Introduction To Graph Theory

Part 1

- Terminologies, Graph Representations
- The DFS Algorithm

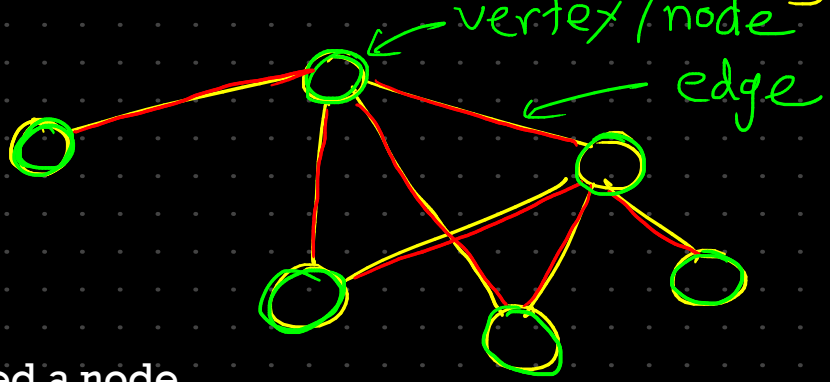
What is a Graph?

G(V, E) is a set of vertices V and a set of edges E.
Each edge e in E has the form e = (u, v) where u, v are two vertices in V

$V = \{ Dhaka, Chattogram, Khulna, \dots \}$

$E = \{ (Dhaka, Chattogram), (Dhaka, Khulna), \dots \}$

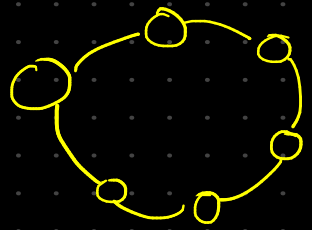
$G(V, E)$

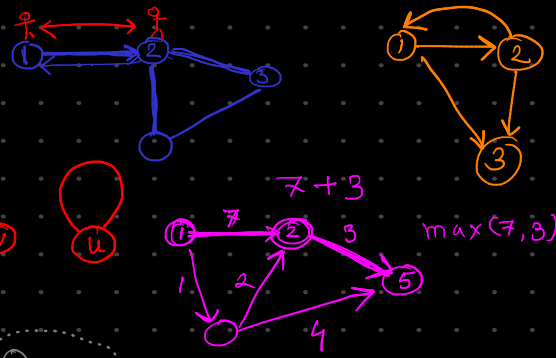


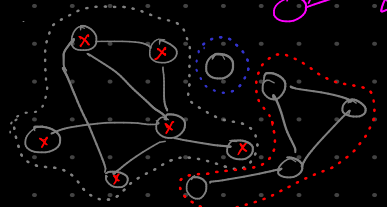
Vertex (plural vertices) is also called a node.

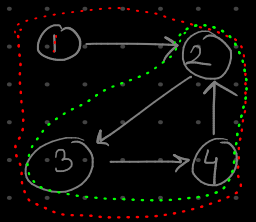
More Terminology

1. Undirected Graphs
2. Directed Graphs
3. Weighted Graphs
4. Multi Edge & Self Loops
5. Components (undirected)
6. Paths & Cycles
7. Degree of a node

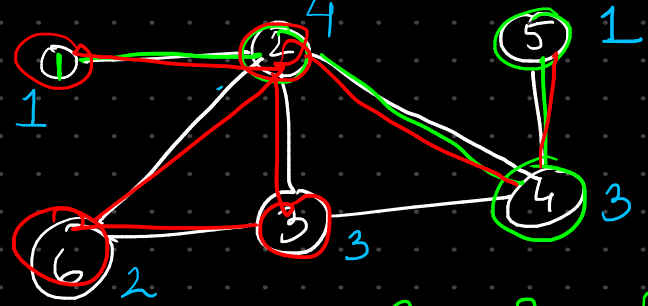


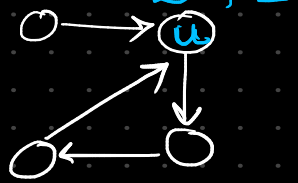






trail
walk



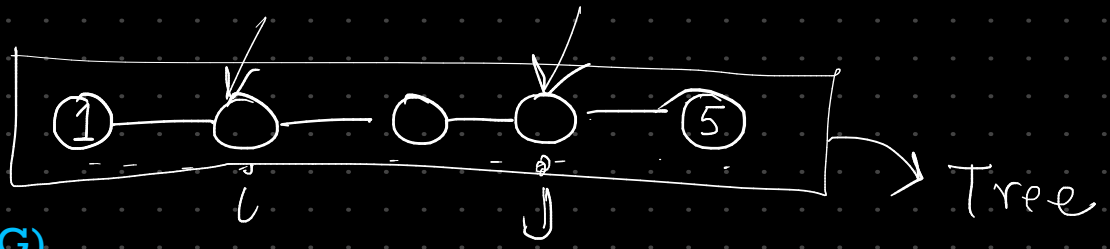


path : edge & node at most once

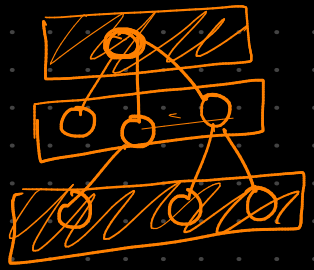
1 - 2 - 3 - 2 - 4 - 5

Interesting Types of Graphs:

