BRUTE

Workshop

BRUTE UDESC

Problemas de Grafos

Enzo de Almeida Rodrigues

Eric Grochowicz

Eduardo Schwarz Moreira

João Marcos de Oliveira

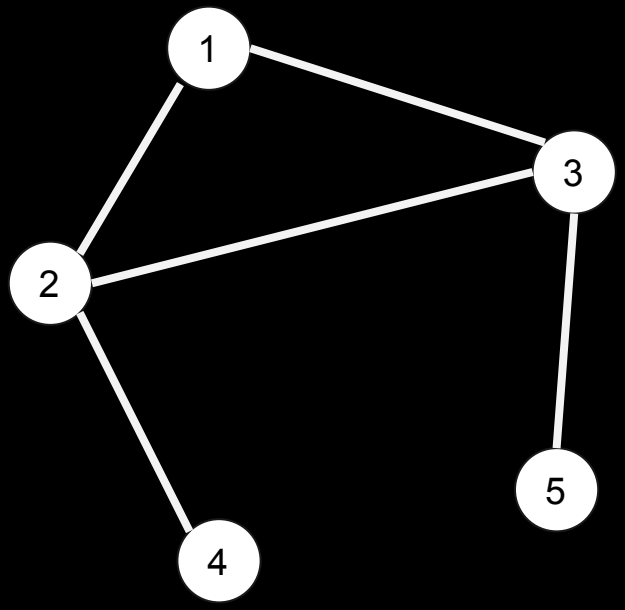
BRUTE

competitive programming

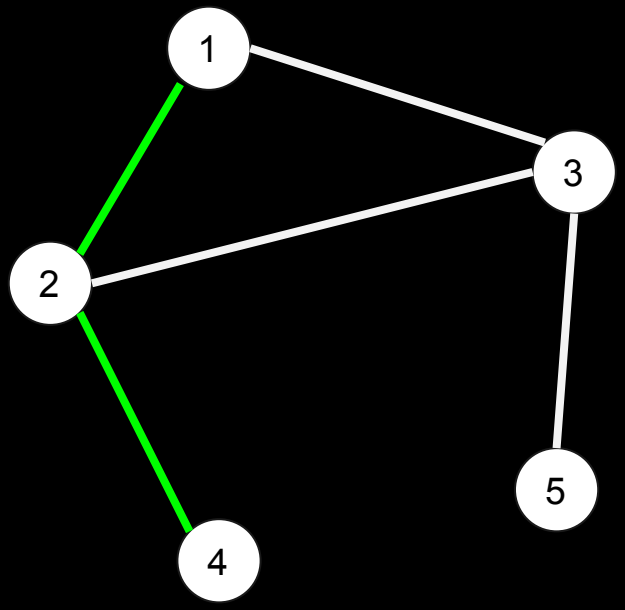
Conteúdos

- Conceito de Grafo
- Árvores
- DFS
- BFS
- DSU

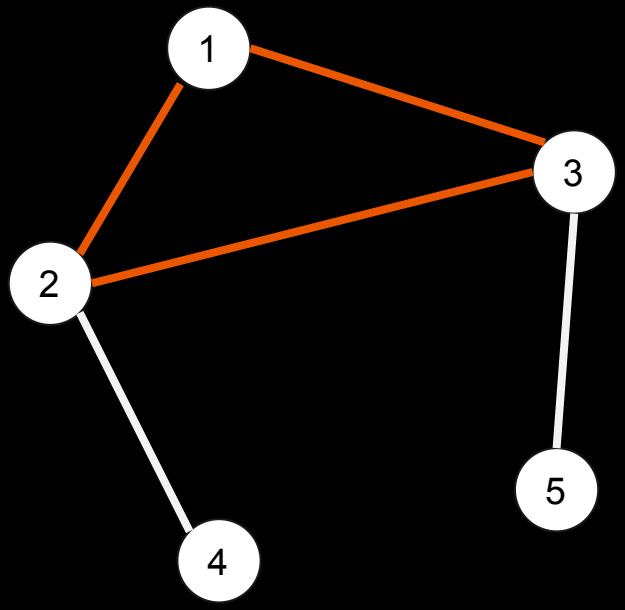
Um grafo



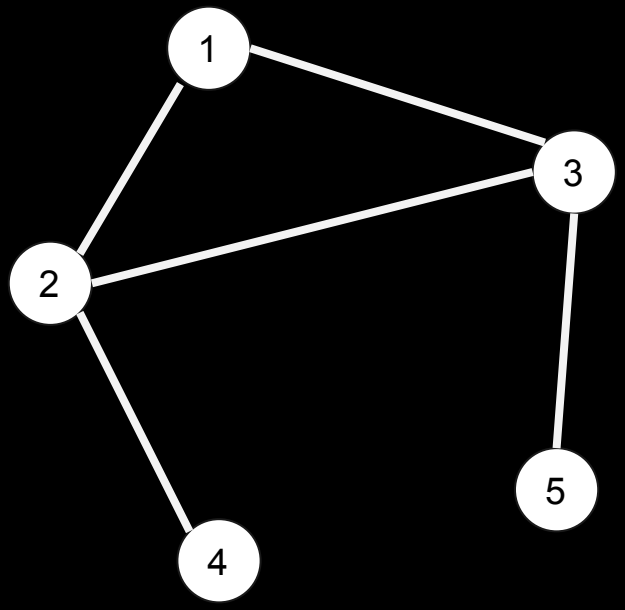
Um caminho



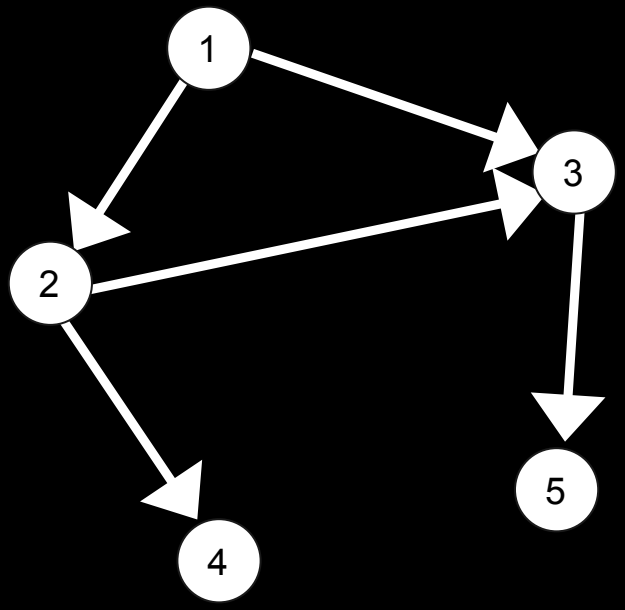
Um ciclo



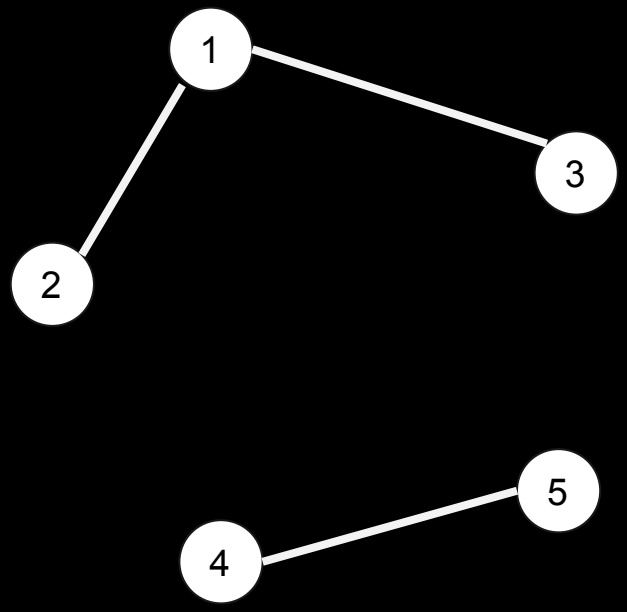
Um grafo não-direcionado (bidirecionado)



Um grafo direcionado (não vai ser abordado)



Duas componentes conexas

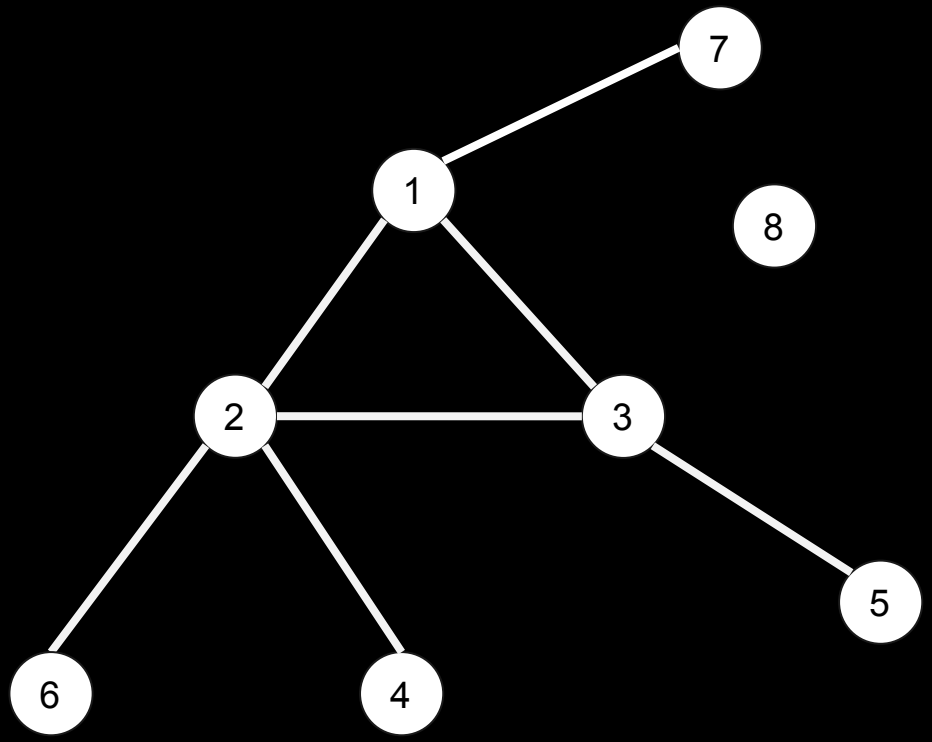


Probleminha

Dado um grafo, dizer se existe um caminho do vértice 1 até X.

