

Fast I/O

```
1 ios_base::sync_with_stdio(false);
2 cin.tie(NULL);
```

Union Find

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef vector<int> vi;
4
5 class UF {
6 private:
7     vi parent, rank;
8
9 public:
10     UF(int N) {
11         parent.assign(N, 0);
12         for (int i = 0; i < N; i++)
13             parent[i] = i;
14         rank.assign(N, 0);
15     }
16
17     int find(int v) {
18         if (parent[v] != v)
19             parent[v] = find(parent[v]);
20         return parent[v];
21     }
22
23     bool same_set(int a, int b) { return find(a) == find(b); }
24
25     void union_set(int a, int b) {
26         int ra = find(a), rb = find(b);
27         if (ra == rb)
28             return;
29         if (rank[ra] < rank[rb])
30             swap(ra, rb);
31         parent[rb] = ra;
32         if (rank[ra] == rank[rb])
33             rank[ra]++;
34     }
35 };
36
37 int main() {
38     UF uf(5);
39     uf.union_set(1, 2);
40     cout << "same_set(1, 2): " << uf.same_set(1, 2) << '\n';
41     cout << "same_set(1, 3): " << uf.same_set(1, 3) << '\n';
42     for (int i = 0; i < 5; i++)
43         cout << "root of " << i << ": " << uf.find(i) << '\n';
44
45     return 0;
46 }
```

MST

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using vi = vector<int>;
4 using iii = tuple<int, int, int>;
5
6 // Union Find class here ...
7
8 // Kruskal's algorithm
9 class MST {
10 public:
11     int N;
12     vector<iii> EL;
13
14     MST(const int n) : N(n), uf(n) {}
15
16     void add_edge(const int u, const int v, const int w) {
17         EL.push_back({w, u, v});
18     }
19
20     void run_mst() {
21         sort(EL.begin(), EL.end());
22
23         int edge_count = 0;
24         for (auto [w, u, v] : EL) {
25             if (edge_count == (N - 1))
26                 break;
27             if (!uf.is_same_set(u, v)) {
28                 uf.union_set(u, v);
29                 ++edge_count;
30                 mst_cost += w;
31                 mst_edges.push_back({u, v, w}); // order: u, v, w
32             }
33         }
34     }
35
36     int get_mst_cost() { return mst_cost; }
37
38     vector<iii> get_mst_edges() { return mst_edges; }
39 private:
40     UF uf;
41     vector<iii> mst_edges;
42     int mst_cost = 0;
43 };
44
45 /*
46 input: n, m; n: number of nodes, m: number of edges. Then follow m lines,
         each
47 line consisting of u, v and w indicating that there is an edge between u
         and v
48 in the graph with weight w
49 4 4
50 0 1 1
51 1 2 2
52 1 3 3
53 2 3 0
54 */
```

```

55 output: minimum spanning tree cost, and edges in lexicographic order
56 3
57 0 1
58 1 2
59 2 3
60 */
61
62 int main() {
63     MST mst(4);
64     mst.add_edge(0, 1, 1);
65     mst.add_edge(1, 2, 2);
66     mst.add_edge(1, 3, 3);
67     mst.add_edge(2, 3, 0);
68
69     mst.run_mst();
70     cout << mst.get_mst_cost() << '\n';
71
72     vector<iii> mst_edges = mst.get_mst_edges();
73
74     // order u, v
75     for (auto &[u, v, w] : mst_edges) {
76         if (u > v)
77             swap(u, v);
78     }
79
80     sort(mst_edges.begin(), mst_edges.end());
81
82     for (auto &[u, v, w] : mst_edges) {
83         cout << u << " " << v << '\n';
84     }
85
86     return 0;
87 }

```

Max Flow

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef vector<int> vi;
5 typedef pair<int, int> ii;
6 typedef pair<ll, ll> pll;
7 #define all(x) (x).begin(), (x).end()
8 #define pb push_back
9
10 // Reference:
11 // https://github.com/stevenhalim/cpbook-code/blob/master/ch8/maxflow.cpp
12 typedef tuple<int, ll, ll> edge;
13 const ll INF = 1e18;
14
15 class max_flow {
16 private:
17     int V;
18     vector<edge> EL;
19     vector<vi> AL;
20     vi d, last;
21     vector<ii> p;
22     unordered_map<ll, int> AL_EL_map;

```