1. Path Graph



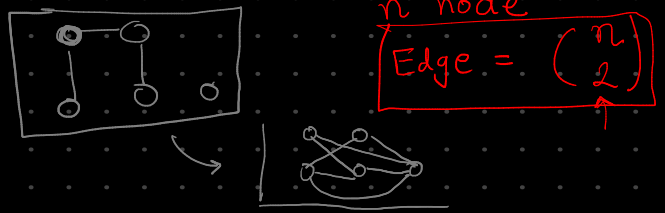
2. Directed Acyclic Graph (DAG)



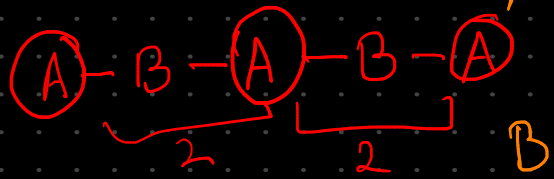
3. Undirected Acyclic Graph

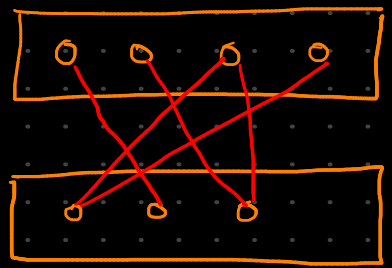
4. Complete Graph

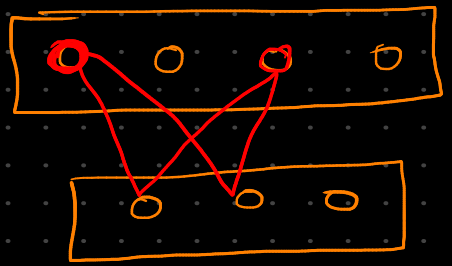
5. Complement of a Graph

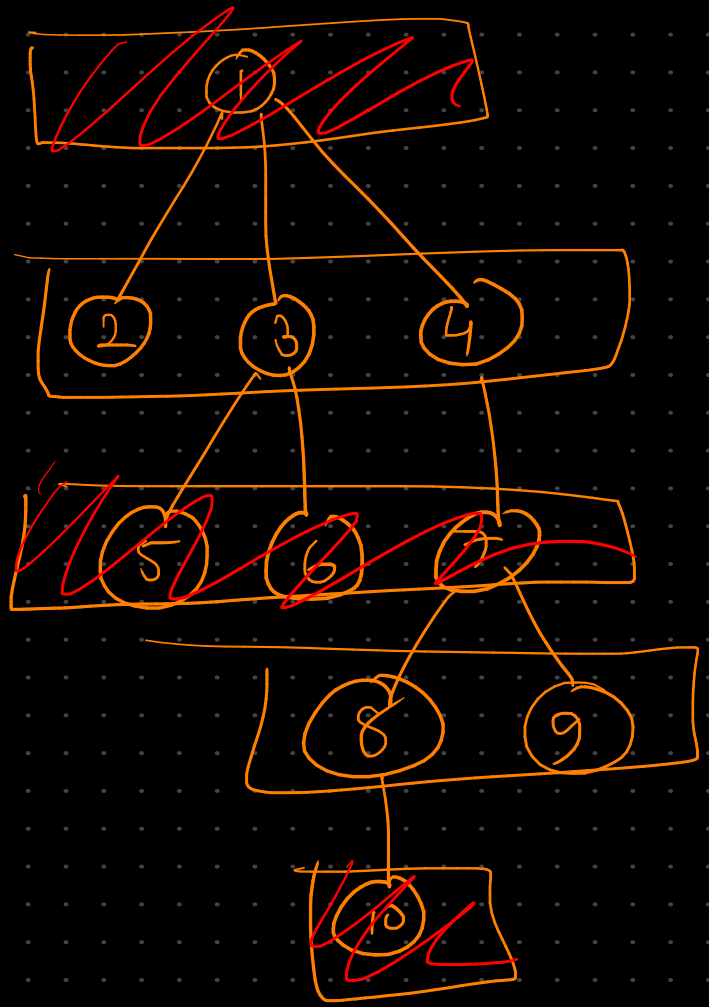


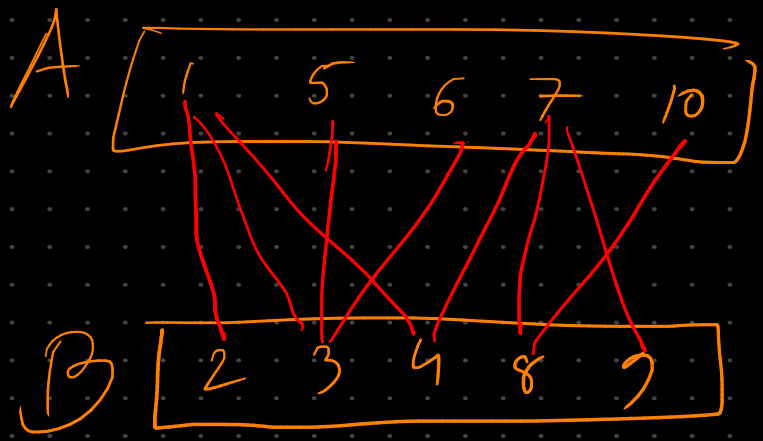
6. Bipartite Graph





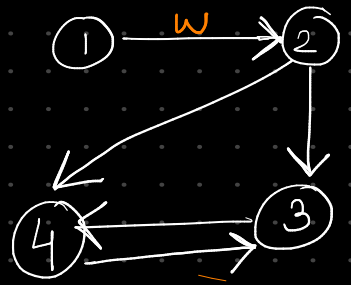



1
2
3
4
5
6
7
8
9
10


A
1
5
6
7
10
B
2
3
4
8
9

How to represent a graph in code?