DFS (Depth-First Search)

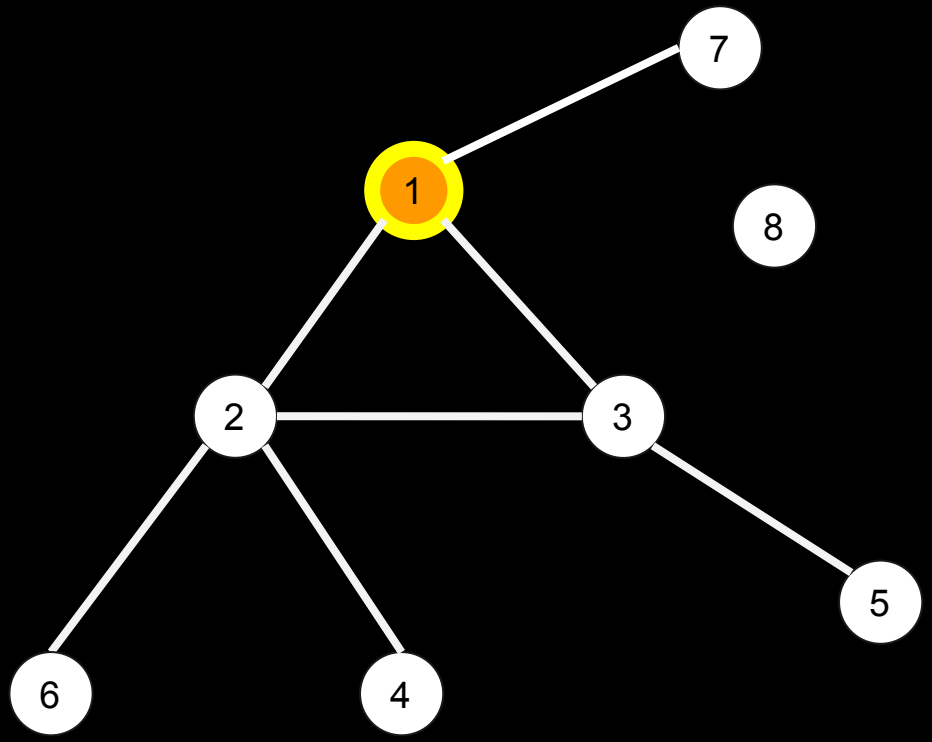
Stack:



DFS (Depth-First Search)

Stack:

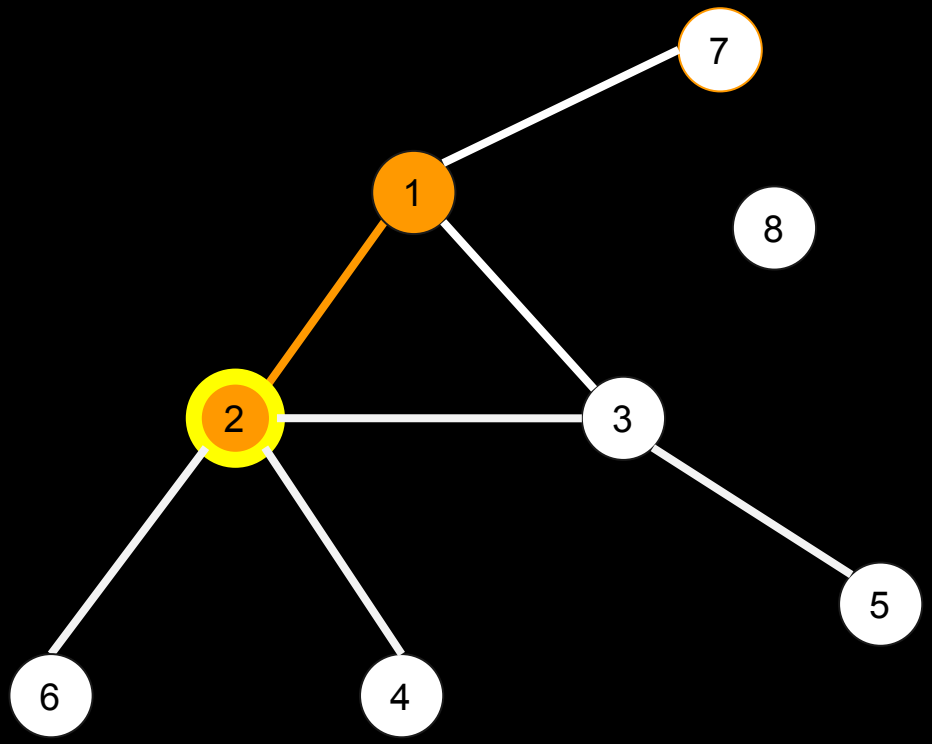
1



DFS (Depth-First Search)

Stack:

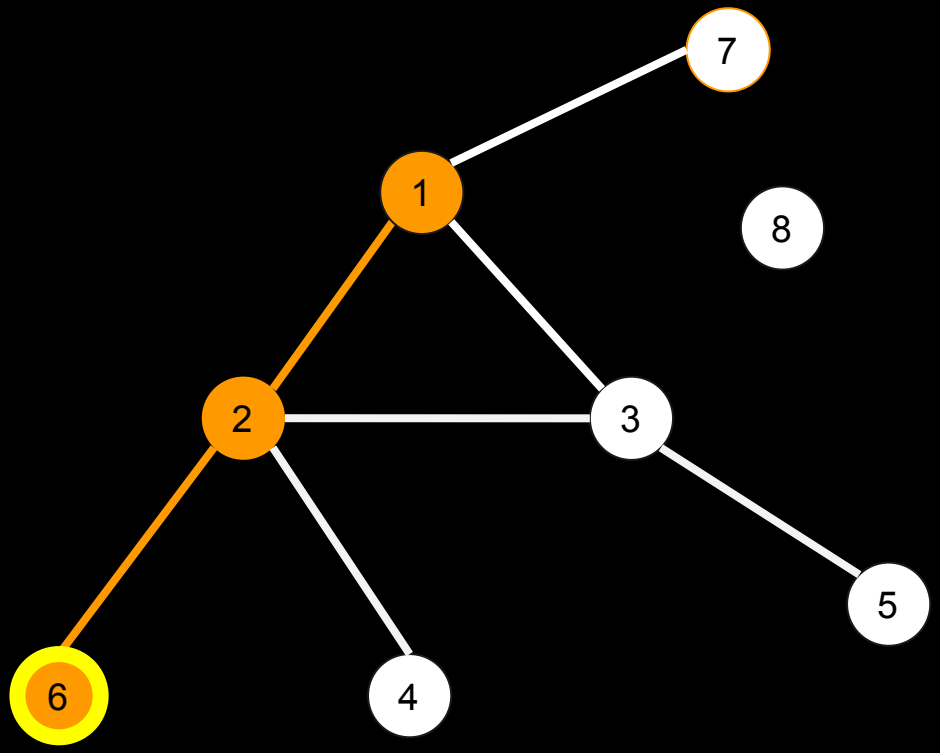
1 2



DFS (Depth-First Search)

Stack:

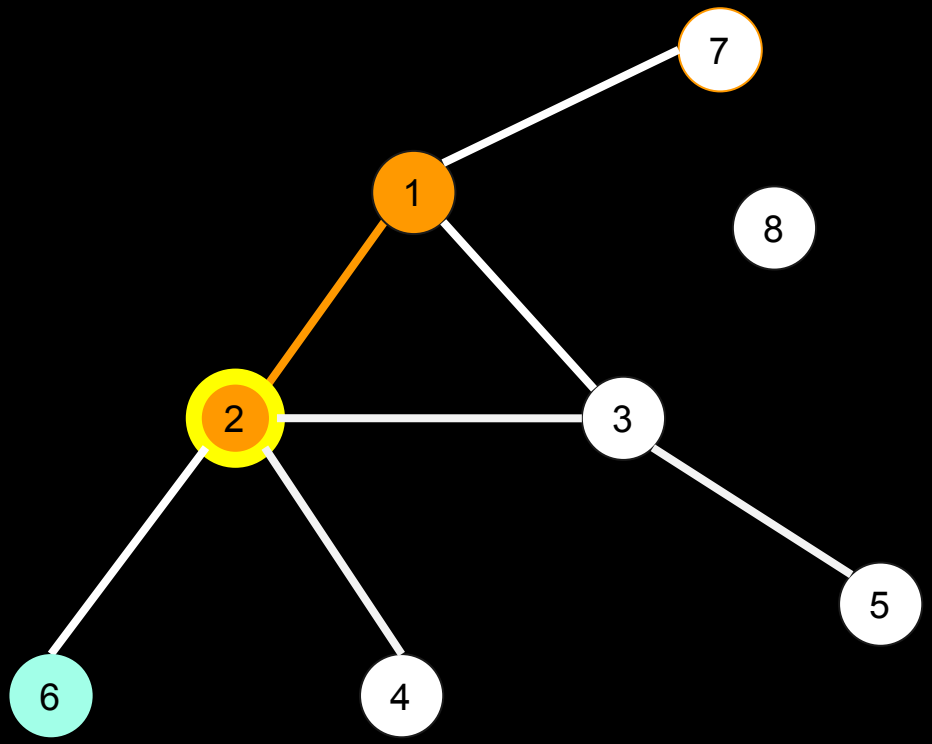
1 2 6



DFS (Depth-First Search)

Stack:

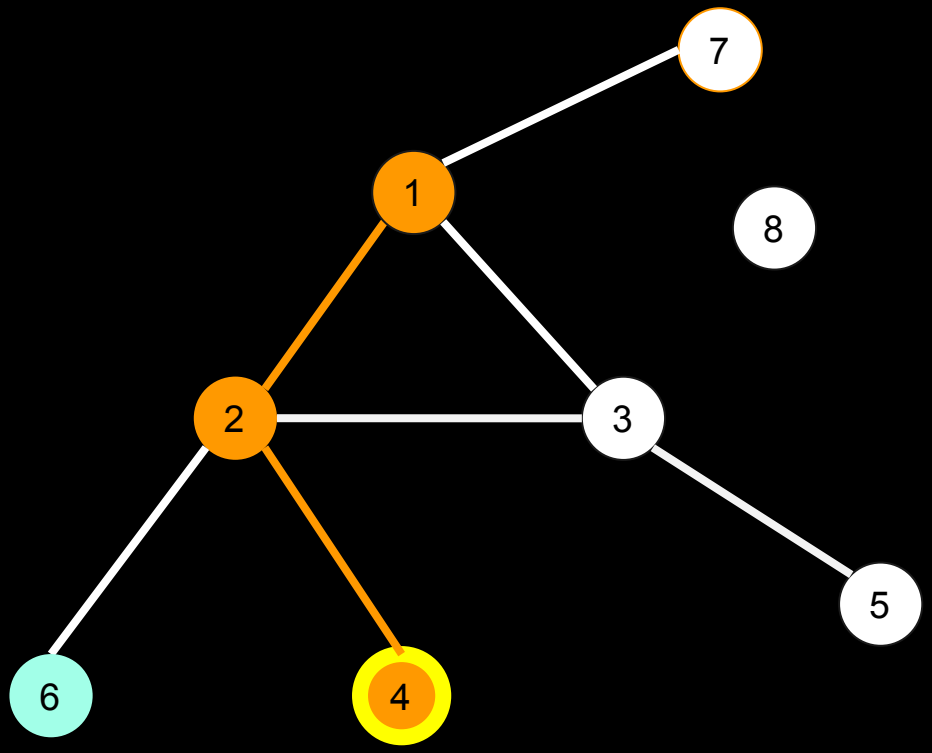
1 2



DFS (Depth-First Search)

Stack:

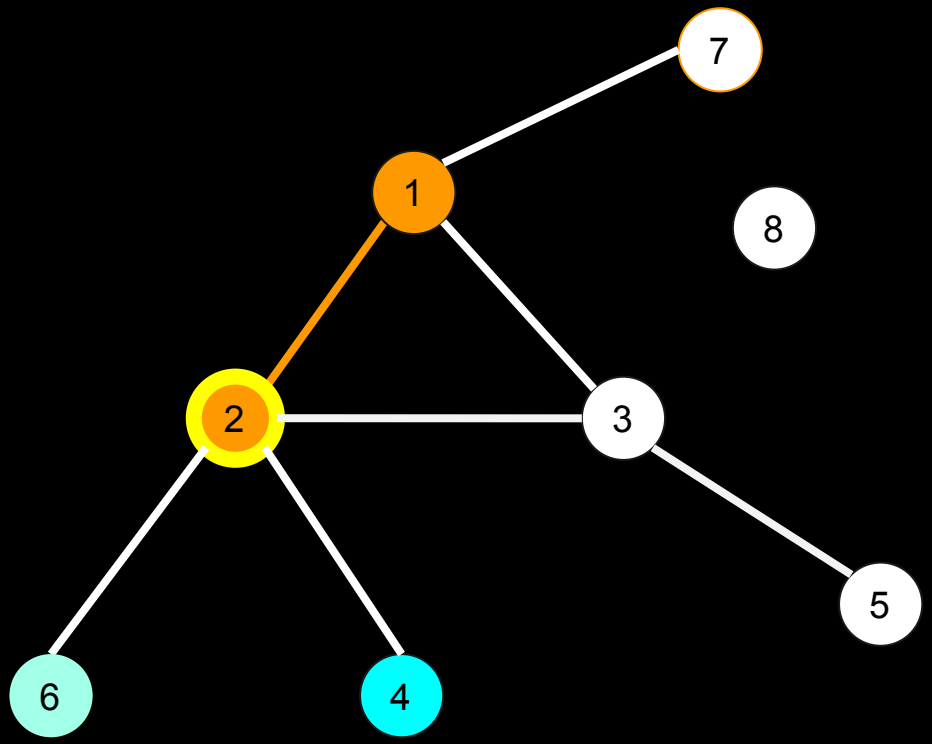
1 2 4



DFS (Depth-First Search)

Stack:

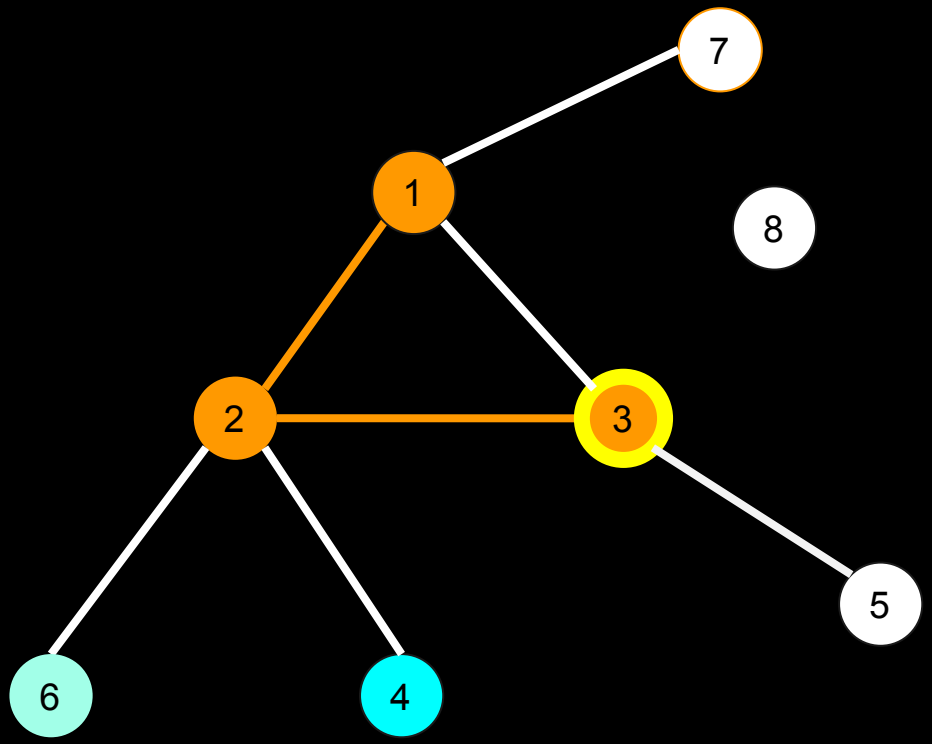
1 2



DFS (Depth-First Search)

Stack:

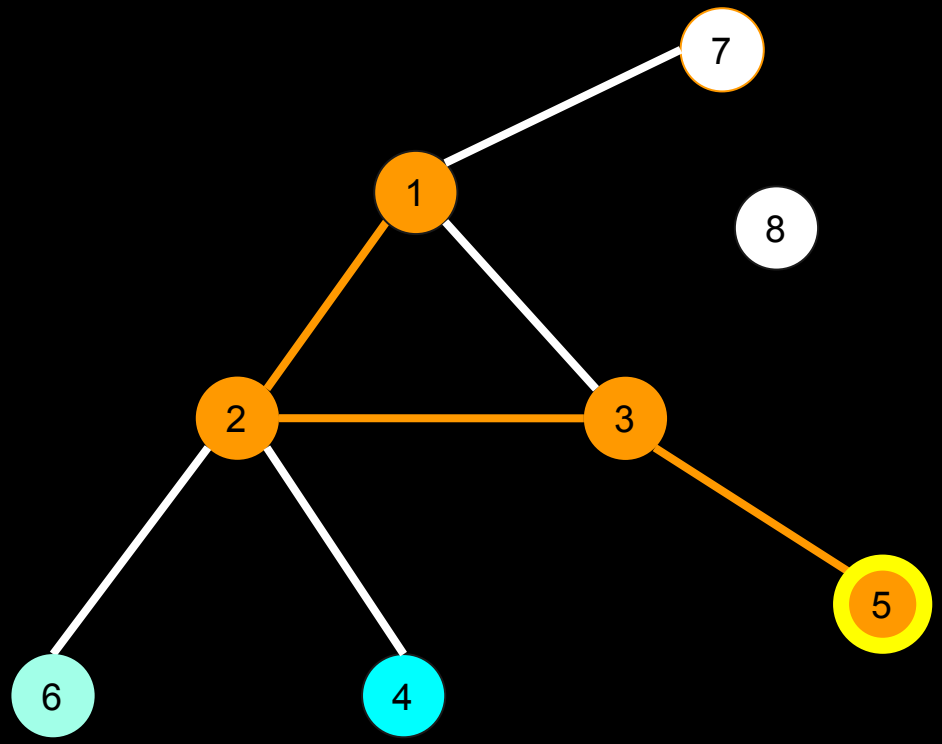
1 2 3



DFS (Depth-First Search)

Stack:

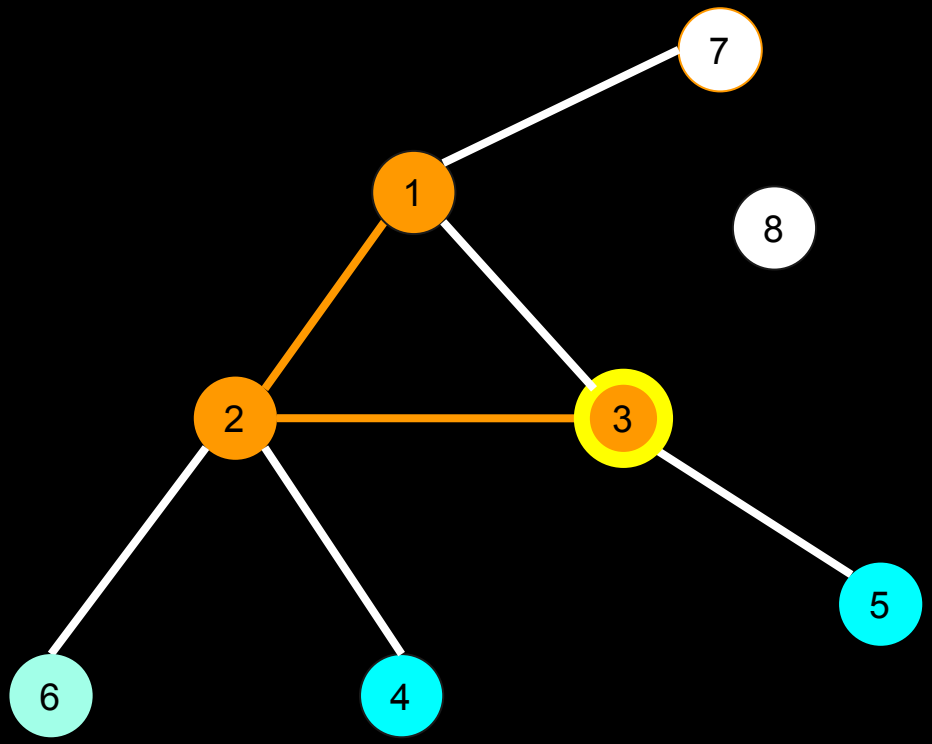
1 2 3 5



DFS (Depth-First Search)

Stack:

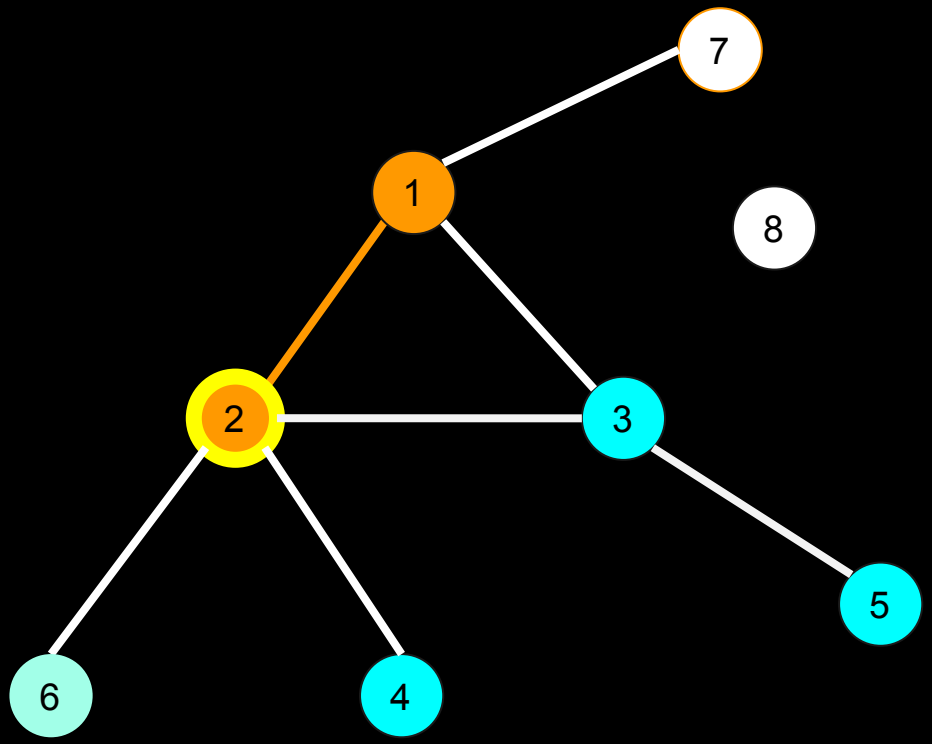
1 2 3



DFS (Depth-First Search)

Stack:

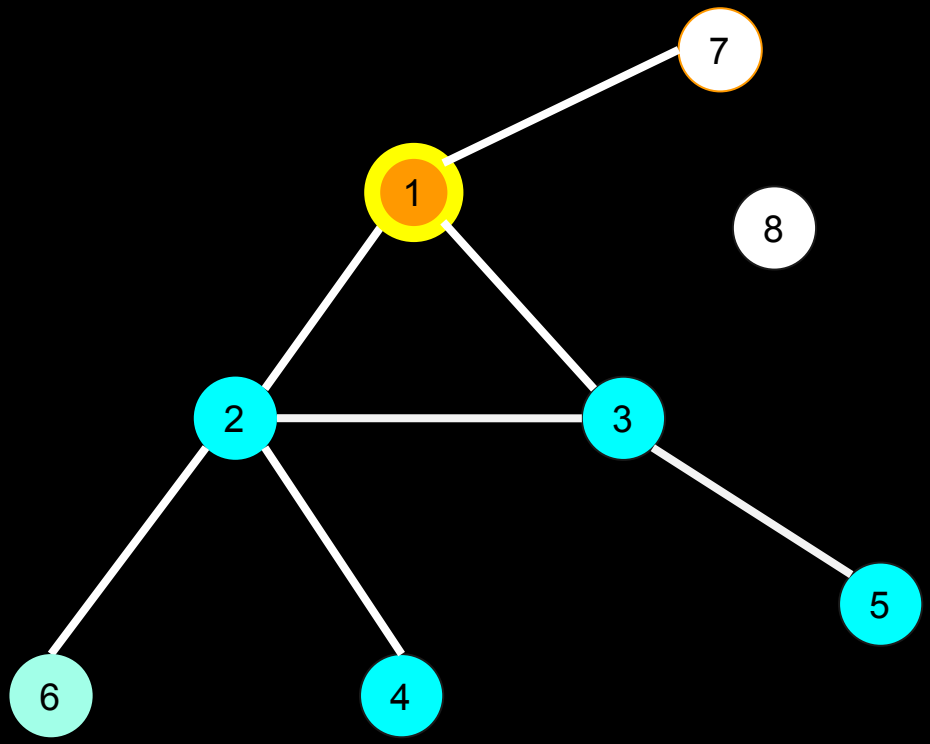
1 2



DFS (Depth-First Search)

Stack:

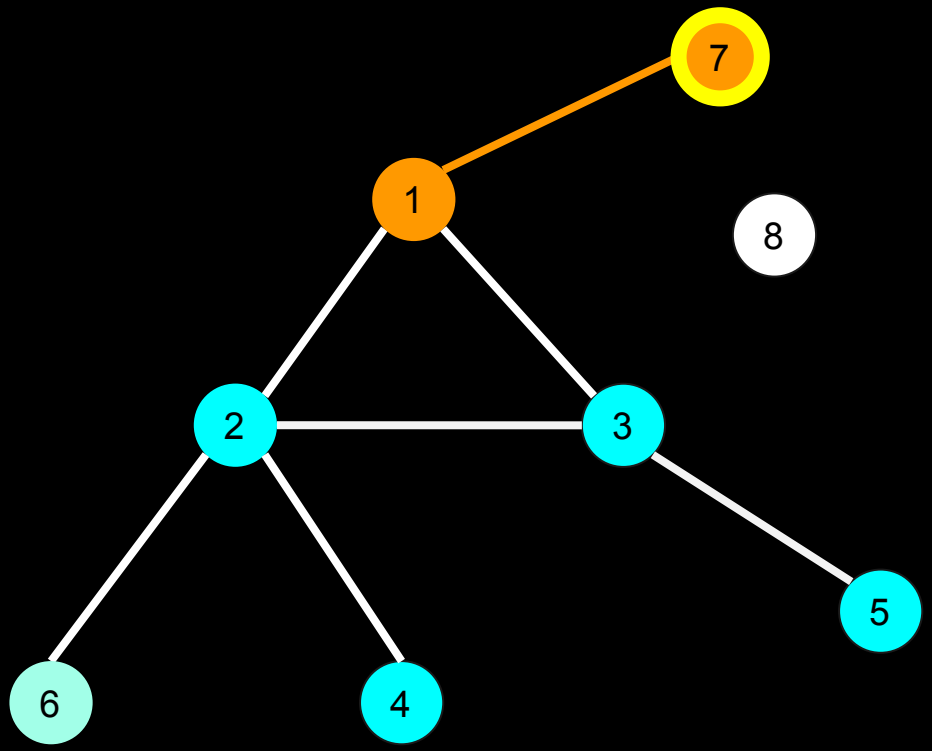
1



DFS (Depth-First Search)

Stack:

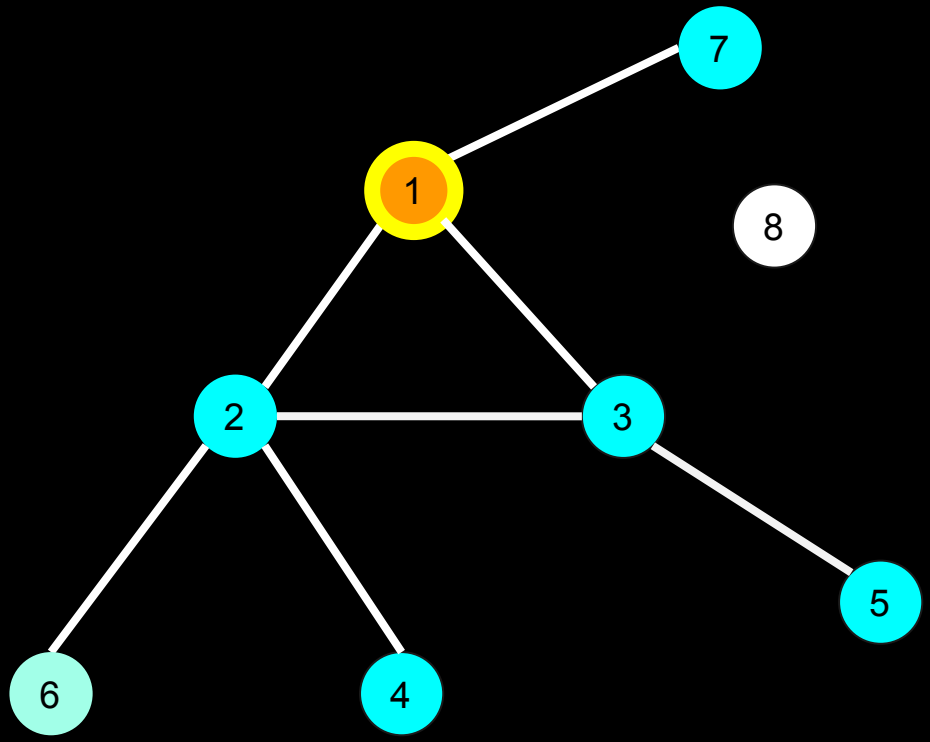
1 7



DFS (Depth-First Search)

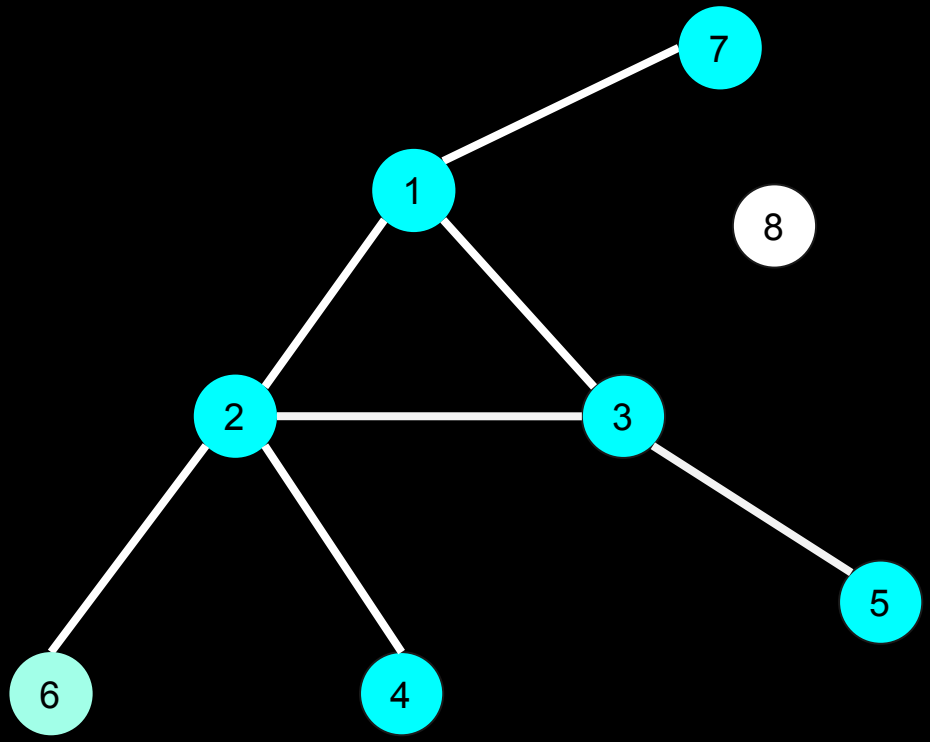
Stack:

1



DFS (Depth-First Search)

Stack:

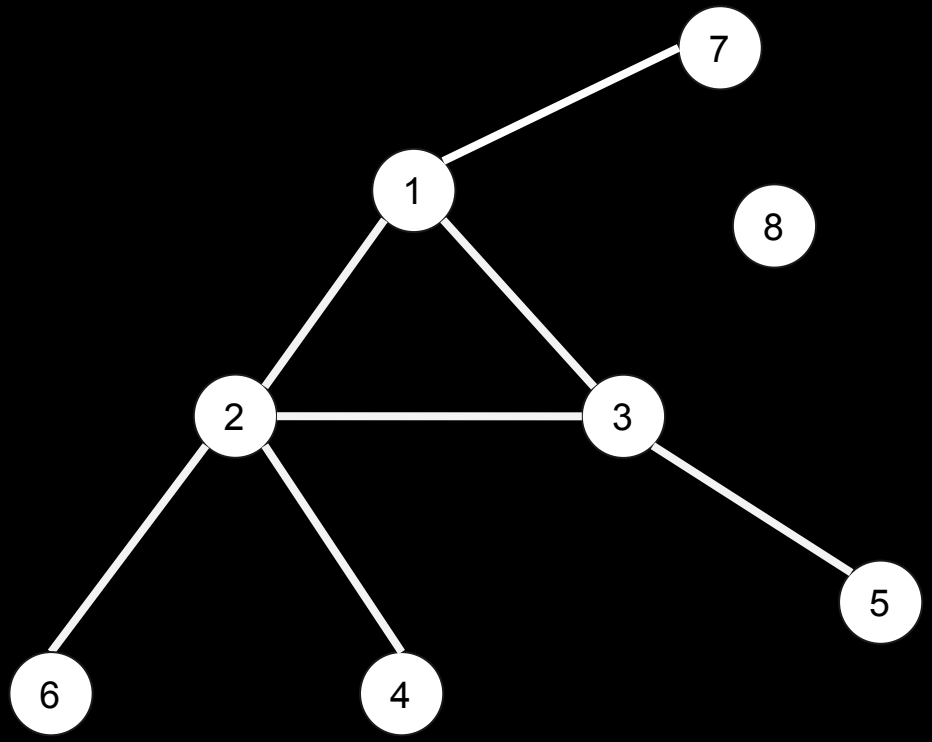


Probleminha

Dado um grafo, se existir algum caminho do vértice 1 até X, dizer o menor caminho entre eles.

