1 GRAPHS - Página 1 de 5

# 1 Graphs

## 1.1 Graph Traversal

### 1.1.1 Breadth First Search

```
void bfs(graph &g, int start)
3
        int n = g.size();
        vi visited(n, 1);
        queue<int> q;
        q.emplace(start);
        visited[start] = 0;
        while (not q.empty())
10
11
            int u = q.front();
12
            q.pop();
13
14
            for (int v : g[u])
15
16
                if (visited[v])
17
18
                    q.emplace(v);
19
                    visited[v] = 0;
20
21
22
23
24 }
```

### 1.1.2 Recursive Depth First Search

```
//Recursive (create visited filled with 1s)
void dfs_r(graph &g, vi &visited, int u)

{
    cout << u << '\n';
    visited[u] = 0;

    for (int v : g[u])
        if (visited[v])
        dfs_r(g, visited, v);
}</pre>
```

## 1.1.3 Iterative Depth First Search

```
//Iterative
void dfs_i(graph &g, int start)
{
    int n = g.size();
    vi visited(n, 1);
    stack<int> s;

s.emplace(start);
```

```
visited[start] = 0;
 9
        while (not s.empty())
10
11
            int u = s.top();
12
            s.pop();
13
14
            for (int v : g[u])
15
16
                if (visited[v])
17
18
                    s.emplace(v);
19
                    visited[v] = 0;
20
21
22
23
24 }
```

## 1.2 Shortest Path Algorithms

## 1.2.1 Dijsktra

All edges have non-negative values

```
1 //g has vectors of pairs of the form (w, index)
   int dijsktra(wgraph g, int start, int end)
 3
   | {
        int n = g.size();
 4
        vi cost(n, 1e9); //~INT_MAX/2
 5
 6
        priority_queue<ii, greater<ii>> q;
 7
        q.emplace(0, start);
 8
        cost[start] = 0;
 9
        while (not q.empty())
10
11
            int u = q.top().second, w = q.top().first;
12
13
            q.pop();
14
            // we skip all nodes in the q that we have discovered before at
15
                 a lower cost
            if (cost[u] < w) continue;</pre>
16
17
            for (auto v : g[u])
18
19
                if (cost[v.second] > v.first + w)
20
21
                    cost[v.second] = v.first + w;
22
                    q.emplace(cost[v.second], v.second);
23
24
25
26
27
        return cost[end];
28
29 }
```

#### 1.2.2 Bellman Ford

Edges can be negative, and it detects negative cycles

```
bool bellman_ford(wgraph &g, int start)
2
        int n = g.size();
        vector<int> dist(n, 1e9); //~INT_MAX/2
        dist[start] = 0:
        rep(i, n - 1) rep(u, n) for (ii p : g[u])
            int v = p.first, w = p.second;
            dist[v] = min(dist[v], dist[u] + w);
        }
10
11
        bool hayCicloNegativo = false;
12
        rep(u, n) for (ii p : g[u])
13
14
            int v = p.first, w = p.second;
15
            if (dist[v] > dist[u] + w)
16
                hayCicloNegativo = true;
17
        }
18
19
        return hayCicloNegativo;
20
21 }
```

## 1.2.3 Floyd Warshall

Shortest path from every node to every other node

```
1
3 Floyd Warshall implemenation, note that g is using an adjacency matrix
         and not an
   adjacency list
    graph floydWarshall (const graph g)
        int n = g.size();
        graph dist(n, vi(n, -1));
10
        rep(i, n)
11
            rep(j, n)
12
                dist[i][j] = g[i][j];
13
14
15
        rep(k, n)
            rep(i, n)
16
                rep(j, n)
17
                    if (dist[i][k] + dist[k][j] < dist[i][j] &&</pre>
18
                        dist[i][k] != INF
19
                        dist[k][j] != INF)
20
                        dist[i][j] = dist[i][k] + dist[k][j];
21
22
23
       return dist;
24 }
```

## 1.3 Minimum Spanning Tree (MST)

#### 1.3.1 Kruskal

```
1 struct edge
   {
2
        int u, v;
4
        edge(int u, int v, ll w) : u(u), v(v), w(w) {}
5
6
7
        bool operator<(const edge &o) const
8
            return w < o.w;
9
10
    };
11
12
    class Kruskal
13
14
      private:
15
       11 sum;
16
        vi p, rank;
17
18
19
     //Amount of Nodes n, and unordered vector of Edges E
20
        Kruskal(int n, vector<edge> E)
21
22
23
            sum = 0;
            p.resize(n);
24
25
            rank.assign(n, 0);
            rep(i, n) p[i] = i;
26
27
            sort(E.begin(), E.end());
            for (auto &e : E)
28
                UnionSet(e.u, e.v, e.w);
29
30
        int findSet(int i)
31
32
            return (p[i] == i) ? i : (p[i] = findSet(p[i]));
33
34
        bool isSameSet(int i, int j)
35
36
            return findSet(i) == findSet(j);
37
38
        void UnionSet(int i, int j, ll w)
39
40
            if (not isSameSet(i, j))
41
^{42}
                int x = findSet(i), y = findSet(j);
43
                if (rank[x] > rank[y])
44
                    p[y] = x;
45
46
                    p[x] = y;
47
48
                if (rank[x] == rank[y])
49
                    rank[y]++;
50
51
```

```
52 | sum += w;
53 | }
54 | }
55 | ll mst_val()
56 | {
57 | return sum;
58 | }
59 | };
```

