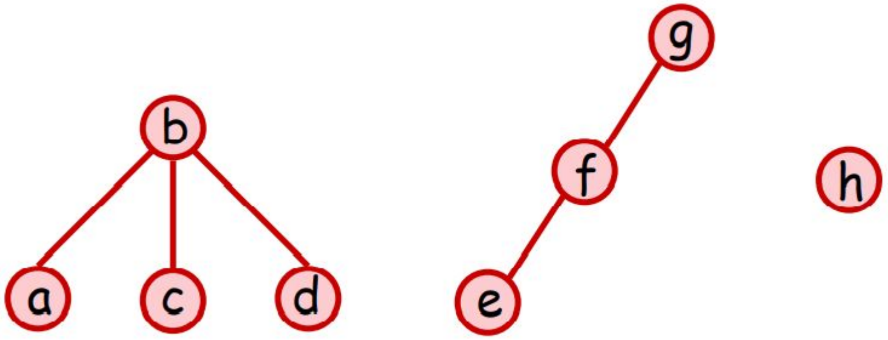
Minimum Spanning Tree

DSU

- maintain disjoint set by a forest
  - each element has its parent
  - each set is a seperate rooted tree
  - the representative of each set is the root of tree
- example: { {a, b, c, d}, {e, f, g}, {h} }



DSU- struct declaration

```
struct DisjointSet {
    int n;
    vector<int> parent, size;

    DisjointSet(int _n) : n(_n), parent(n), size(n, 1) {
        iota(parent.begin(), parent.end(), 0);
    }

    int find_root(int x);
    bool same(int x, int y);
    void uni(int x, int y);
};
```

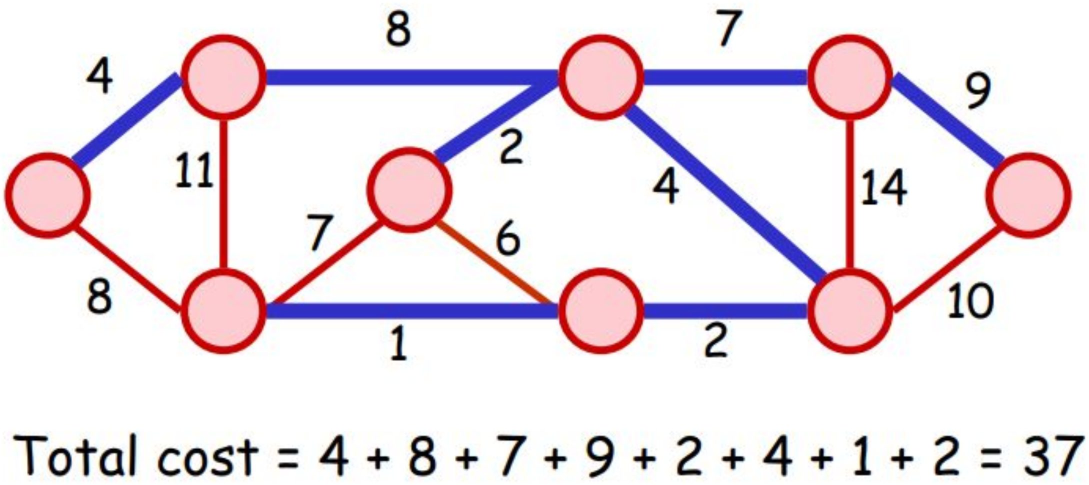
index	0	1	2	3	4
parent	0	1	2	3	4
size	1	1	1	1	1

Time Complexity

- m operations: $\Theta(m\,\alpha(n))$ (with both)
- m operations: $\Theta(m \log n)$ (with Union by size Only) (why?)
- m operations: $\Theta(m \log n)$ (with Path Compression Only)

