1 Graphs

1.1 Traversal (DFS/BFS)

1.2 Single-source shortest paths (Dijkstra)

1.3 All-pairs shortest paths (Floyd-Warshall)

1.4 Min spanning tree (Kruskal)

1.5 Topological sort

1.6 Max flow

2 Data Structures

2.1 Union-find

A union-find structure can be used to keep track of a collection of disjoint sets, with the ability to quickly test whether two items are in the same set, and to quickly union two given sets into one. It is used in Kruskal's Minimum Spanning Tree algorithm, and can also be useful on its own.

Kattis: `10kindsofpeople`, `drivingrange`, `islandhopping`, `kastenlauf`, `lostmap`, `minspantree`, `numbersetseasy`, `treehouses`, `unionfind`, `virtualfriends`, `wheresmyinternet`,

```
public class UnionFind {
    private byte[] r; private int[] p;
    public UnionFind(int n) {
        r = new byte[n]; p = new int[n];
        for (int i = 0; i < n; i++) {
            r[i] = 0; p[i] = i;
        }
    }
    public int find(int v) {
        while (v != p[v]) {
            p[v] = p[p[v]]; v = p[v];
        }
        return v;
    }
    public boolean connected(int u, int v) {
        return find(u) == find(v);
    }
    public void union(int u, int v) {
        int ru = find(u), rv = find(v);
        if (ru != rv) {
            if      (r[ru] > r[rv]) p[rv] = ru;
            else if (r[rv] > r[ru]) p[ru] = rv;
```

```
            else {
                p[ru] = rv;
                r[rv]++;
            }
        }
    }
}
```