# 1.4 Lowest Common Ancestor (LCA)

Supports multiple trees

```
1 | class LcaForest
2
        int n;
        vi parent;
        vi level;
        vi root;
        graph P;
    public:
9
        LcaForest(int n)
10
        {
11
            this \rightarrow n = n:
12
13
            parent.assign(n, -1);
            level.assign(n, -1);
14
            P.assign(n, vi(lg(n) + 1, -1));
15
            root.assign(n, -1);
16
        }
17
        void addLeaf(int index, int par)
18
19
            parent[index] = par;
20
            level[index] = level[par] + 1;
^{21}
            P[index][0] = par;
22
            root[index] = root[par];
23
            for (int j = 1; (1 << j) < n; ++j)
24
25
                if (P[index][j - 1] != -1)
26
                    P[index][j] = P[P[index][j - 1]][j - 1];
27
            }
28
        }
29
        void addRoot(int index)
30
31
            parent[index] = index;
32
            level[index] = 0;
33
            root[index] = index;
34
        }
35
36
        int lca(int u, int v)
37
            if (root[u] != root[v] || root[u] == -1)
38
                return -1;
39
            if (level[u] < level[v])</pre>
40
                swap(u, v);
41
            int dist = level[u] - level[v];
42
            while (dist != 0)
43
```

```
44
                int raise = lg(dist);
45
                u = P[u][raise];
46
                dist -= (1 << raise);
47
48
            if (u == v)
49
50
                return u;
            for (int j = lg(n); j >= 0; --j)
51
52
                if (P[u][j] != -1 && P[u][j] != P[v][j])
53
54
                    u = P[u][j];
55
                    v = P[v][j];
56
57
58
            return parent[u];
59
60
61 };
```

## 1.5 Max Flow

```
class Dinic
1
2
   | {
3
        struct edge
4
5
            int to, rev;
6
            11 f, cap;
7
       };
8
9
        vector<vector<edge>> g;
        vector<ll> dist;
10
11
        vector<int> q, work;
        int n, sink;
12
13
        bool bfs(int start, int finish)
14
15
            dist.assign(n, -1);
16
            dist[start] = 0;
17
            int head = 0, tail = 0;
18
            q[tail++] = start;
19
            while (head < tail)
20
21
                int u = q[head++];
22
                for (const edge &e : g[u])
23
^{24}
                    int v = e.to;
25
26
                    if (dist[v] == -1 and e.f < e.cap)
27
                        dist[v] = dist[u] + 1;
28
                        q[tail++] = v;
29
30
31
32
33
            return dist[finish] != -1:
```

```
}
34
35
        11 dfs(int u, 11 f)
36
37
            if (u == sink)
38
                return f;
39
            for (int &i = work[u]; i < (int)g[u].size(); ++i)</pre>
40
41
                 edge &e = g[u][i];
42
                 int v = e.to;
43
                if (e.cap <= e.f or dist[v] != dist[u] + 1)</pre>
44
45
                ll df = dfs(v, min(f, e.cap - e.f));
46
                if (df > 0)
47
48
                     e.f += df;
49
50
                     g[v][e.rev].f -= df;
                     return df;
51
                }
52
            }
53
            return 0;
54
        }
55
56
      public:
57
        Dinic(int n)
58
59
            this->n = n;
60
            g.resize(n);
61
            dist.resize(n);
62
            q.resize(n);
63
        }
64
65
        void add_edge(int u, int v, ll cap)
66
67
            edge a = \{v, (int)g[v].size(), 0, cap\};
68
            edge b = {u, (int)g[u].size(), 0, 0}; //Poner cap en vez de 0 si
69
                   la arista es bidireccional
            g[u].pb(a);
70
            g[v].pb(b);
71
        }
72
73
        11 max flow(int source, int dest)
74
75
            sink = dest;
76
            11 \text{ ans} = 0;
77
            while (bfs(source, dest))
78
79
80
                work.assign(n, 0);
                while (ll delta = dfs(source, LLONG_MAX))
81
                     ans += delta;
82
83
            return ans:
84
85
   };
86
```

## 1.6 Others

### 1.6.1 Diameter of a tree

```
1
   graph Tree;
   vi dist;
3
4
    // Finds a diameter node
5
    int bfs1()
6
7
   | {
8
        int n = Tree.size();
        queue<int> q;
9
10
        q.emplace(0);
11
12
        dist[0] = 0;
        int u;
13
        while (not q.empty())
14
15
            u = q.front();
16
            q.pop();
17
18
            for (int v : Tree[u])
19
20
                if (dist[v] == -1)
21
22
23
                    q.emplace(v);
                    dist[v] = dist[u] + 1;
24
25
26
27
28
        return u;
   }
29
30
    // Fills the distances from one diameter node and finds another diameter
    int bfs2()
32
33
   | {
        int n = Tree.size();
34
        vi visited(n, 1);
35
        queue<int> q;
36
37
        int start = bfs1();
        q.emplace(start);
38
        visited[start] = 0;
39
40
        while (not q.empty())
41
42
43
            u = q.front();
            q.pop();
44
45
            for (int v : Tree[u])
46
47
                if (visited[v])
48
49
50
                    q.emplace(v);
```

```
visited[v] = 0;
51
                   dist[v] = max(dist[v], dist[u] + 1);
52
               }
53
           }
54
       }
55
56
       return u;
57
58
    // Finds the diameter
   int bfs3()
60
61
       int n = Tree.size();
62
       vi visited(n, 1);
63
       queue<int> q;
64
       int start = bfs2();
       q.emplace(start);
66
       visited[start] = 0;
67
       int u;
       while (not q.empty())
69
70
           u = q.front();
71
           q.pop();
72
73
           for (int v : Tree[u])
74
75
               if (visited[v])
76
77
                   q.emplace(v);
78
                   visited[v] = 0;
79
                   dist[v] = max(dist[v], dist[u] + 1);
80
               }
81
           }
82
       }
83
       return dist[u];
84
85 }
```