DFS (Depth First Search)

Ref: https://cp-algorithms.com/graph/depth-first-search.html

- DFS
- **code**

```
#include <bits/stdc++.h>
#define pb push_back
#define int long long

using namespace std;
using vi = vector<int>;

void dfs(int u, vector<bool>& vis, vector<vector<int>>& adj) {
    if (vis[u])
        return;
    vis[u] = 1;
    cout << u << " ";
    for (int v : adj[u]) {
        dfs(v, vis, adj);
    }

    return;
}

int32_t main() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> adj(n + 1);
    vector<bool> vis(n + 1, 0);

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

    dfs(1, vis, adj); // print the nodes
}
```

- Print number of vertices in the subtree/ print max value in the subtree.

```
int dfs(int u, vector<bool>& vis, vector<vector<int>>& adj) {
    if (vis[u])
        return 0;
    vis[u] = 1;
    int ans = 1;
    for (int v : adj[u]) {
        ans += dfs(v, vis, adj);
    }

    cout << u << ": " << ans << '\n';

    return ans;
}
```

- cycle detection

```
bool isCycle;

void dfs(int u, int par, vector<bool>& vis, vector<vector<int>>& adj) {
    if (isCycle)
        return;

    vis[u] = 1;

    for (int v : adj[u]) {
        if (v == par)
            continue;
        if (vis[v]) {
            isCycle = true;
            return;
        }
    }

    for (int v : adj[u]) {
        if (v == par)
            continue;
        dfs(v, u, vis, adj);
    }

    return;
}
```

DSU (Disjoint Set Union)

Ref: https://cp-algorithms.com/data_structures/disjoint_set_union.html

- Usages and operations
  -> findset/ findpar
  -> merge/ union
  -> initialize
- Code

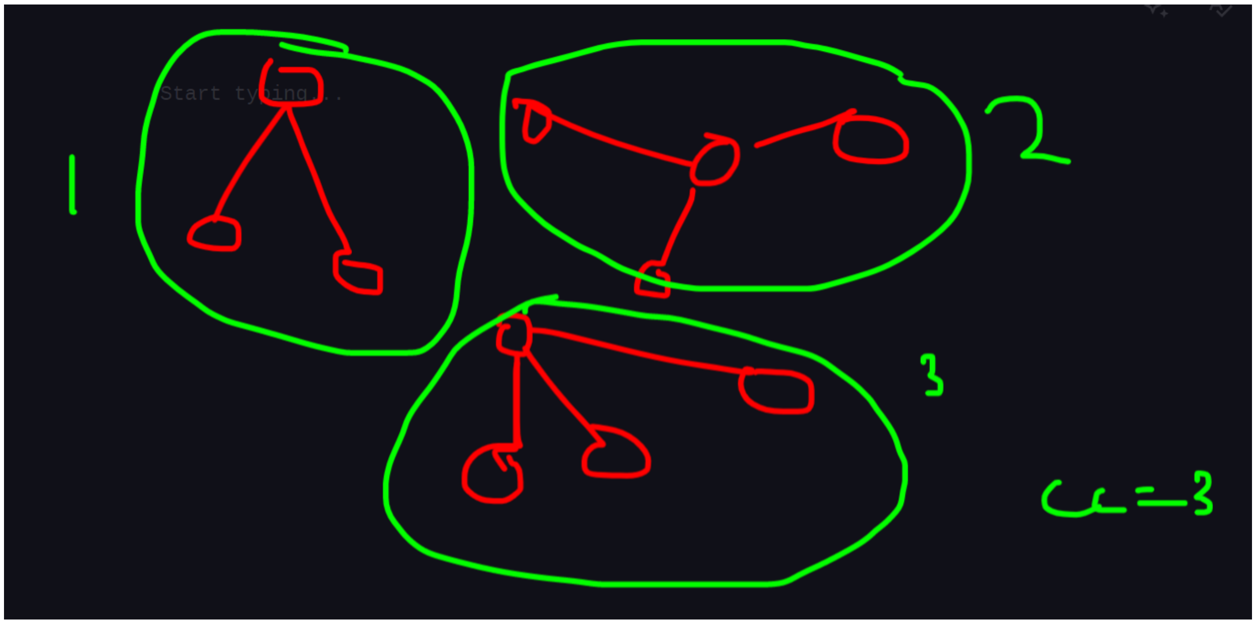
```
class dsu {
  public:
    vector<int> v;
    void init(int n) {
        v.resize(n);
        for (int i = 0; i < n; i++) {
            v[i] = i;
        }
        for (int x : v)
            cout << x << " ";
        cout << '\n';
    }
    int get(int u) {
        if (v[u] == u)
            return u;

        return v[u] = get(v[u]);
    }
    int merge(int x, int y) {
        v[get(x)] = get(y);
        for (int x : v)
            cout << x << " ";
        cout << '\n';
    }
};
```