BFS (Breadth-First Search)

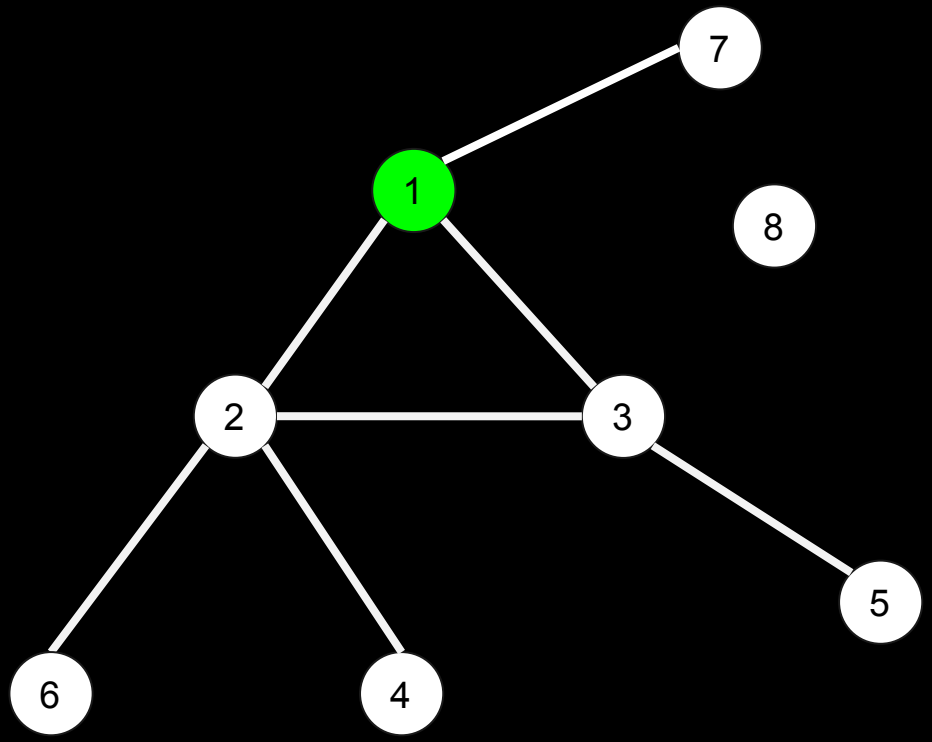
Queue:



BFS (Breadth-First Search)

Queue:

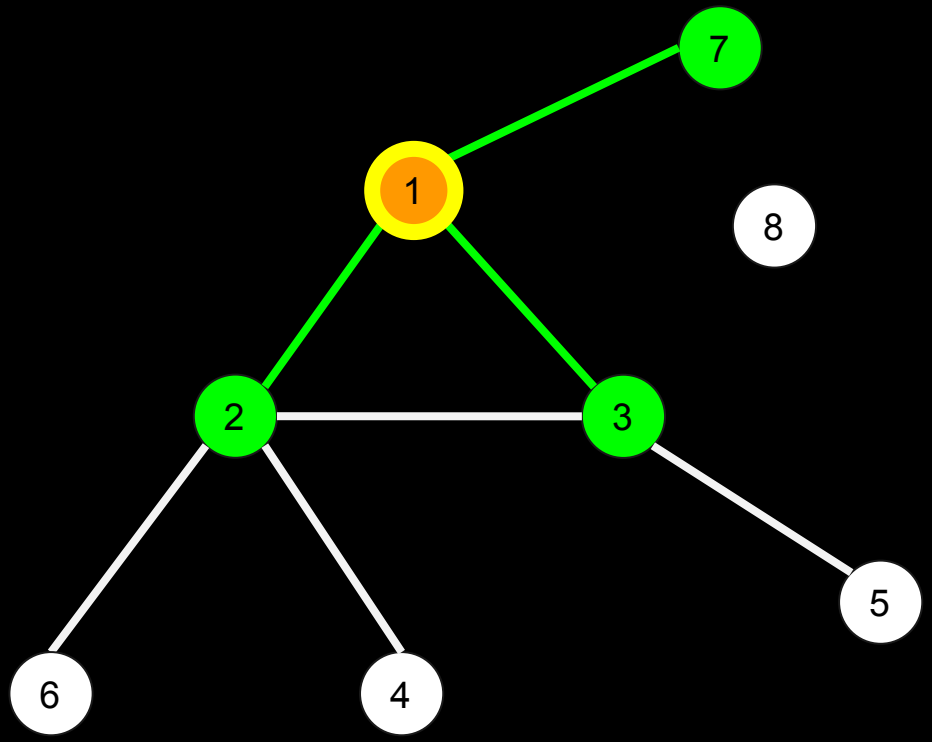
1



BFS (Breadth-First Search)

Queue:

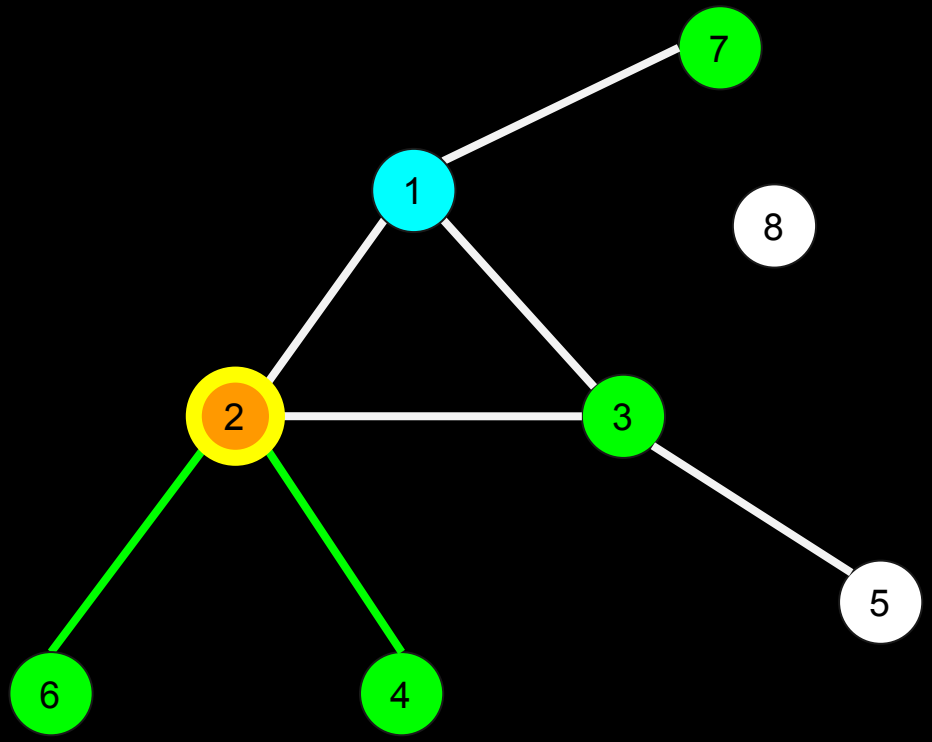
2 3 7



BFS (Breadth-First Search)

Queue:

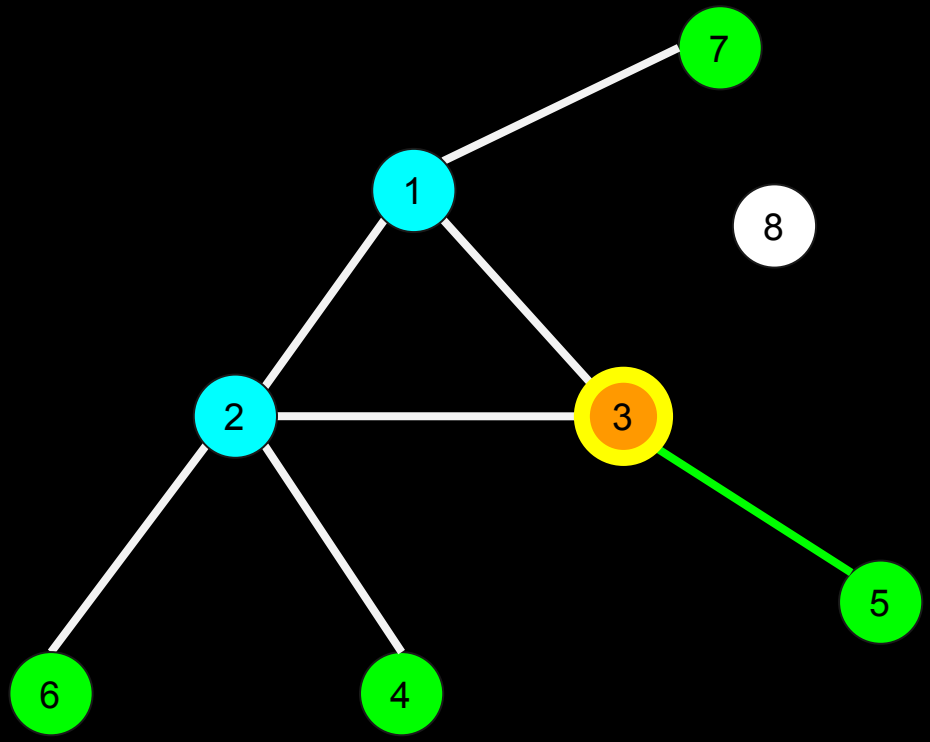
3 7 6 4



BFS (Breadth-First Search)

Queue:

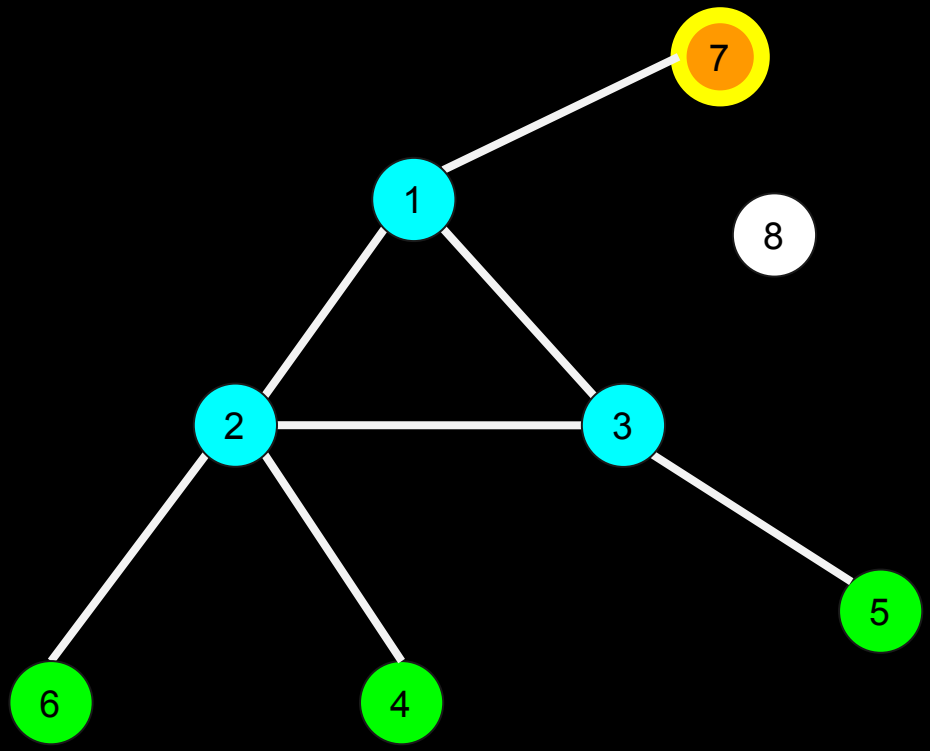
7 6 4 5



BFS (Breadth-First Search)

Queue:

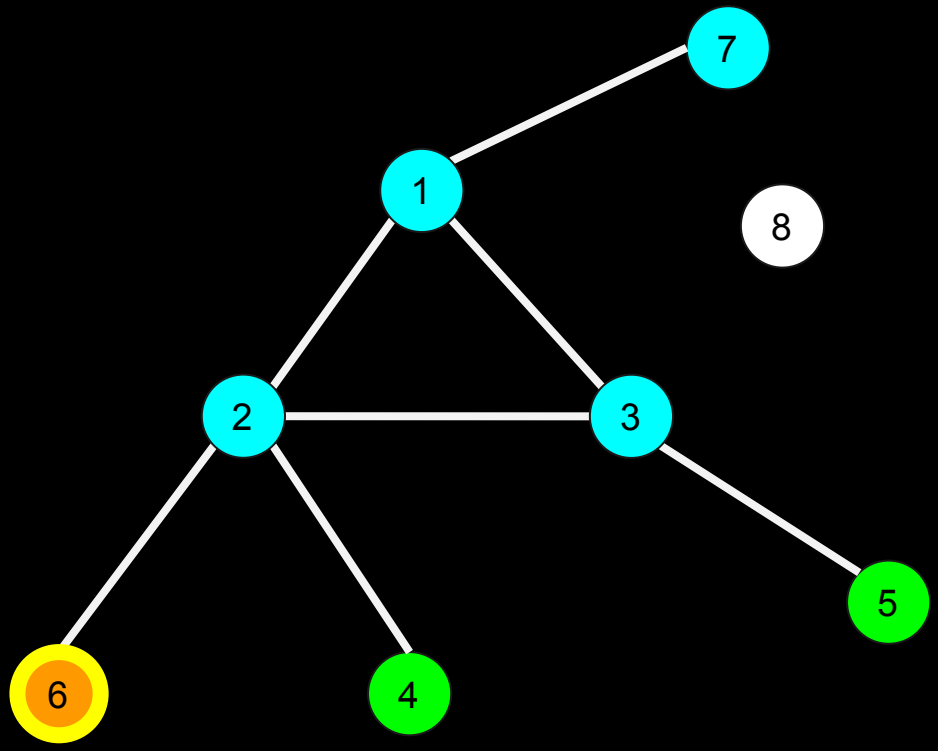
6 4 5



BFS (Breadth-First Search)

Queue:

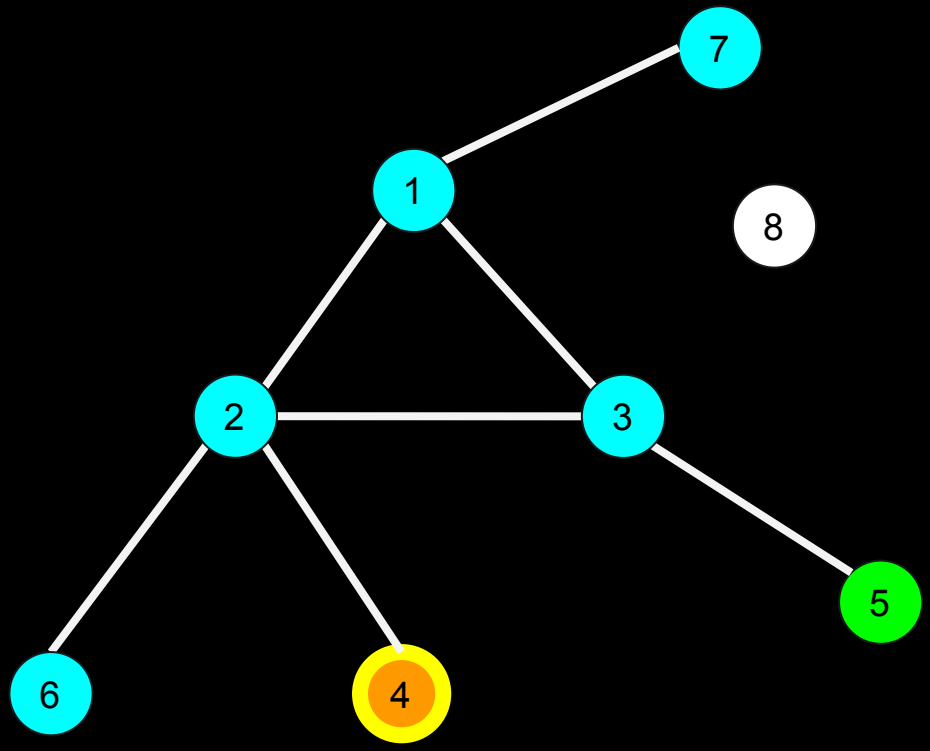
4 5



BFS (Breadth-First Search)

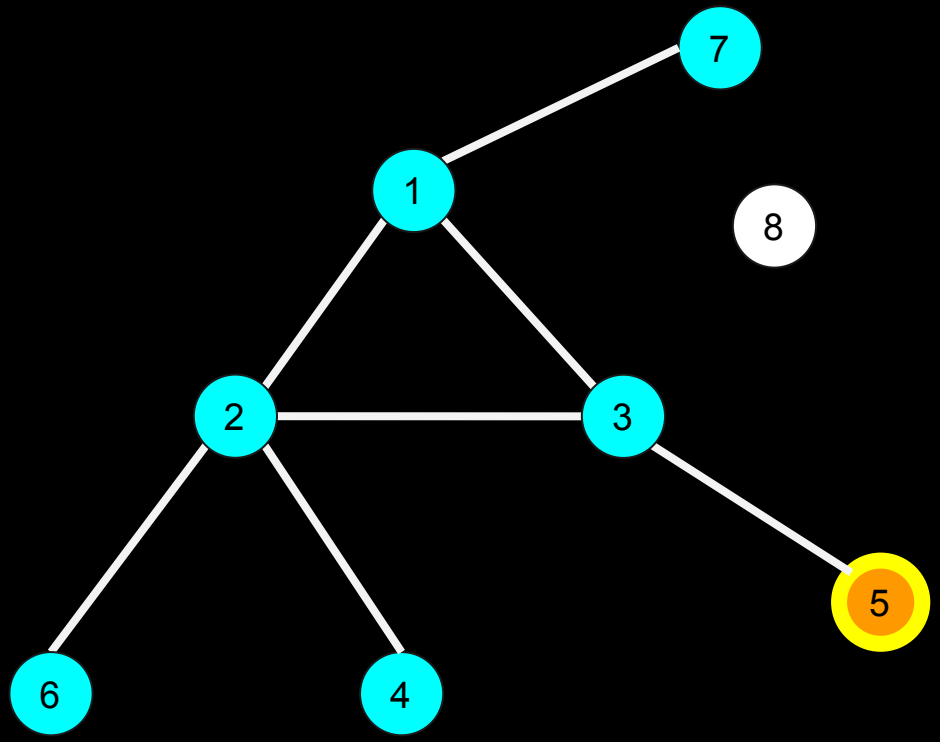
Queue:

5



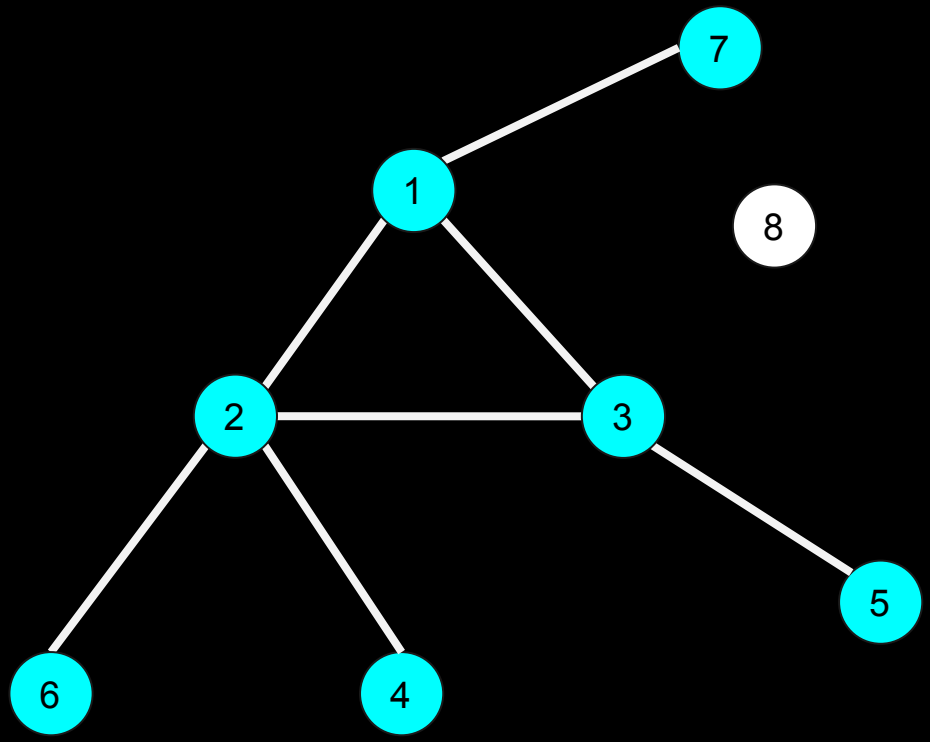
BFS (Breadth-First Search)

Queue:



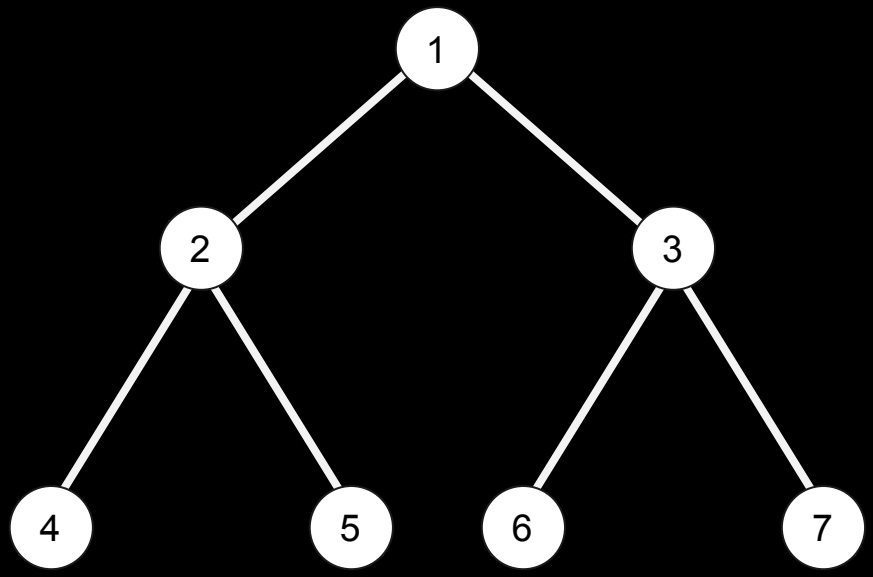
BFS (Breadth-First Search)

Queue:



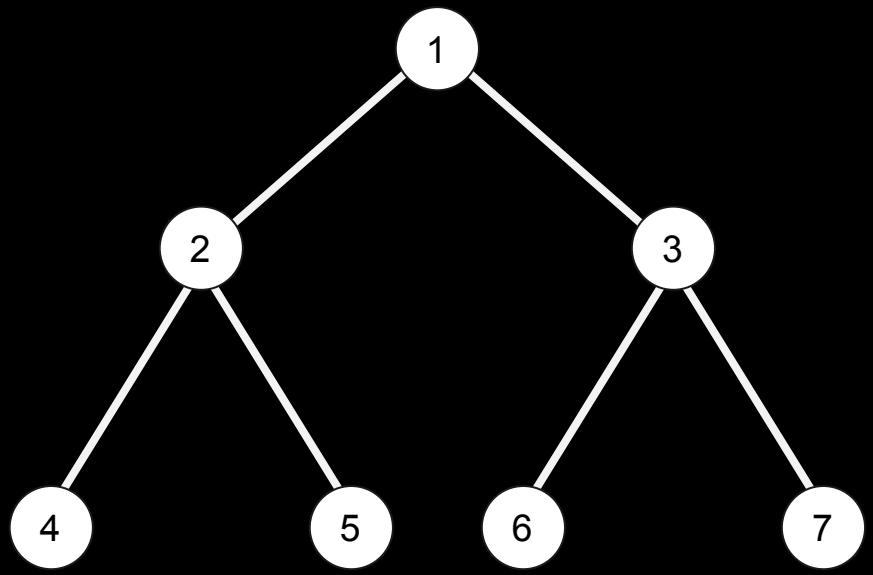
Uma árvore

- Grafo não direcionado
- Sem ciclos
- N nodos
- N-1 arestas
- Totalmente conexo



Uma árvore

- Minimamente conexo
  - Para cada par de nodos, existe um caminho único
- Diâmetro
  - Maior distância entre qualquer par de nodos na árvore
- Centro
  - Nodo que minimiza a maior distância partindo dele