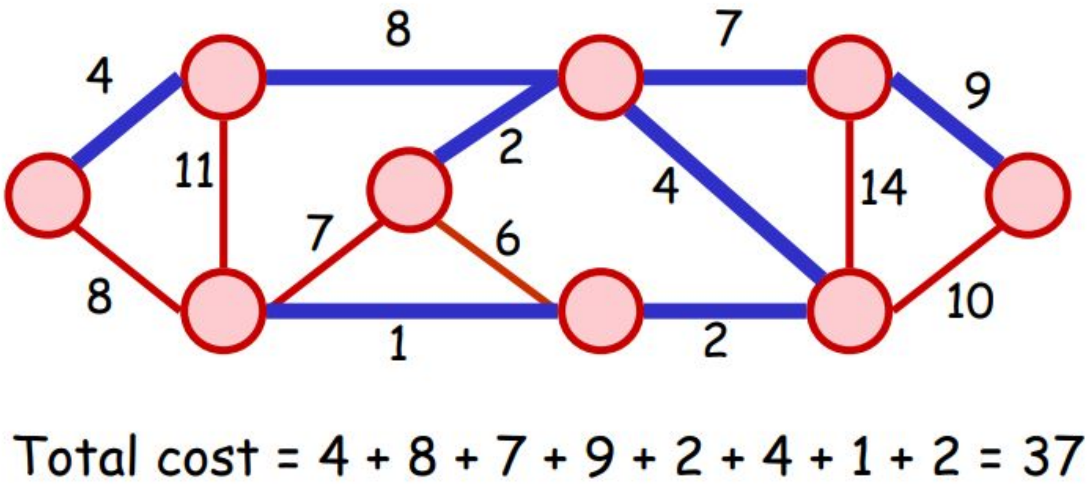
Minimum Spanning Tree

- Let G = (V,E) be an undirected, connected graph
- A spanning tree of G is a tree, using only edges in E, which connects all vertices of G



Minimum Spanning Tree

- Assume each of the edges have weight
- A MST of G is spanning tree such that the sum of edge weights is minimized



Minimum Spanning Tree

- We try to implement the Kruskal’s Algorithm
- Time Complexity: O(ElogE)
- use Disjoint Set!!!

Kruskal’s Algorithm - Preprocess

```
typedef struct {
    ll cost;
    int u,v;
}edge;
```

```
vector<edge> edges(m);
vector<int> fa(n+1);
for(int i=1;i<=n;i++) fa[i] = i;
sort(edges.begin(),edges.end(),cmp);
```

Kruskal’s Algorithm - Adding Edge

```
for(int i=0;i<m;i++) {
    ll w = edges[i].cost;
    int u = edges[i].u, v = edges[i].v;
    if(find(u,fa)!=find(v,fa)) {
        unite(u,v,fa);
        ans+=w;
    }
}
```

- add from least-costing edge
- tree -> no-cycle
- if insert edge (u, v) forms a cycle, skip it(check by DSU)

Prim’s Algorithm

- Idea: Add edges with minimum edge weight to tree one at a time. **At all times during the algorithm, the set of selected edges form a tree.**
- Step 1: Start with a tree T contains a single arbitrary vertex.
- Step 2: Among all edges, add a least cost edge (u,v) to T such that T U (u,v) is still a tree.
- Step 3: Repeat step 2 until T contains n-1 edges.

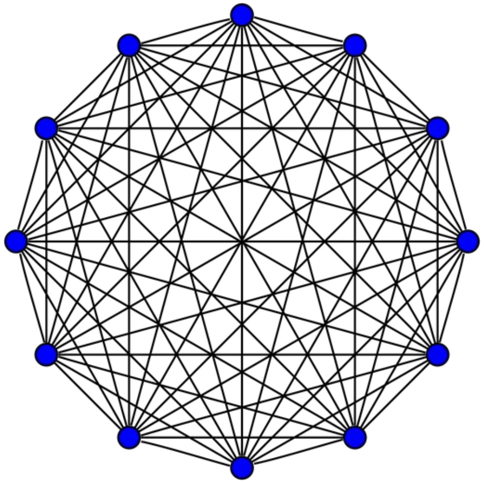
Prim’s Algorithm

```
#define INF 2e9
vector< vector<int> > mat(n, vector<int>(n, INF));
vector<int> mn_edge(n, INF);
vector<int> vis(n, 0);
```

```
int u = 0, ans = 0;
for (int rd = 0; rd < n - 1; rd++) {
    // repeat n-1 rounds
    vis[u] = 1;
    int nxt = -1;
    for (int v = 0; v < n; v++) {
        if (mn_edge[v] > mat[u][v])
            mn_edge[v] = mat[u][v];
        if (!vis[v] && (nxt == -1 || mn_edge[nxt] > mat[u][v])) {
            nxt = v;
        }
    }
    if (nxt == -1) {
        // No MST!
    }
    u = nxt;
    ans += mn_edge[nxt];
}
```

Prim’s Algorithm

- Time Complexity: O(N^2) by adjacency matrix
- When E = O(N ^ 2), Prim is faster than Kruskal !



