TALLER PROGCOMP: TRACK GRAFOS

ALGORITMOS DE RECORRIDO - DFS Y BFS

Gabriel Carmona Tabja

Universidad Técnica Federico Santa María,
Università di Pisa

September 1, 2024

GRAFOS

No solo nos sirve representar utilizando grafos, también nos gustaría poder recorrerlo para poder responder consultas.
Para ello, utilizaremos dos algoritmos de recorrido:

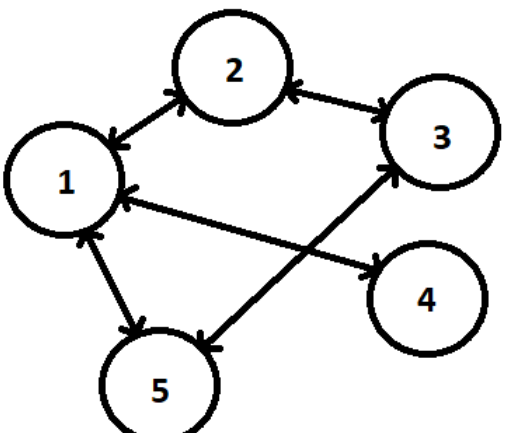
- DFS, Depth-First Search (Búsqueda en profundidad)
- BFS, Breath-First Search (Búsqueda en anchura)

Para ambos códigos utilizaremos la lista de adyacencia como representación.