DSU (Disjoint Set Union)

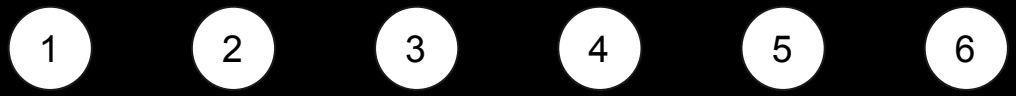
Estrutura de dados que nos permite verificar componentes conexas de forma dinâmica

Operações

- União: Une dois conjuntos
- Find: Verificar em qual conjunto dado elemento está

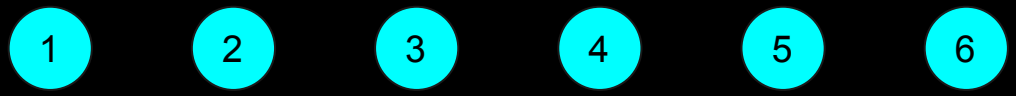
DSU (Disjoint Set Union)

- Cada conjunto tem um representante
- Representamos um conjunto como uma árvore enraizada no representante



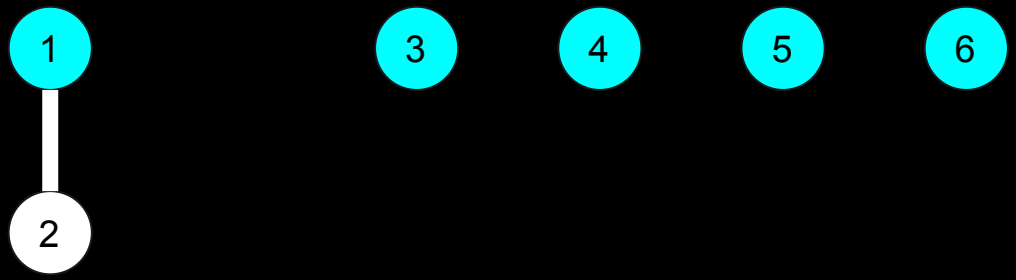
DSU (Disjoint Set Union)

- União
- Cada elemento começa como representante do próprio conjunto



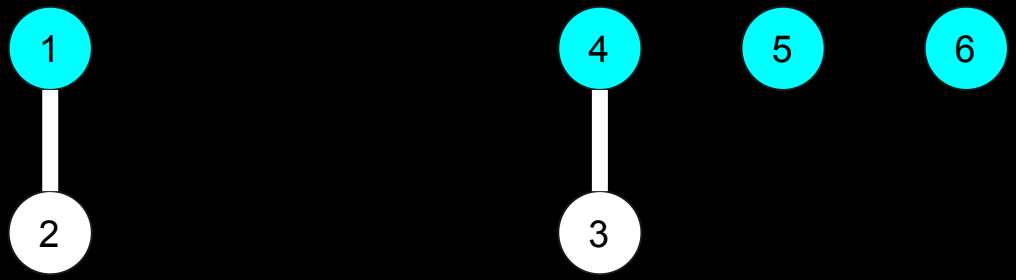
DSU (Disjoint Set Union)

- União(1,2)
- União(4,3)



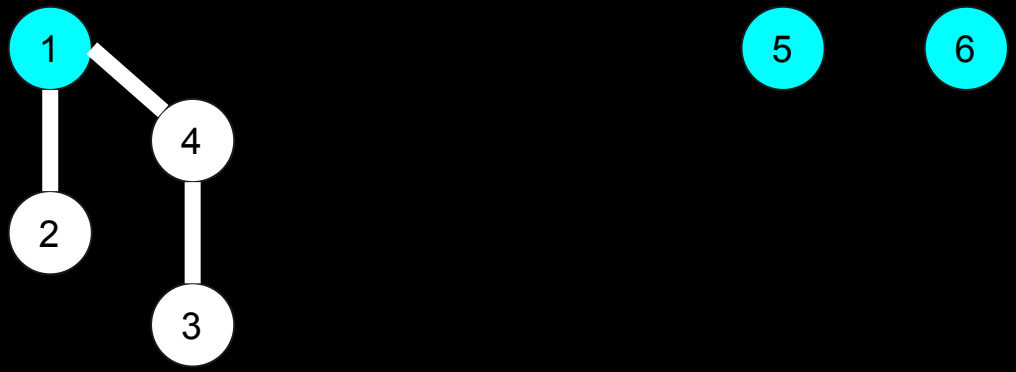
DSU (Disjoint Set Union)

- União(1,2)
- União(4,3)
- União(2,3)



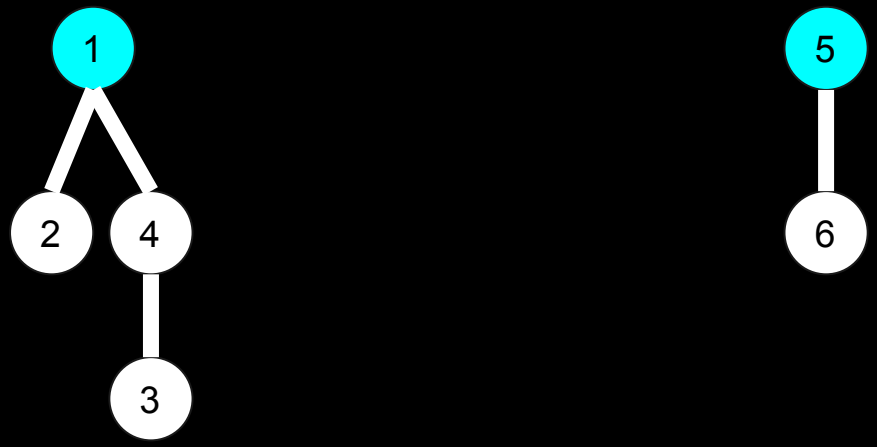
DSU (Disjoint Set Union)

- União(1,2)
- União(4,3)
- União(2,3)
- União(5,6)



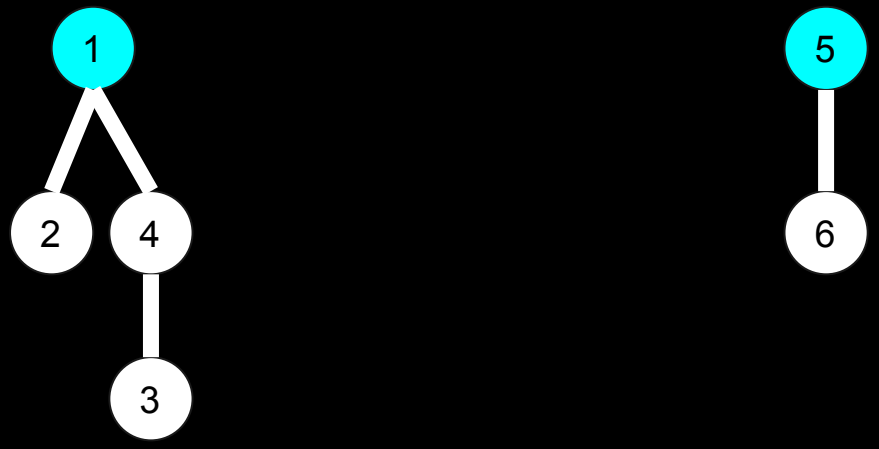
DSU (Disjoint Set Union)

- União(1,2)
- União(4,3)
- União(2,3)
- União(5,6)



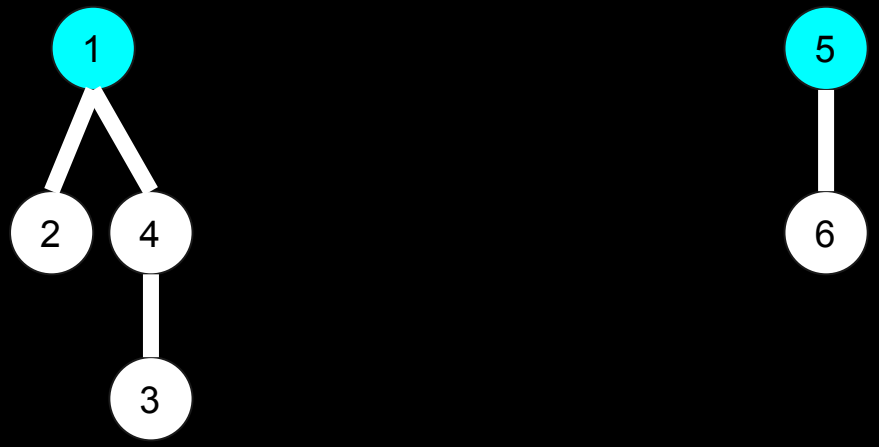
DSU (Disjoint Set Union)

- “Find”
- A resposta do Find é o representante do conjunto



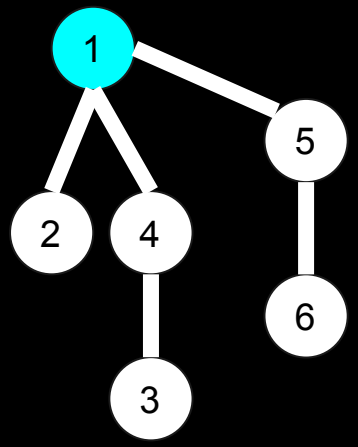
DSU (Disjoint Set Union)

- União por tamanho da árvore
- A menor árvore é ligada na maior
- União(1,6)



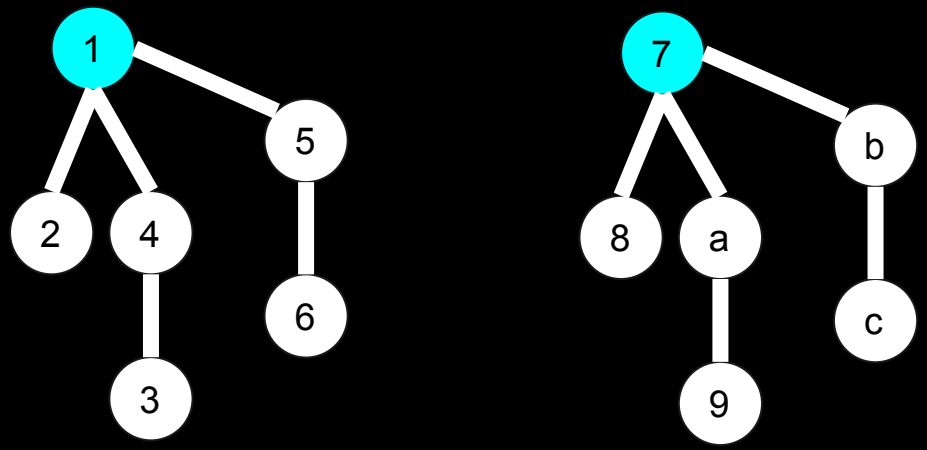
DSU (Disjoint Set Union)

- União por tamanho da árvore
- A menor árvore é ligada na maior
- União(1,6)



