Universidad de Concepción
Facultad de Ingeniería
Departamento de Ing. Informática y Cs
de la Computación

Profesor
Diego Seco

Laboratorio 7:

Estructuras de Datos (503220)

Estudiante: Nicolás Araya (2018448613)

1)

```
int main(){
    int N, M;
    cin>>N>>M;
    if(N<=1) return 0;
    vector<vector<int>> grafo;
    vector<int> iniciador;
    grafo.assign(N+1, iniciador);

    while(M--){
        int A, B;
        cin>>A>>B;
        grafo.at(A).push_back(B);
        grafo.at(B).push_back(A);
    }

    //printGrafo(grafo);
    DFS(grafo);
    BFS(grafo);

    return 0;
}
```

2)

DFS (recursivo):

```
void DFS(vector<vector<int>> grafo){
    cout<<"Recorrido DFS"<<endl;
    vector<bool> discovered;
    bool f = false;
    discovered.assign(grafo.size(), f);
    vector<bool> *pointer = &discovered;
    for(int i = 0; i<grafo.size(); i++){
        if(discovered.at(i)==false && !grafo[i].empty()){
            DFS2(i, grafo, pointer);
        }
    }
    cout<<endl<<endl;
}
```

```
void DFS2(int n, vector<vector<int>> grafo, vector<bool>* discovered){
    discovered->at(n) = true;
    cout<<n<<endl;
    vector<int> adyacentes = grafo[n];
    for(int i = 0; i<adyacentes.size(); i++){
        if(discovered->at(adyacentes[i])==false && !grafo[adyacentes[i]].empty()){
            DFS2(adyacentes[i], grafo, discovered);
        }
    }
}
```

BFS:

```
void BFS(vector<vector<int>> grafo){
    cout<<"Recorrido BFS"<<endl;
    vector<bool> discovered;
    bool f = false;
    discovered.assign(grafo.size(), f);
    vector<bool> *pointer = &discovered;
    for(int i = 0; i<grafo.size(); i++){
        if(discovered.at(i)==false && !grafo[i].empty()){
            BFS2(i, grafo, pointer);
        }
    }
    cout<<endl<<endl;
}
```

```
void BFS2(int n, vector<vector<int>> grafo, vector<bool>* discovered){
    vector<int> L;
    L.push_back(n);
    cout<<n<<endl;
    discovered->at(n) = true;
    while(!L.empty()){
        vector<int> L2;
        for(int i = 0; i<L.size(); i++){
            vector<int> ady = grafo[L[i]];
            for(int j = 0; j<ady.size(); j++){
                if(discovered->at(ady[j])==false){
                    discovered->at(ady[j])=true;
                    cout<<ady[j]<<endl;
                    L2.push_back(ady[j]);
                }
            }
        }
        L = L2;
    }
}
```

3) Teóricamente, tanto BFS como DFS tienen complejidad O(n+m), con n correspondiente a números de nodos, y m al numero de aristas. Su complejidad se debe a la implementación del grafo con una lista de adyacencia. La cual permite que cada vértice contenga un arreglo únicamente los nodos a los que es adyacente. Esta implementación hace que los algoritmos DFS y BFS sean más óptimos en comparación a si el grafo fuese implementado con Matriz de Adyacencia o Lista de Aristas.