1. Adjacency Matrix
2. Adjacency List
3. Edge List



$n \leq 2 \times 10^5$

Matrix $n \times n$ ← # of nodes

$O(n^2)$ memory

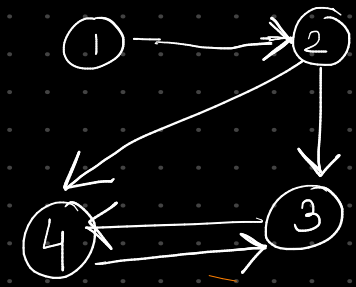
	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	0	0	0	1
4	0	0	1	0

adj[N][N]

for each edge u,v :

adj[u][v] = 1

Adj. List



1 → {2}
2 → {4, 3}
3 → {4}
4 → {3}

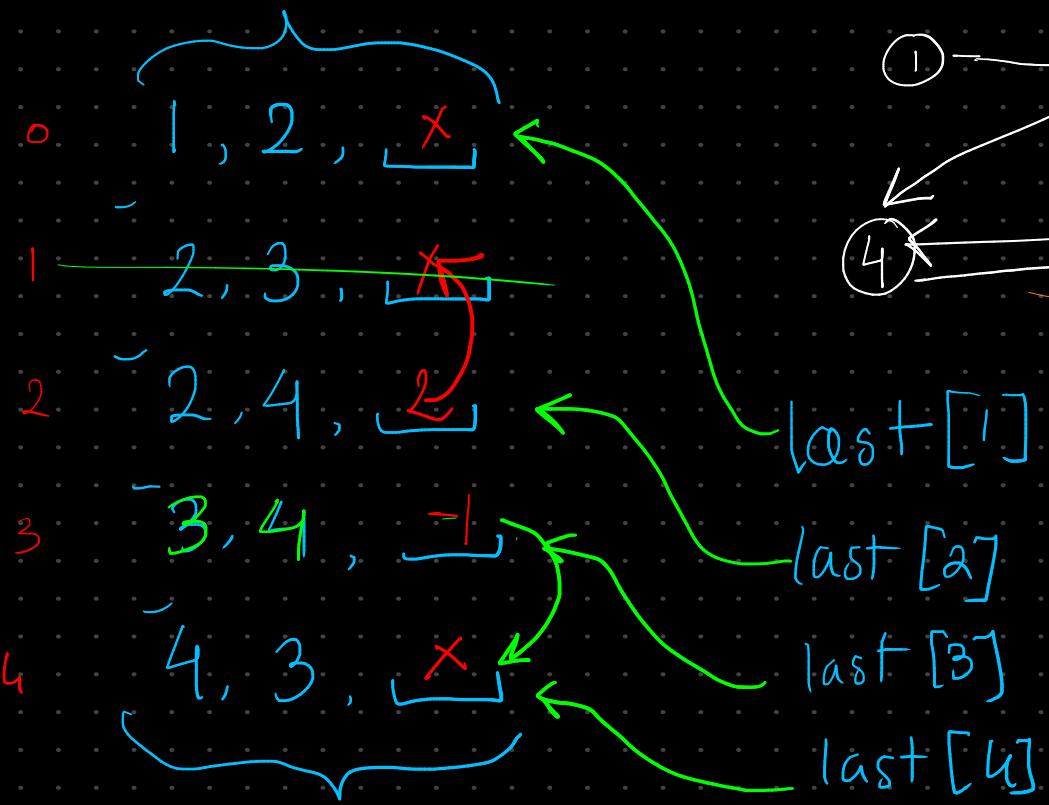
$O(V+E)$

```
vector<int> g[MAXN];

for each edge (u,v):
    g[u].push_back(v);
```

Edge List

```
Edge {
    int from, to;
    int prv;
}
```

Edge E[MAXE]

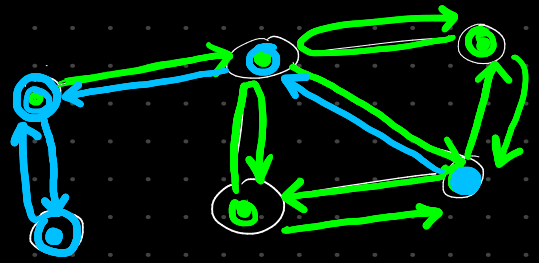


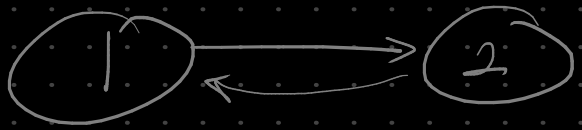
v i:=last[v]

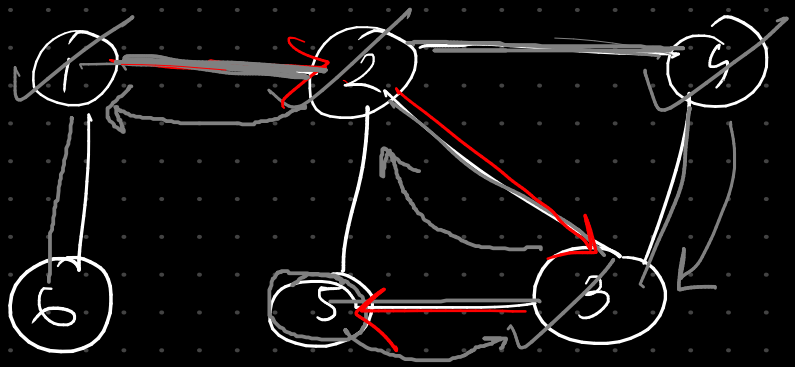
i:=Edge[i].prv

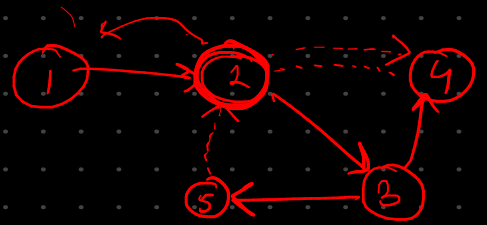
The Depth First Search (DFS) Algorithm

- An algorithm to traverse a graph.
- Imagine you are in a new city and wish to look around every single roads and landmarks. How would you go about doing it?