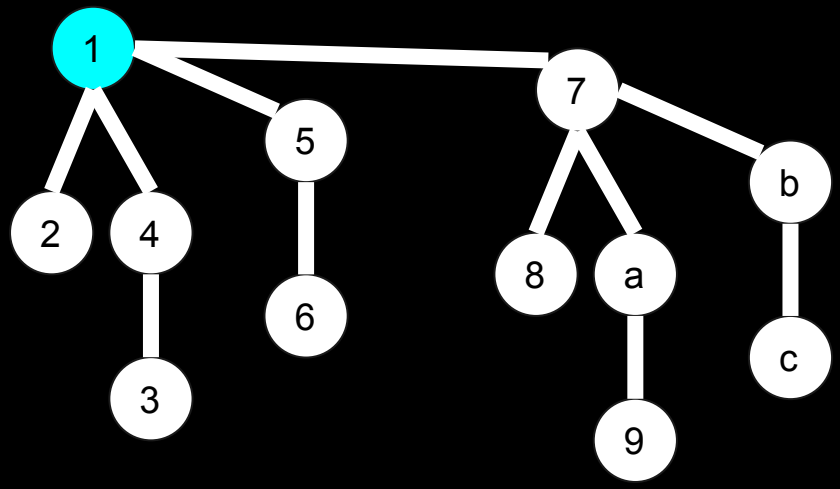
DSU (Disjoint Set Union)

- União por tamanho da árvore
- A menor árvore é ligada na maior
- União(1,6)
- União (1,7)



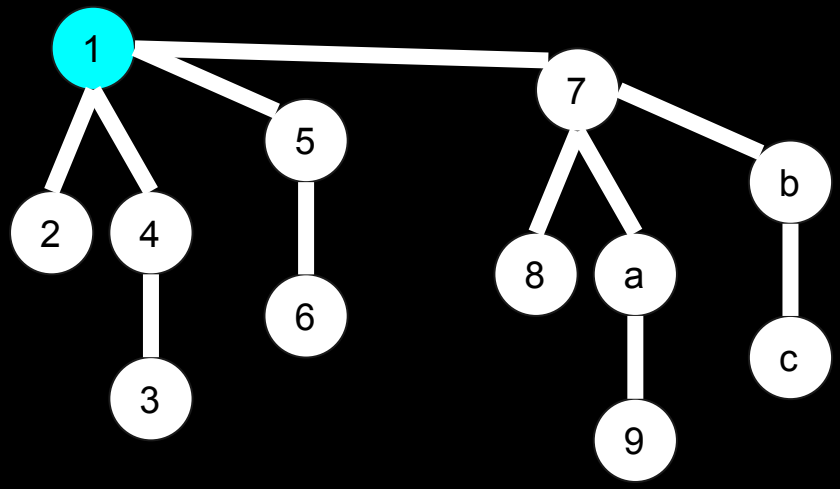
DSU (Disjoint Set Union)

- União por tamanho da árvore
- A menor árvore é ligada na maior
- União(1,6)
- União (1,7)



DSU (Disjoint Set Union)

- No pior caso, eu dobro o tamanho da árvore
- A maior altura, aumenta em 1
- Cada árvore terá altura na ordem de log2(X), onde X é o número de elementos no conjunto



Representação de Grafo no código

Geralmente os inputs são da forma:

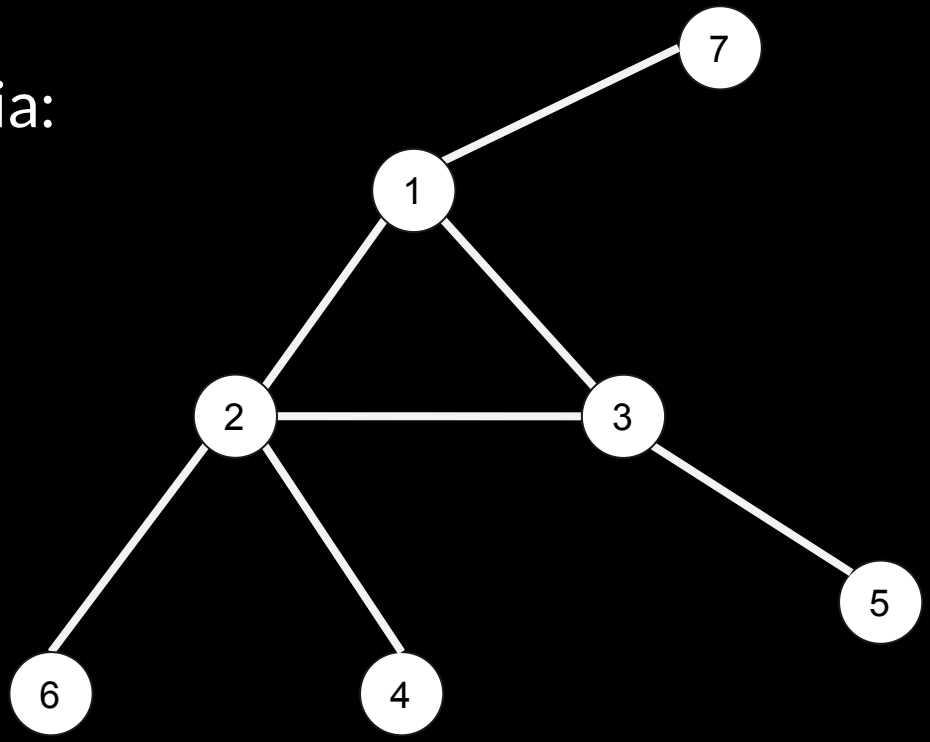
n m
1 2
2 3
2 4
1 5
5 6
...

n: número de nodos
m: número de arestas

As próximas **m** linhas contém 2 inteiros, indicando as ligações

Representação de Grafo no código

Lista de adjacência:
1: { 2, 3, 7 }
2: { 1, 3, 4, 6 }
3: { 1, 2, 5 }
4: { 2 }
5: { 3 }
6: { 2 }
7: { 1 }



pegar o input (bidirecionado)

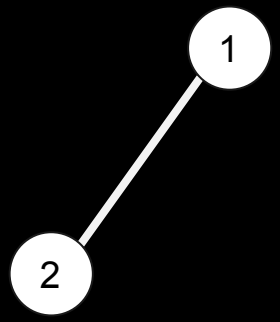
```cpp
const int maxn = 1e5 + 5;
vector<int> Adj[maxn];
bool vis[maxn];

void dfs(int u) {
    ...
}

int main() {
    int n, m; cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        Adj[u].push_back(v);
        Adj[v].push_back(u);
    }
    dfs(1);
}
```

```cpp
for(int i = 0; i < m; i++){
    int a, b; cin >> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
}
```

1:
2:
3:
4:
5:

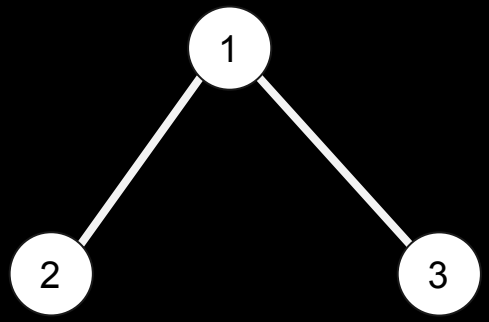
```
for(int i = 0; i < m; i++){
    int a, b; cin >> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
}
```

1: { 2 }
2: { 1 }
3:
4:
5:



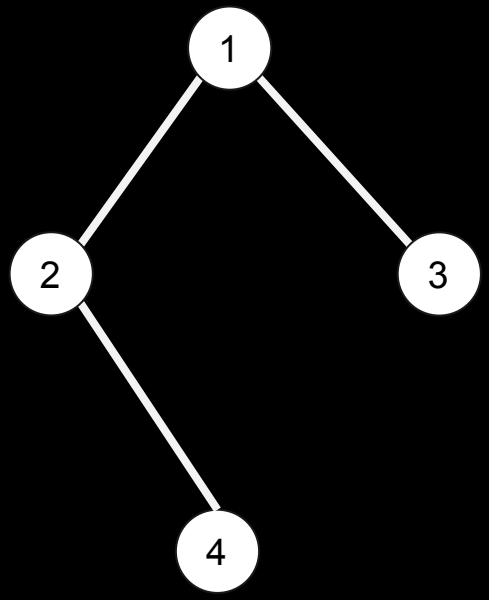
```
for(int i = 0; i < m; i++){
    int a, b; cin >> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
}
```

1: { 2, 3 }
2: { 1 }
3: { 1 }
4:
5:



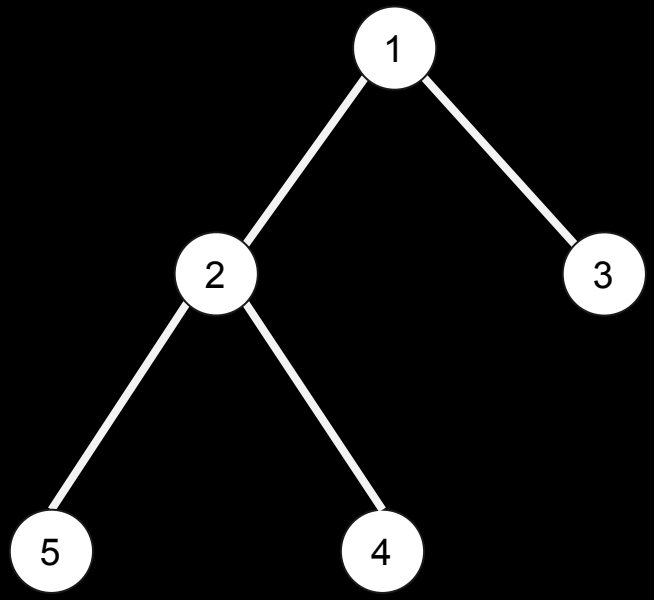
```cpp
for(int i = 0; i < m; i++){
    int a, b; cin >> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
}
```

1: { 2, 3 }
2: { 1, 4 }
3: { 1 }
4: { 2 }
5:



```cpp
for(int i = 0; i < m; i++){
    int a, b; cin >> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
}
```

1: { 2, 3 }
2: { 1, 4 }
3: { 1 }
4: { 2 }
5: { 2 }



DFS

```cpp
const int maxn = 1e5 + 5;
vector<int> Adj[maxn];
bool vis[maxn];

void dfs(int u) {
    vis[u] = true;
    for (int v : Adj[u]) {
        if (vis[v]) continue;
        dfs(v);
    }
}
```

BFS

```cpp
const int maxn = 1e5 + 5;
vector<int> Adj[maxn];
int dis[maxn];

void bfs(int root) {
    queue<int> q;
    q.push(root);
    dis[root] = 0;
    while(!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : Adj[u]) {
            if (dis[u] + 1 < dis[v]) {
                dis[v] = dis[u] + 1;
                q.push(v);
            }
        }
    }
}
```

DSU

```cpp
vector<int> pai;
vector<int> tamanho;

void inicializa_dsu(int n) {
    tamanho.assign(n,1);
    pai.assign(n,1);
    for(int i=0;i<n;i++) pai[i] = i;
}
int find(int a) {
    if(pai[a]==a) {
        return a;
    }
    return find(pai[a]);
}
void uniao(int a, int b) {
    int ra = find(a);
    int rb = find(b);
    if(ra == rb) return;
    if(tamanho[rb]>tamanho[ra]) swap(ra,rb);
    pai[rb] = ra;
    tamanho[ra]+=tamanho[rb];
}
```

Obrigado

t.me/bruteudesc
instagram.com/bruteudesc
bruteudesc.com

Segunda a Quinta
F304
Sextas-feiras
F301

Problemas

brute.linkh.at/grafos

senha: brute