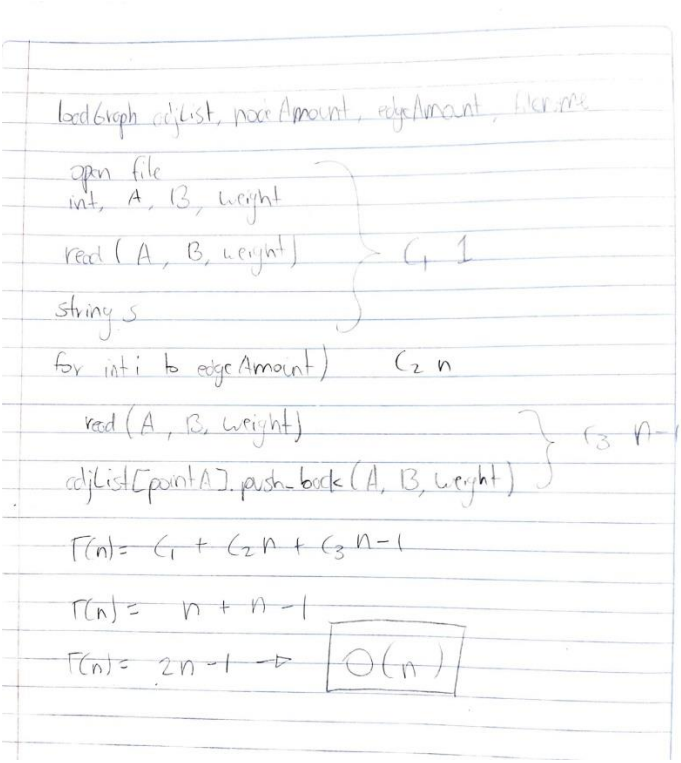
Jorge Del Barco Garza
A01284234

17 Noviembre 2021

Actividad 4.1 – Grafo: sus representaciones y recorridos

loadGraph

```
void loadGraph(vector<vector<Node<int>>> &adjList, int nodeAmount, int edgeAmount, string filename){

    fstream in(filename);

    int pointA, pointB, edgeWeight;

    // we read input to ignore first line.
    in >> pointA >> pointB >> edgeWeight;

    string s;
    // here we start from one bc we want to ignore the 0 index that is empty, but this will not
    // always be the case.
    for (int i = 0; i < edgeAmount; i++){

        in >> pointA >> pointB >> edgeWeight;

        adjList[pointA].push_back(Node<int>(pointA, pointB, edgeWeight));

        // PREGUNTALE AL PROFE SI ESTO ESTÁ BIEN.
        /* EN TEORÍA ES UN GRAFO NO DIRIGIDO, POR LO QUE LOS NODOS PUEDEN INTERCONECTARSE POR ESO
        LA LÍNEA  DEBAJO ES VÁLIDA*/
        //adjList[pointB].push_back(Node<int>(pointB, pointA, edgeWeight));
    }
}
```