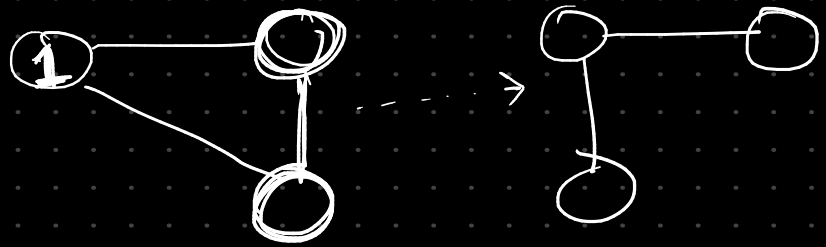




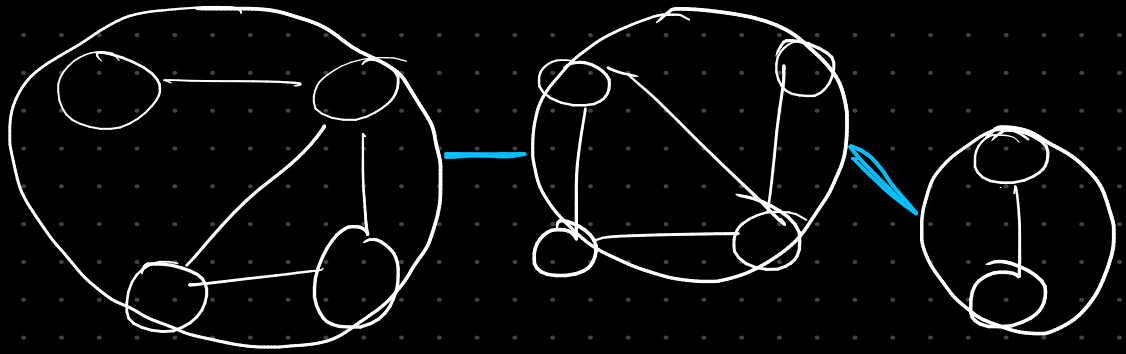
4 2

DFS

```
void dfs (int u) {
    visited[u] = true;
    for (int v : g[u]) {
        if (visited[v] == false) {
            dfs(v);
        }
    }
}
```



Finding Number of Components in an Undirected Graph

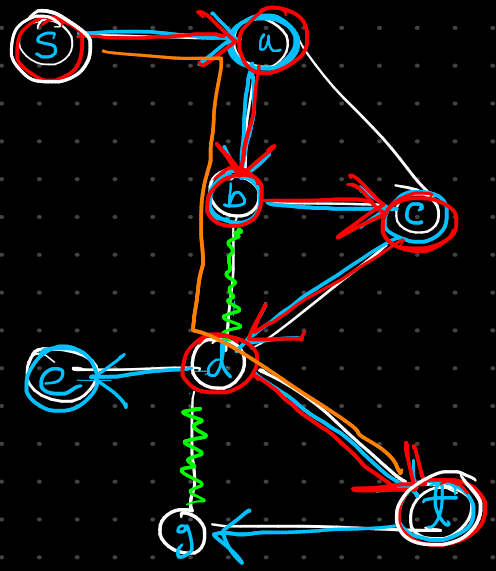


3 components

#components is precisely the number of dfs calls we need across all nodes.

```
int ncomp = 0;
for (int u = 1; u <= n; ++u) {
    if (!visited[u]) {
        dfs(u);
        ++ncomp;
    }
}
cout << ncomp << "\n";
```

Finding any path between two nodes



connected graph

s, t

s → a → b → c → d → t

path = {t, d, c, b, a, s}

```
void dfs (int u) {
    visited[u] = true;
    for (int v : g[u]) {
        if (visited[v] == false) {
            pi[v] = u;
            dfs(v);
        }
    }
}
```

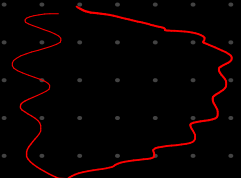
modify the dfs to store π

π[u] = from where I came to u.

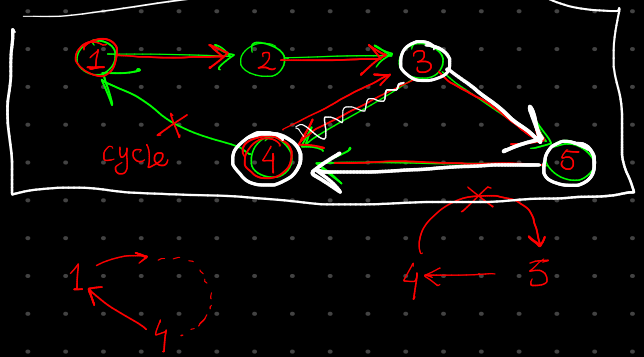
Detecting Cycles (Undirected)

https://cses.fi/problemset/task/1669

```
void dfs (int u) {
    visited[u] = true;
    for (int v : g[u]) {
        if (v == pi[u]) continue;

        if (visited[v] == false) {
            pi[v] = u;
            dfs(v);
        }
        else {  already visited
            // Detected cycle
            cout << "Found a cycle between: " << u << " " << v << "\n";

            int cur = u;
            while (cur != v) {
                cout << cur << ", ";
                cur = pi[cur];
            }
            cout << cur << "\n";
        }
    }
}
```