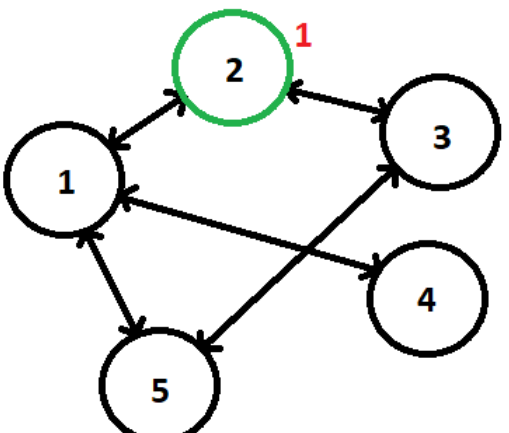
Part I

DFS

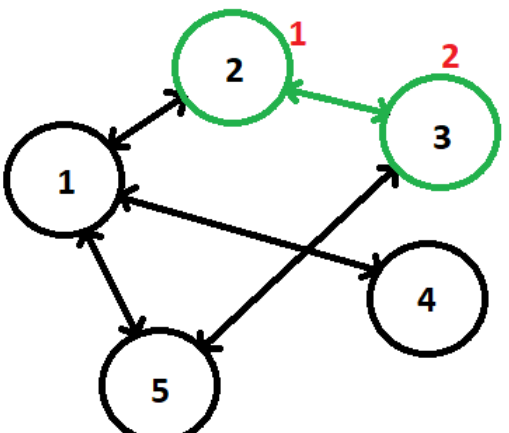
DFS



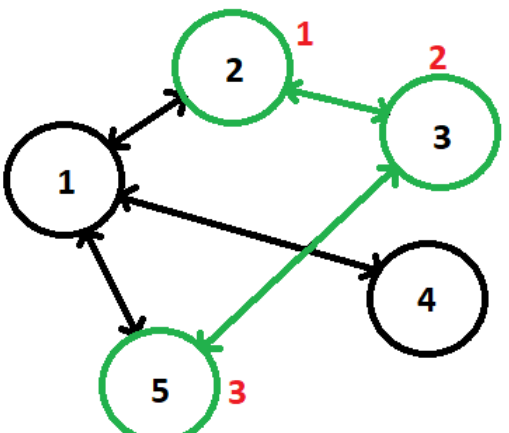
DFS



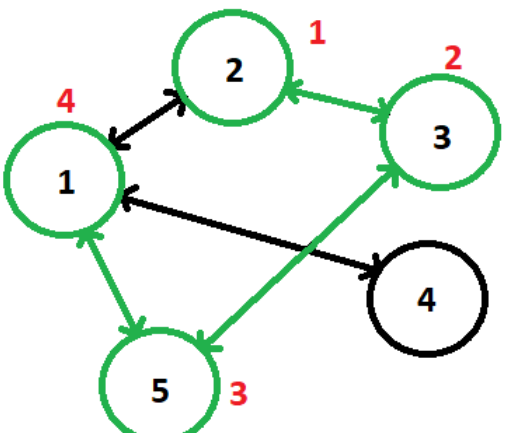
DFS



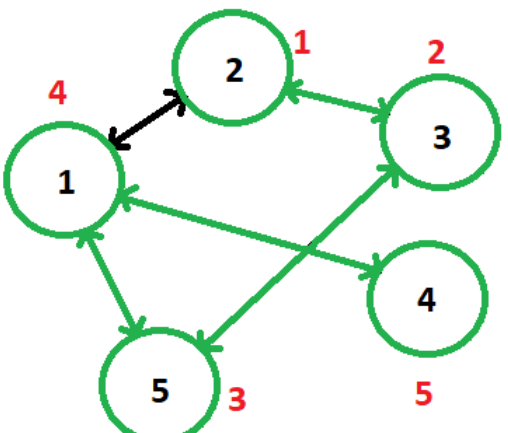
DFS



DFS



DFS



DFS Código

La complejidad de este algoritmo es $O(n + a)$, o sea vertices más aristas.

```
// n cantidad de nodos
// a cantidad de aristas
int n, v;
// true es para visitado, false es para no visitado
vector< bool > visitado;
vector< vector< int > > lista;

void dfs(int u) {
  visitado[u] = true;
  for(int i = 0; i < lista[u].size(); i++){
    int v = lista[u][i];
    if(visitado[v] == false){
      dfs(v);
    }
  }
}
```

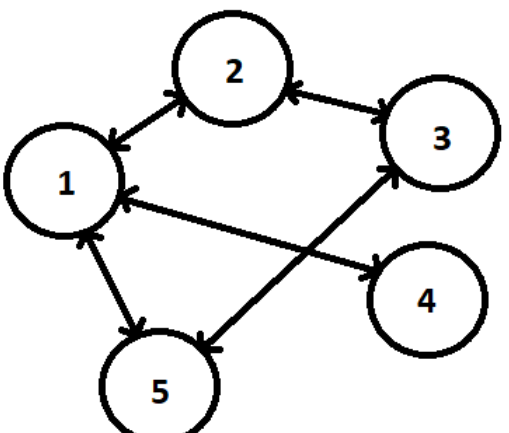
UTILIDAD - SI SE PUEDE LLEGAR A UN NODO

```cpp
int main(){
  cin >> n >> a;
  lista.resize(n);
  visitado.resize(n, false);

  for(int i = 0; i < a; i++){
    int x, y;
    cin >> x >> y;
    // se anade una arista
    lista[x].push_back(y);
    lista[y].push_back(x);
  }

  // por ejemplo quiero saber si es que puedo llegar al nodo n - 1 desde el nodo 0

  dfs(0);
  if(visitado[n - 1] == true){
    cout << "Si se puede llegar\n";
  } else {
    cout << "No se puede llegar\n"
  }
  return 0;
}
```