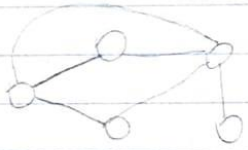


BFS

```
void BFS(vector<vector<Node<int>>> &adjList, int rootNode){

    queue<int> gray; // processed nodes
    vector<bool> visited;
    int currentNode;

    // set all nodes to not visited
    for (unsigned int i = 0; i < adjList.size(); i++){
        visited.push_back(false);
    }

    // aqui se hace un push del nodo en el que estamos
    gray.push(adjList[rootNode][0].getOrigin());

    // el nodo inicial lo marcas como visitado
    visited[rootNode] = true;

    // OJO: EL LOOP PODRÍA SER SIZE() - 1 PORQUE NO ESTAMOS CONTANDO EL CERO PERO SÍ ESTÁ AHÍ. CUIDADO

    while(!gray.empty()){

        currentNode = gray.front();
        gray.pop();

        // OUTPUT
        cout << currentNode;
```

```
        for (unsigned int adjNode = 0; adjNode < adjList[currentNode].size(); adjNode++){

            //cout << adjList[currentNode][adjNode].getDestination() << " ";

            // si ya está visitado ese nodo, pasa al siguiente.
            if (visited[adjList[currentNode][adjNode].getDestination()] == true){
                continue;

            } else {
                // si no lo haz visitado. Marcalo como visitado
                visited[adjList[currentNode][adjNode].getDestination()] = true;
            }

            // add nodes attached to currentNode
            gray.push(adjList[currentNode][adjNode].getDestination());

        }
            cout << " ";
    }
    cout << endl;
}
```

```
Void BFS (adjList, rootNode)
    for 0 to adjList.size                                   c1 n
        visited.push_back(false)
    gray.push(adjList[rootNode][0].getOrig.n);              c2 1
    visited[rootNode] = true                                c3 1
c4 {Tj} while (gray not empty)                              * atravesamos cada nodo y vértice exactamente una vez.
    c5 {tj-1} {
        currentNode = gray.front()
        gray.pop()
        print(currentNode)
    }
    c6 {tj-1} {
        for adjNode = 0 to adjList[currentNode.size()]
            if adjnode in visited
                continue
    }
    visited[adjNode] = true                                 c7 1
    gray.push(adjList[currentNode][adjNode])                c8 1
```



O(n)

$T(n) = c_1 n + c_2 + c_3 + c_4 \{t_j + c_5 \{t_j - 1 + c_6 \{t_j - 1 + c_7$

$T(n) = O(n) \rightarrow \boxed{O(V+E)}$

DFS

```
void DFS(vector<vector<Node<int>>> &adjList, int rootNode){

    timeCounter = 0;

    if (dfsVisited[adjList[rootNode][0].getOrigin()] == true){
      return;
    }

    // imprime el primer nodo que visitaste
    cout << adjList[rootNode][0].getOrigin() << " ";

    for (unsigned int currentNode = rootNode; currentNode < adjList.size()-1; currentNode++){
      //cout << currentNode << endl;

          // si no hemos visitado el nodo. Lo visitamos con DFS-Visit
          // adjList[currentNode][0].getOrigin() -> 1
        if (dfsVisited[adjList[currentNode][0].getOrigin()] == false){

          // ponemos el nodo fuente como ya visitado
          dfsVisited[adjList[rootNode][0].getOrigin()] = true;

          // pasamos la lista entera y la lista de adjacencia del currentNode
          DFSVisit(adjList, adjList[currentNode]);
        }

    }
      // ponemos el nodo fuente como ya visitado
        dfsVisited[adjList[rootNode][0].getOrigin()] = true;
```

```
    // edge case para casos donde no hay un nodo conectado
    for (unsigned int i = 0; i < dfsVisited.size(); i++){



        // si no haz visitado ese nodo. Hazle dfs
        if (dfsVisited[i] == false && adjList[i].size() != 0){
            // hazle DFS a ese nodo
            DFS(adjList, i);
        }
    }

}
```

Void DFS (adjList, rootNode)

```
timeCounter = 0                                          C1 1
if visited[adjList[rootNode][0].getOrigen() == true
    return
print adjList[rootNode][0].getOrigen()
for int currentNode = rootNode to adjList.size()-1       C2 n
    if visited[adjList[currentNode[0].getOrigen()] == false
        visited[adjList[rootNode][0].getOrigen()] = true
        DFSVisit (adjList, adjList[currentNode])
    visited[adjList[rootNode][0].getOrigen()] = true

for int i = 0 to visited.size                            C3
    if visited[i] == false and adjList[i].size() != 0
        DFS(adjList, i)      * toco cada nodo
                               y entro una vez.
```

$T(n) = n - 1$

$O(n) \rightarrow \boxed{O(V+E)}$