Detecting Cycles (Directed)



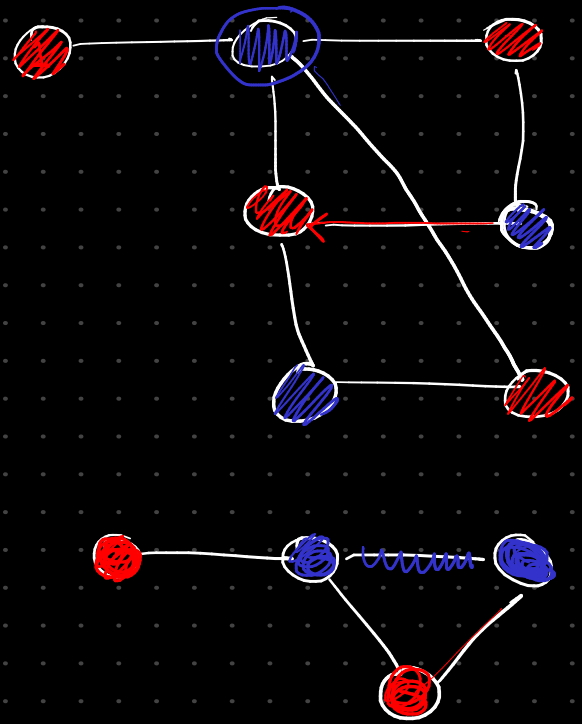
Not all previously visited nodes give a cycle!

When is it a cycle?

if visited[v] == true AND dfs is still running from v.

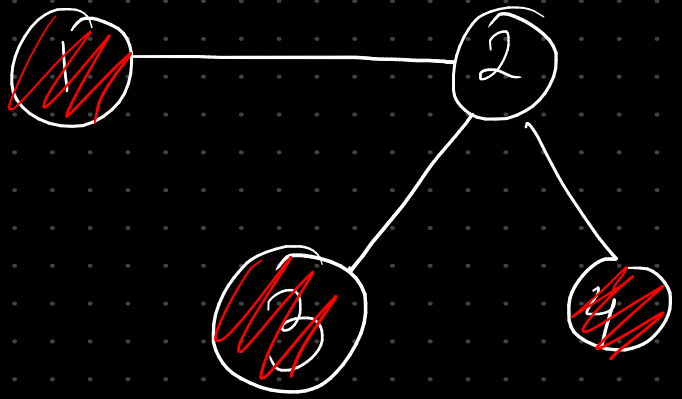
Bicoloring a Graph

https://cses.fi/problemset/task/1668
https://lightoj.com/problem/back-to-underworld



Check bipartite or not

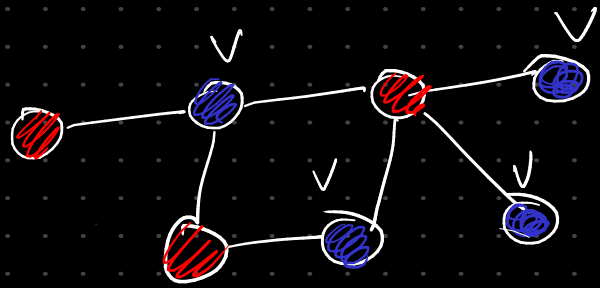
```
void dfs (int u, int col) {
    visited[u] = true;
    color[u] = col;

    for (int v : g[u]) {
        if (!visited[v]) {
            dfs(v, col ^ 1);
        }
        else { // already visited
            if (color[v] == color[u]) {
                // this is not bipartite
            }
        }
    }
}
```



Vampire = 2

Vampire = 3

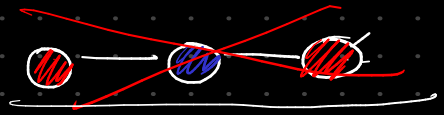
max(R , B)



