

1 Graphs

1.1 Graph Traversal

1.1.1 Breadth First Search

```

1 #include "../headers/headers.h"
2
3 void bfs(graph &g, int start)
4 {
5     int n = g.size();
6     vi visited(n, 1);
7     queue<int> q;
8
9     q.emplace(start);
10    visited[start] = 0;
11    while (not q.empty())
12    {
13        int u = q.front();
14        q.pop();
15
16        for (int v : g[u])
17        {
18            if (visited[v])
19            {
20                q.emplace(v);
21                visited[v] = 0;
22            }
23        }
24    }
25 }
```

1.1.2 Recursive Depth First Search

```

1 #include "../headers/headers.h"
2 //Recursive (create visited filled with 1s)
3 void dfs_r(graph &g, vi &visited, int u)
4 {
5     cout << u << '\n';
6     visited[u] = 0;
7
8     for (int v : g[u])
9         if (visited[v])
10             dfs_r(g, visited, v);
11 }
```

1.1.3 Iterative Depth First Search

```

1 #include "../headers/headers.h"
2 //Iterative
3 void dfs_i(graph &g, int start)
4 {
5     int n = g.size();
6     vi visited(n, 1);
```

```

7     stack<int> s;
8
9     s.emplace(start);
10    visited[start] = 0;
11
12    while (not s.empty())
13    {
14        int u = s.top();
15        s.pop();
16
17        cout << u << '\n';
18
19        for (int v : g[u])
20            if (visited[v])
21            {
22                s.emplace(v);
23                visited[v] = 0;
24            }
25    }
26 }
```

1.2 Shortest Path Algorithms

1.2.1 Dijkstra

All edges have non-negative values

```

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

```

29     return cost[end];
30 }

```

1.2.2 Bellman Ford

Edges can be negative, and it detects negative cycles

```

1  #include "../headers/headers.h"
2  bool bellman_ford(wgraph &g, int start)
3  {
4      int n = g.size();
5      vector<int> dist(n, 1e9); //~INT_MAX/2
6      dist[start] = 0;
7      rep(i, n - 1) rep(u, n) for (ii p : g[u])
8      {
9          int v = p.first, w = p.second;
10         dist[v] = min(dist[v], dist[u] + w);
11     }
12
13     bool hayCicloNegativo = false;
14     rep(u, n) for (ii p : g[u])
15     {
16         int v = p.first, w = p.second;
17         if (dist[v] > dist[u] + w)
18             hayCicloNegativo = true;
19     }
20
21     return hayCicloNegativo;
22 }

```

1.2.3 Floyd Warshall

Shortest path from every node to every other node

```

1  #include "../headers/headers.h"
2
3  /*
4  Floyd Warshall implemenation, note that g is using an adjacency matrix
5  and not an
6  adjacency list
7  */
8  graph floydWarshall (const graph g)
9  {
10     int n = g.size();
11     graph dist(n, vi(n, -1));
12
13     rep(i, n)
14         rep(j, n)
15             dist[i][j] = g[i][j];
16
17     rep(k, n)
18         rep(i, n)
19             rep(j, n)
20                 if (dist[i][k] + dist[k][j] < dist[i][j] &&

```

```

20         dist[i][k] != INF &&
21         dist[k][j] != INF)
22             dist[i][j] = dist[i][k] + dist[k][j];
23
24     return dist;
25 }

```

1.3 Minimum Spanning Tree (MST)

1.3.1 Kruskal

```

1  #include "../headers/headers.h"
2  struct edge
3  {
4      int u, v;
5      ll w;
6      edge(int u, int v, ll w) : u(u), v(v), w(w) {}
7
8      bool operator<(const edge &o) const
9      {
10         return w < o.w;
11     }
12 };
13
14 class Kruskal
15 {
16     private:
17         ll sum;
18         vi p, rank;
19
20     public:
21         //Amount of Nodes n, and unordered vector of Edges E
22         Kruskal(int n, vector<edge> E)
23         {
24             sum = 0;
25             p.resize(n);
26             rank.assign(n, 0);
27             rep(i, n) p[i] = i;
28             sort(E.begin(), E.end());
29             for (auto &e : E)
30                 UnionSet(e.u, e.v, e.w);
31
32             int findSet(int i)
33             {
34                 return (p[i] == i) ? i : (p[i] = findSet(p[i]));
35             }
36             bool isSameSet(int i, int j)
37             {
38                 return findSet(i) == findSet(j);
39             }
40             void UnionSet(int i, int j, ll w)
41             {
42                 if (not isSameSet(i, j))
43                     {

```

```

44         int x = findSet(i), y = findSet(j);
45         if (rank[x] > rank[y])
46             p[y] = x;
47         else
48             p[x] = y;
49
50         if (rank[x] == rank[y])
51             rank[y]++;
52
53         sum += w;
54     }
55 }
56 ll mst_val()
57 {
58     return sum;
59 }
60 };

```

1.4 Lowest Common Ancestor (LCA)

Supports multiple trees