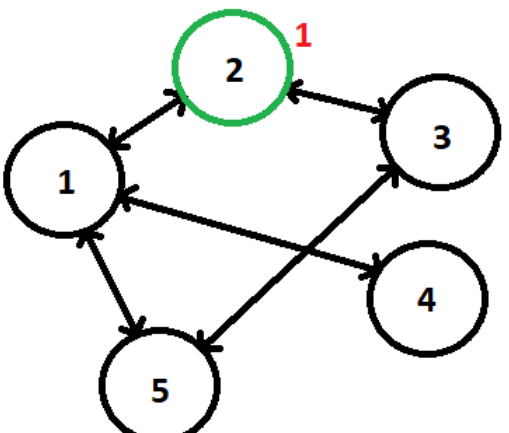
Part II

BFS

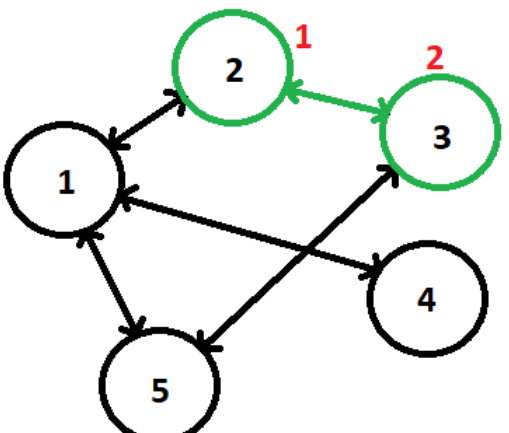
BFS



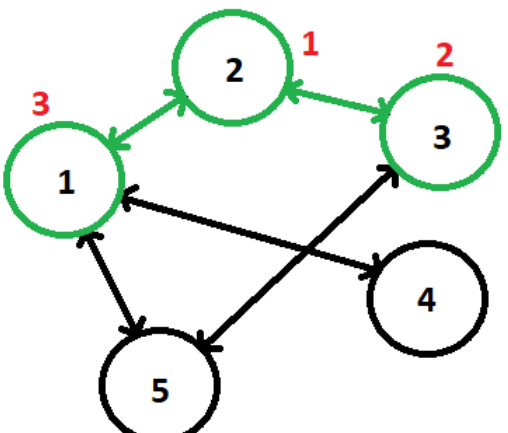
BFS



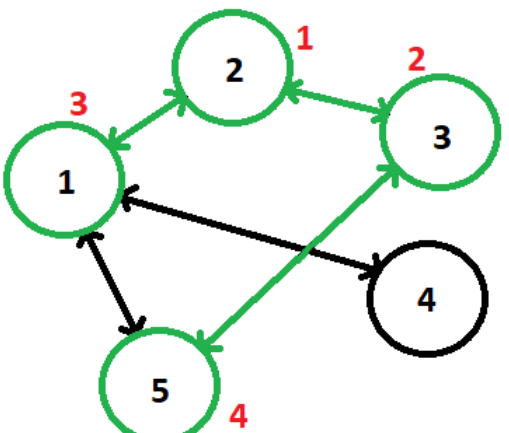
BFS



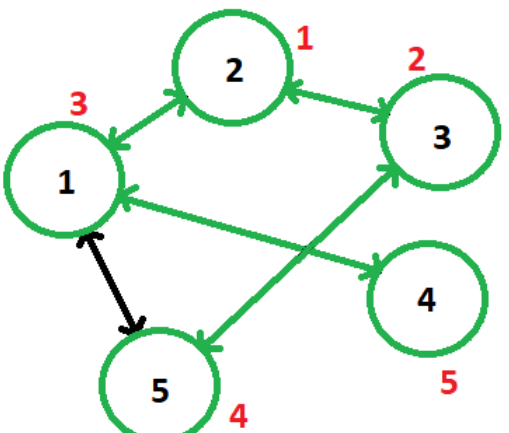
BFS



BFS



BFS



BFS Código

La complejidad de este algoritmo es $O(V + E)$

```
#define INF 1000000000
int n, a;
// que almacenara distancias hasta el nodo ‘i‘
vector< int > d;
vector< vector< int > > lista;
void bfs(int s) {
  d[s] = 0; // nodo inicial
  queue< int > q;
  //se anade nodo recien visitado
  q.push(s);
  while(!q.empty()){
    int u = q.front();
    q.pop();
    for(int i = 0; i < lista[u].size(); i++) {
      int v = lista[u][i];
      // si es infinito es porque no lo hemos visitado
      if(d[v] == INF){
        d[v] = d[u] + 1;
        q.push(v);
      }
    }
  }
}
```

Utilidad - Camino Mínimo

Puedes determinar con esto la distancia mínima desde un nodo hasta cualquier nodo!!

```
int main(){
  cin >> n >> a;
  lista.resize(n, vector< int >());
  d.resize(n, INF);

  for(int i = 0; i < a; i++){
    int x, y;
    cin >> x >> y;
    // se anade una arista
    lista[x].push_back(y);
    lista[y].push_back(x);
  }

  // por ejemplo, cual es la distancia minima para llegar de 0 hasta n -1 (si es que se puede)

  bfs(0);
  if(d[n - 1] == INF){
    cout << "No se puede llegar\n";
  } else {
    cout << "La distancia minima es: "  << d[n - 1] << "\n";
  }
  return 0;
}
```

REFERENCES I