3 , 4

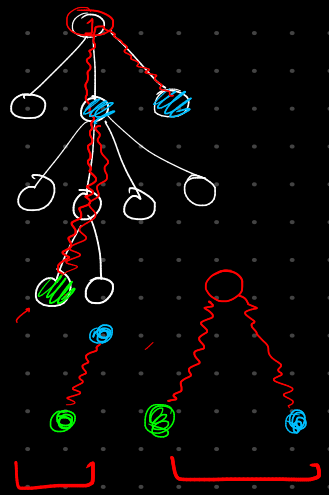
+

2 1



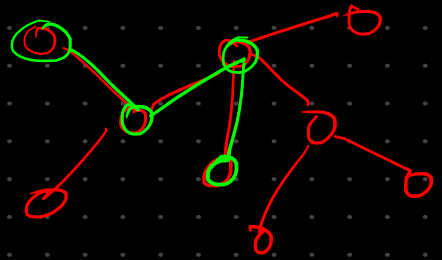
DFS on Trees

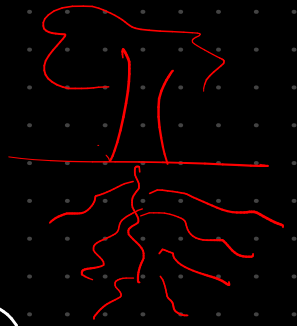
https://cses.fi/problemset/task/1674

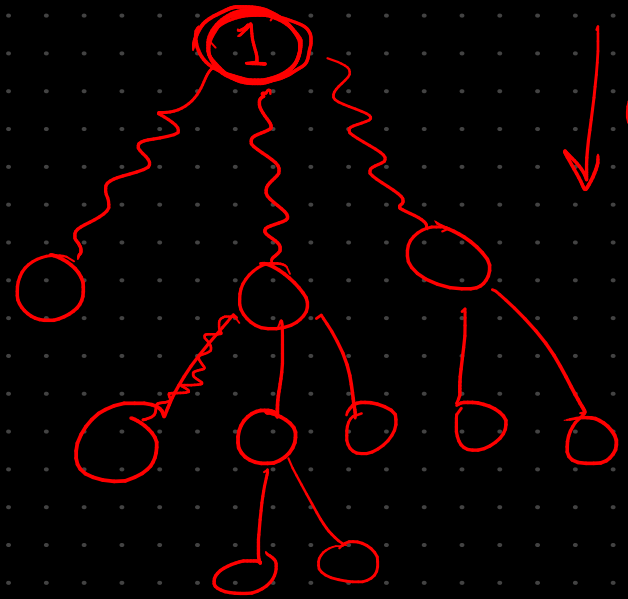


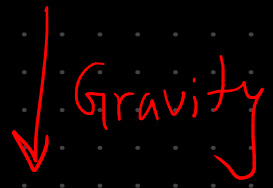
Chain

V type

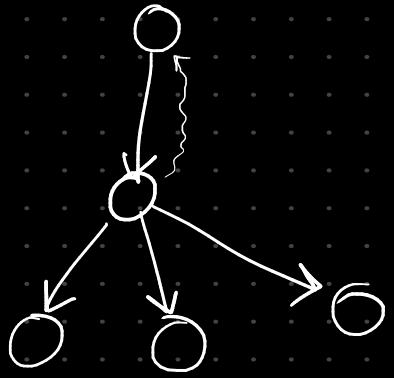


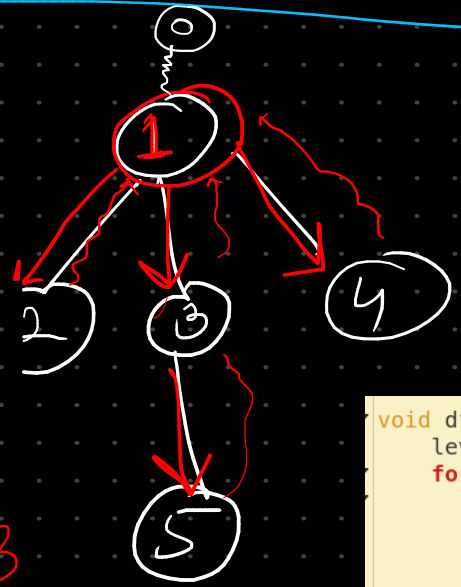






Find shortest distance to all nodes from 1.





Level 1

Level 2

Level 3

dist[1] = 0

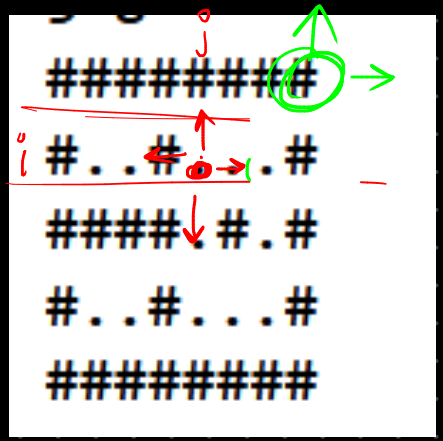
dist[2] = 0+1

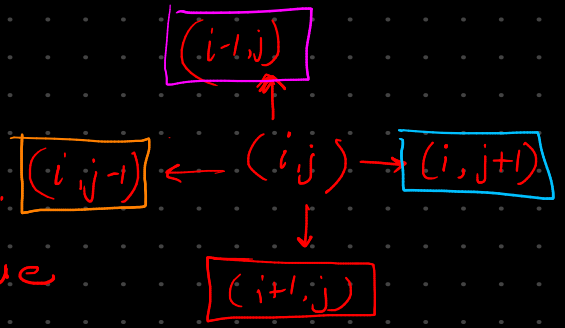
```
void dfs (int u, int f) {
    lev[u] = lev[f] + 1;
    for (int v : g[u]) {
        if (v != f) {
            // or, lev[v] = lev[u] + 1;
            dfs(v, u);
        }
    }
}
```

DFS on 2D Grids

https://cses.fi/problemset/task/1192
https://lightoj.com/problem/guilty-prince

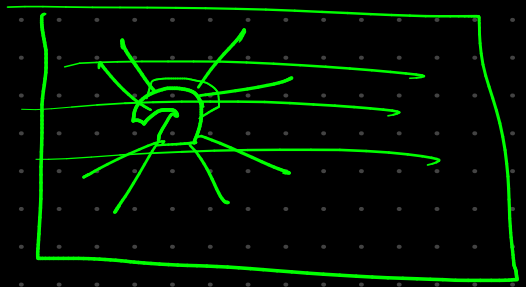
```
int dx[] = {1, 0, -1, 0};
int dy[] = {0, 1, 0, -1};
dfs(r, c) {
    vis[r][c] = true;
    for (int k=0 ; k<4 ; k++) {
        int to_r = r + dx[k];
        int to_c = c + dy[k];
        if(valid(to_r, to_c) == false) continue;
        if (board[to_r][to_c] == '#') continue
```