Borůvka’s Algorithm

- Idea: Select several edges at each stage.
- Step 1: Start with a forest that has n spanning trees (each has one vertex).
- Step 2: Select one minimum cost edge for each tree. This edge has exactly one vertex in the tree.
- Step 3: Delete multiple copies of selected edges and if two edges with the same cost connecting two trees, keep only one of them.
- Step 4: Repeat until we obtain only one tree.

Borůvka’s Algorithm

- In each round, at least half of the tree will be estimated.
- At most O(log N) round
- In each round we need to iterate through all the edge.
- Time Complexity: O(E log N)
- It is faster than Kruskal’s O(E log E)

Borůvka's Algorithm

- Easy way to implement Borůvka
- use Disjoint Set!!!

```
struct Edge {
    int u, v, w;
    Edge (int _u, int _v, int _w): u(_u), v(_v), w(_w) {}
};
```

```
struct DSU {
    vector<int> dsu, sz;
    DSU(int n) {
        dsu.resize(n + 1);
        sz.resize(n + 1, 1);
        for (int i = 0; i <= n; i++) dsu[i] = i;
    }
    int get(int x) {
        return (dsu[x] == x ? x : dsu[x] = get(dsu[x]));
    }
    int oni(int a, int b) {
        a = get(a), b = get(b);
        if(a == b) return 0;
        if(sz[a] > sz[b]) swap(a, b);
        dsu[a] = b;
        sz[b] += sz[a];
        return 1;
    }
};
```

Borůvka’s Algorithm

```
long long Boruvka(const int &n, vector<Edge> &edge) {
    DSU dsu(n);
    int cc = n; // # of conected componets
    long long ans = 0;
    while (cc > 1) {
        int ok = 0;
        vector<int> min_edge_index(n + 1, -1);
        for (int i = 0; i < SZ(edge); i++) {
            int u = dsu.get(edge[i].u), v = dsu.get(edge[i].v), w = edge[i].w;
            if (u == v) continue;
            if (min_edge_index[u] == -1 || edge[min_edge_index[u]].w > w) {
                min_edge_index[u] = i;
            }
            if (min_edge_index[v] == -1 || edge[min_edge_index[v]].w > w) {
                min_edge_index[v] = i;
            }
        }
```

```
        for (int i = 1; i <= n; i++) {
            int idx = min_edge_index[i];
            if (idx != -1 && dsu.get(i) == i && dsu.oni(edge[idx].u, edge[idx].v)) {
                ok = 1;
                cc--;
                ans += edge[idx].w;
            }
        }
        if (!ok) return -1; // graph is not connected
    }
    return ans;
}
```

in class sample codes

Prim:

https://ide.usaco.guide/O3SBSKHiIY25XaPa05c

Borůvka:

https://ide.usaco.guide/O3SBzDJJd54XQ63UmPu

Note that this is a O(E log V α(V)) version

LAB 15. Breakfast Transport

Breakfast Transport

- Given **N** points on a 2D plane. The i-th point is on (x_i, y_i)
- The cost to set up a edge between the i-th and j-th points is $(x_i - x_j)^2 + (y_i - y_j)^2$
- Calculate the minimum total cost required to ensure that all **N** points are connected.

$$1 \leq N \leq 10000$$
$$1 \leq x_i, y_i \leq 10^9$$

Breakfast Transport

- It is a MST problem, but we have N^2 edge to deal with.
- Kruskal might be fast enough to pass, but Prim is cleaner.

Breakfast Transport

- We don’t need to store the graph using adjacency list/matrix

```
int cal(int x1, int y1, int x2, int y2) {
    return (x1 - x2) * (x1 - x2) + (y1-  y2) * (y1 - y2);
}
```

```
int mn, now = 0, nex = 0, ans = 0;
vis[now] = 1;
for (int rd = 0; rd < n - 1; rd++) {
    mn = -1;
    for (int i = 0; i < n; i++) {
        if (vis[i]) continue;
        dis[i] = min(dis[i], cal(x[now], y[now], x[i], y[i]));
        if (mn == -1 || mn > dis[i]) {
            mn = dis[i]; nex = i;
        }
    }
    vis[nex] = 1;
    ans += dis[nex];
    now = nex;
}
```

Breakfast Transport

- Code