```

1  #include "../headers/headers.h"
2  class LcaForest
3  {
4      int n;
5      vi parent;
6      vi level;
7      vi root;
8      graph P;
9
10 public:
11     LcaForest(int n)
12     {
13         this->n = n;
14         parent.assign(n, -1);
15         level.assign(n, -1);
16         P.assign(n, vi(lg(n) + 1, -1));
17         root.assign(n, -1);
18     }
19     void addLeaf(int index, int par)
20     {
21         parent[index] = par;
22         level[index] = level[par] + 1;
23         P[index][0] = par;
24         root[index] = root[par];
25         for (int j = 1; (1 << j) < n; ++j)
26         {
27             if (P[index][j - 1] != -1)
28                 P[index][j] = P[P[index][j - 1]][j - 1];
29         }
30     }
31     void addRoot(int index)
32     {
33         parent[index] = index;
34         level[index] = 0;

```

```

35         root[index] = index;
36     }
37     int lca(int u, int v)
38     {
39         if (root[u] != root[v] || root[u] == -1)
40             return -1;
41         if (level[u] < level[v])
42             swap(u, v);
43         int dist = level[u] - level[v];
44         while (dist != 0)
45         {
46             int raise = lg(dist);
47             u = P[u][raise];
48             dist -= (1 << raise);
49         }
50         if (u == v)
51             return u;
52         for (int j = lg(n); j >= 0; --j)
53         {
54             if (P[u][j] != -1 && P[u][j] != P[v][j])
55             {
56                 u = P[u][j];
57                 v = P[v][j];
58             }
59         }
60         return parent[u];
61     }
62 };

```

1.5 Max Flow

```

1  #include "../headers/headers.h"
2  class Dinic
3  {
4      struct edge
5      {
6          int to, rev;
7          ll f, cap;
8      };
9
10     vector<vector<edge>> g;
11     vector<ll> dist;
12     vector<int> q, work;
13     int n, sink;
14
15     bool bfs(int start, int finish)
16     {
17         dist.assign(n, -1);
18         dist[start] = 0;
19         int head = 0, tail = 0;
20         q[tail++] = start;
21         while (head < tail)
22         {
23             int u = q[head++];

```

```

24         for (const edge &e : g[u])
25         {
26             int v = e.to;
27             if (dist[v] == -1 and e.f < e.cap)
28             {
29                 dist[v] = dist[u] + 1;
30                 q[tail++] = v;
31             }
32         }
33     }
34     return dist[finish] != -1;
35 }
36
37 ll dfs(int u, ll f)
38 {
39     if (u == sink)
40         return f;
41     for (int &i = work[u]; i < (int)g[u].size(); ++i)
42     {
43         edge &e = g[u][i];
44         int v = e.to;
45         if (e.cap <= e.f or dist[v] != dist[u] + 1)
46             continue;
47         ll df = dfs(v, min(f, e.cap - e.f));
48         if (df > 0)
49         {
50             e.f += df;
51             g[v][e.rev].f -= df;
52             return df;
53         }
54     }
55     return 0;
56 }
57
58 public:
59     Dinic(int n)
60     {
61         this->n = n;
62         g.resize(n);
63         dist.resize(n);
64         q.resize(n);
65     }
66
67     void add_edge(int u, int v, ll cap)
68     {
69         edge a = {v, (int)g[v].size(), 0, cap};
70         edge b = {u, (int)g[u].size(), 0, 0}; //Poner cap en vez de 0 si
71         la arista es bidireccional
72         g[u].pb(a);
73         g[v].pb(b);
74     }
75
76     ll max_flow(int source, int dest)
77     {
78         sink = dest;

```

```

78         ll ans = 0;
79         while (bfs(source, dest))
80         {
81             work.assign(n, 0);
82             while (ll delta = dfs(source, LLONG_MAX))
83                 ans += delta;
84         }
85         return ans;
86     }
87 };

```

1.6 Others

1.6.1 Diameter of a tree

```

1  #include "../headers/headers.h"
2
3  graph Tree;
4  vi dist;
5
6  // Finds a diameter node
7  int bfs1()
8  {
9      int n = Tree.size();
10     queue<int> q;
11
12     q.emplace(0);
13     dist[0] = 0;
14     int u;
15     while (not q.empty())
16     {
17         u = q.front();
18         q.pop();
19
20         for (int v : Tree[u])
21         {
22             if (dist[v] == -1)
23             {
24                 q.emplace(v);
25                 dist[v] = dist[u] + 1;
26             }
27         }
28     }
29     return u;
30 }
31
32 // Fills the distances from one diameter node and finds another diameter
33 // node
34 int bfs2()
35 {
36     int n = Tree.size();
37     vi visited(n, 1);
38     queue<int> q;
39     int start = bfs1();
40     q.emplace(start);

```

```
40     visited[start] = 0;
41     int u;
42     while (not q.empty())
43     {
44         u = q.front();
45         q.pop();
46
47         for (int v : Tree[u])
48         {
49             if (visited[v])
50             {
51                 q.emplace(v);
52                 visited[v] = 0;
53                 dist[v] = max(dist[v], dist[u] + 1);
54             }
55         }
56     }
57     return u;
58 }
59
60 // Finds the diameter
61 int bfs3()
62 {
63     int n = Tree.size();
64     vi visited(n, 1);
65     queue<int> q;
66     int start = bfs2();
67     q.emplace(start);
68     visited[start] = 0;
69     int u;
70     while (not q.empty())
71     {
72         u = q.front();
73         q.pop();
74
75         for (int v : Tree[u])
76         {
77             if (visited[v])
78             {
79                 q.emplace(v);
80                 visited[v] = 0;
81                 dist[v] = max(dist[v], dist[u] + 1);
82             }
83         }
84     }
85     return dist[u];
86 }
```