- Optimization
  -> path comp O(logn(n))
  -> union by rank

  new time:: O(alpha(n)) where α(n) is the inverse Ackermann function, which grows very slowly. In fact, it grows so slowly, that it doesn't exceed 4 for all reasonable n (approximately n<10600).

- Problem: Components in a graph

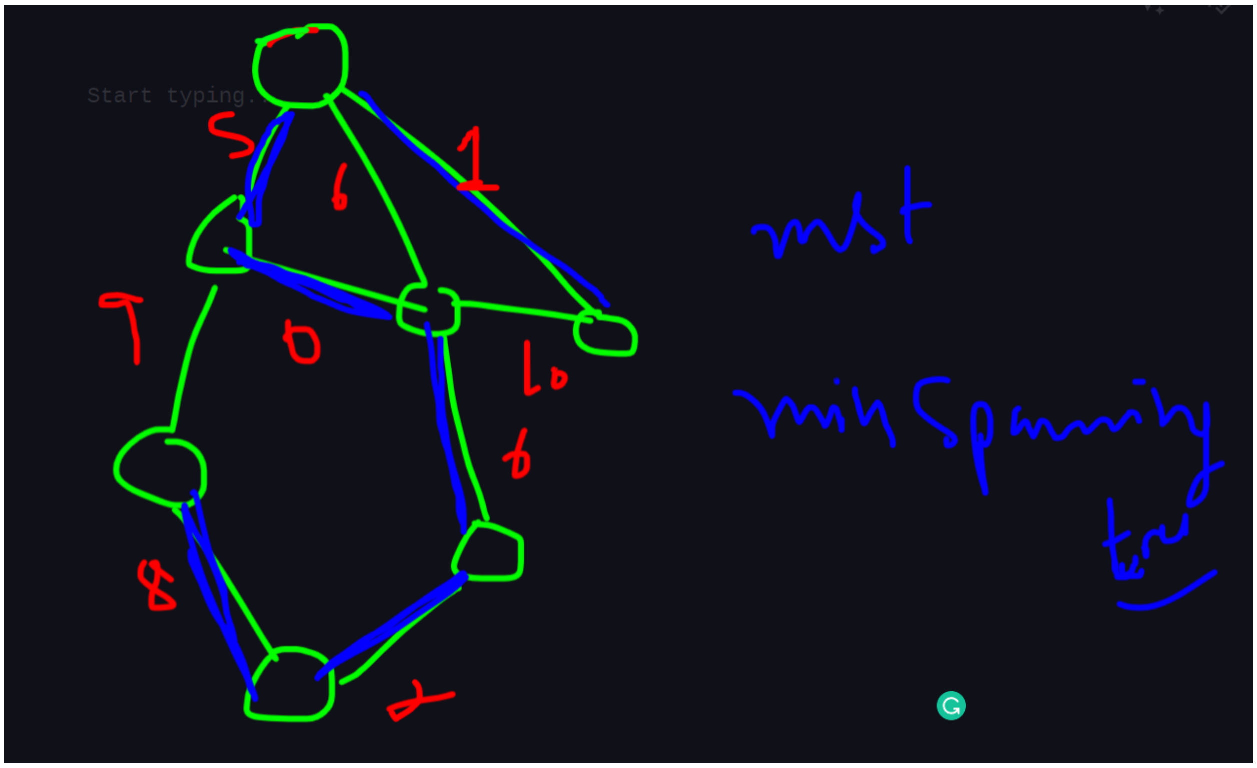


- Problem: Merging Communities
  -> comm -> dis set
  -> merger
  -> print number of comm

MST (Minimum spanning tree)

Ref: https://cp-algorithms.com/graph/mst_kruskal_with_dsu.html

- MST:
- subgraph-> all vertex & sum + edges = min



- Code