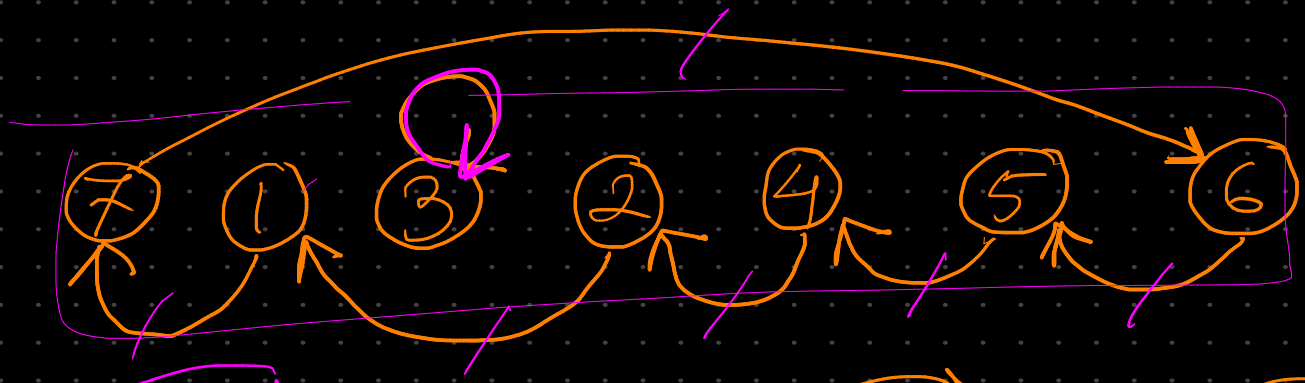
```
    dfs(to_r, to_c);
  }
}
```

```
dx[ ]
dy[ ]
```

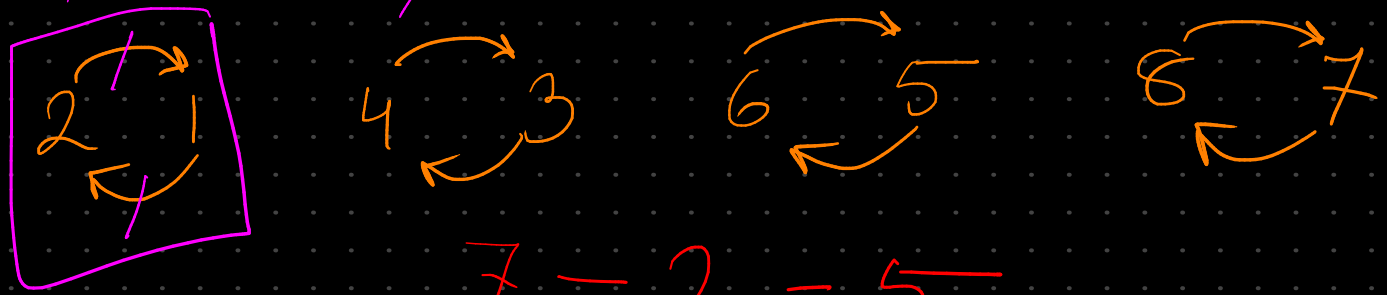


Permutations https://www.hackerrank.com/challenges/minimum-swaps-2/problem

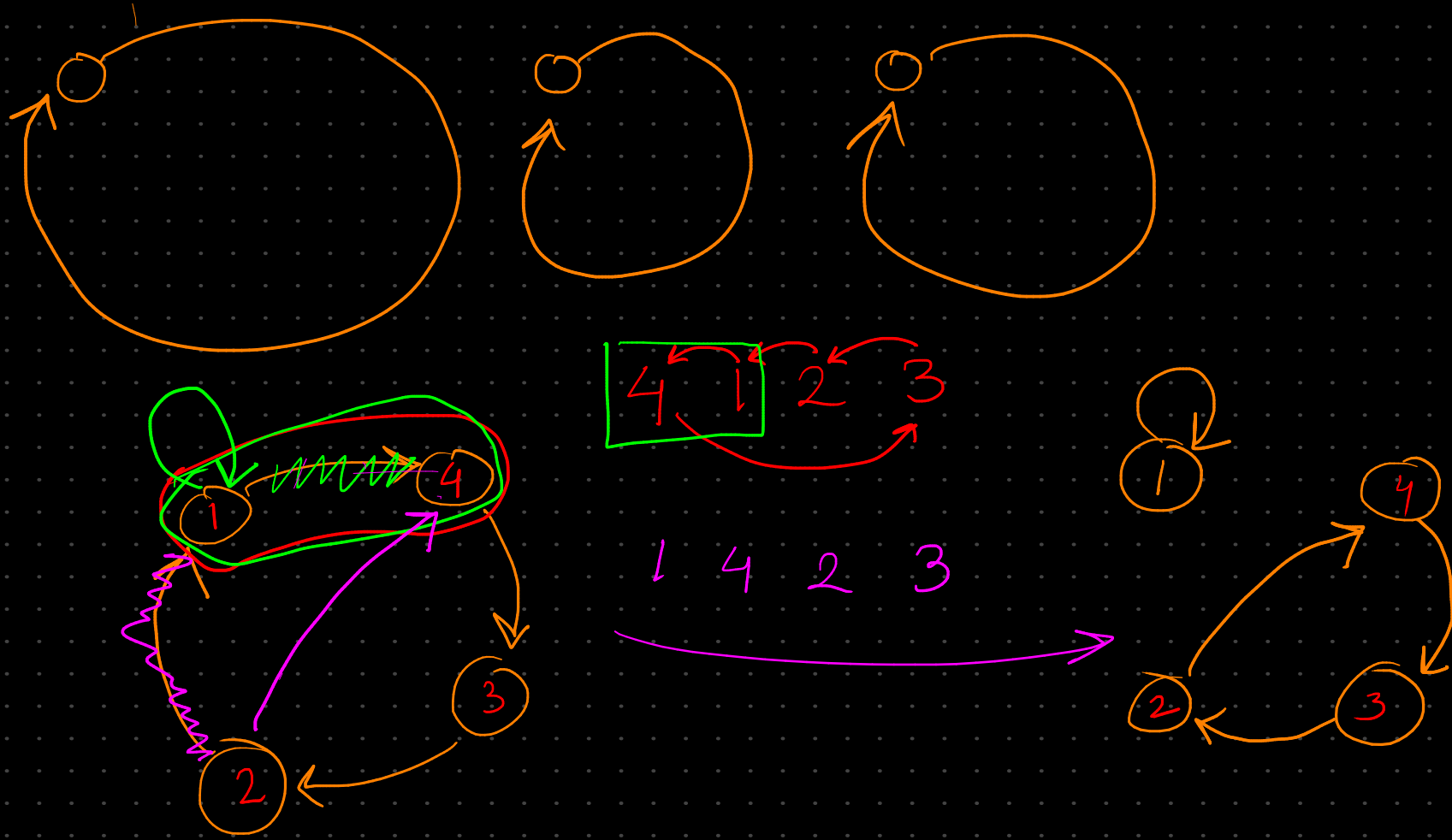
Finding minimum #of moves to sort a perm.



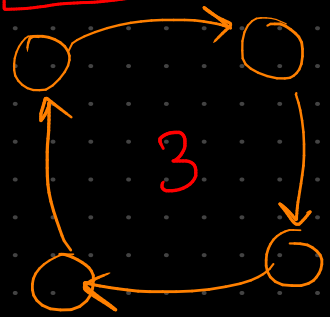
8 edge cycle

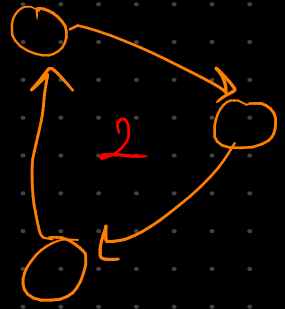


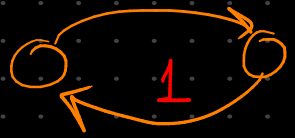
$7 - 2 = 5$

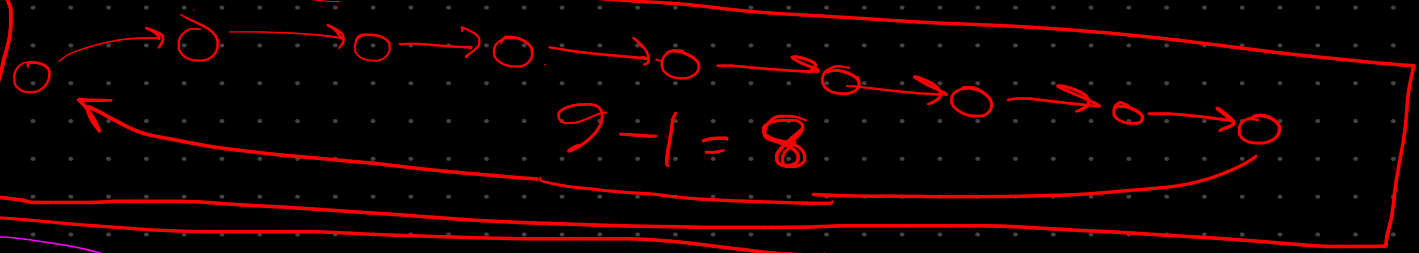

4 1 2 3
1 4 2 3
1
4
2
3

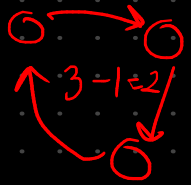
Ans = n − component_count

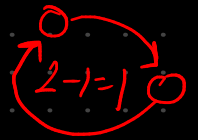


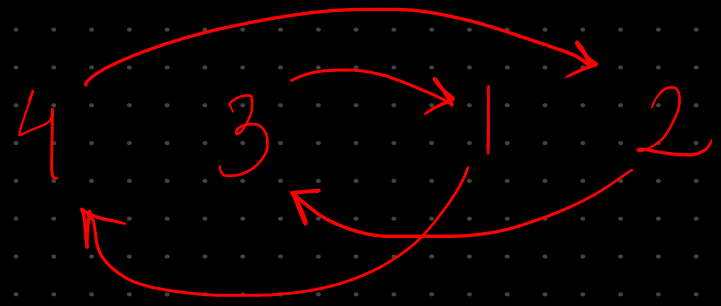




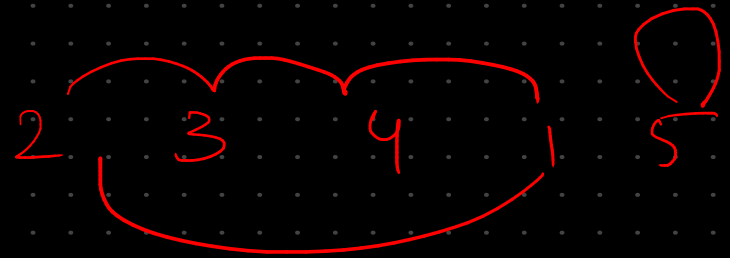




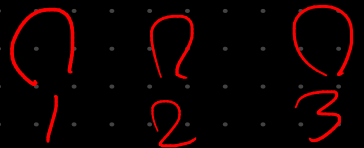




$4-1=3$



$5-2=3$



Small talk on Functional Graphs

A directed graph with n nodes, where each node has outdegree = 1

The Disjoint Set Union (DSU) Data Structure

Maintains a list of disjoint sets. You can at any moment:

- create a new disjoint set
- ask if two items belong in the same set
- join any two sets

Representatives

Path Compression

Union by Rank

Time Complexity

Check if two nodes belong to same component

Maintain size of each component

Though it seems pretty basic, we shall see more cool applications of DSU pretty soon.