```

23
24 bool BFS(int s, int t) { // find augmenting path
25     d.assign(V, -1);
26     d[s] = 0;
27     queue<int> q({s});
28     p.assign(V, {-1, -1}); // record BFS sp tree
29     while (!q.empty()) {
30         int u = q.front();
31         q.pop();
32         if (u == t)
33             break; // stop as sink t
34         reached
35         for (auto &idx : AL[u]) { // explore
36             auto &[v, cap, flow] = EL[idx]; // stored in EL[
37             idx]
38             if ((cap - flow > 0) && (d[v] == -1)) // positive
39                 residual edge
40                 d[v] = d[u] + 1, q.push(v), p[v] = {u, idx}; // 3 lines in one!
41             }
42         }
43     }
44     return d[t] != -1; // has an augmenting path
45 }
46
47 ll DFS(int u, int t, ll f = INF) { // traverse from s->t
48     if ((u == t) || (f == 0))
49         return f;
50     for (int &i = last[u]; i < (int)AL[u].size(); ++i) { // from last
51         edge
52         auto &[v, cap, flow] = EL[AL[u][i]];
53         if (d[v] != d[u] + 1)
54             continue; // not part of layer graph
55         if (ll pushed = DFS(v, t, min(f, cap - flow))) {
56             flow += pushed;
57             auto &rflow = get<2>(EL[AL[u][i] ^ 1]); // back edge
58             rflow -= pushed;
59             return pushed;
60         }
61     }
62     return 0;
63 }
64
65 public:
66     max_flow(int initialV) : V(initialV) {
67         EL.clear();
68         AL.assign(V, vi());
69     }
70
71     // if you are adding a bidirectional edge u<->v with weight w into your
72     // flow graph, set directed = false (default value is directed = true)
73     void add_edge(int u, int v, ll w, bool directed = true) {
74         if (u == v)
75             return; // safeguard: no self loop
76         EL.emplace_back(v, w, 0); // u->v, cap w, flow 0
77         AL[u].push_back(EL.size() - 1); // remember this index
78         AL_EL_map[u * V + v] = EL.size() - 1;
79         EL.emplace_back(u, directed ? 0 : w, 0); // back edge
80         AL[v].push_back(EL.size() - 1); // remember this index

```

```

76     AL_EL_map[v * V + u] = EL.size() - 1;
77 }
78
79 ll dinic(int s, int t) {
80     ll mf = 0; // mf stands for max_flow
81     while (BFS(s, t)) { // an O(V^2*E) algorithm
82         last.assign(V, 0); // important speedup
83         while (ll f = DFS(s, t)) // exhaust blocking flow
84             mf += f;
85     }
86     return mf;
87 }
88
89 ll get_flow_edge(int u, int v) { return get<2>(EL[AL_EL_map[u * V + v
    ]]); }
90 };

```

Dijkstra

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ii = pair<int, int>;
4 constexpr int INF = 1e9;
5
6 /*
7  * Graph:
8  * 4 5
9  * 0 1 5
10 * 0 2 2
11 * 0 3 10
12 * 2 3 3
13 * 1 3 1
14 * src=0
15 */
16
17 int main() {
18     // create a graph
19     int n = 4, s = 0;
20     vector<vector<pair<int, int>>> graph(4);
21     graph[0].push_back({1, 5});
22     graph[1].push_back({0, 5});
23     graph[0].push_back({2, 2});
24     graph[2].push_back({0, 2});
25     graph[0].push_back({3, 10});
26     graph[3].push_back({0, 10});
27     graph[1].push_back({3, 1});
28     graph[3].push_back({1, 1});
29     graph[2].push_back({3, 3});
30     graph[3].push_back({2, 3});
31
32     vector<int> dist(n, INF);
33     dist[s] = 0;
34     priority_queue<ii, vector<ii>, greater<ii>> pq;
35     pq.push({0, s});
36
37     // (Modified) Dijkstra's algorithm
38     while (!pq.empty()) {
39         auto [d, u] = pq.top();

```

```

40         pq.pop();
41         if (d > dist[u]) continue;
42         for (auto &[v, w]: graph[u]) {
43             if (dist[u] + w >= dist[v]) continue;
44             dist[v] = dist[u] + w; // relax
45             pq.push({dist[v], v});
46         }
47     }
48
49     for (int i = 0; i < n; ++i) {
50         cout << "Dist. from " << s << " to " << i << ": " << dist[i] << '\n';
51     }
52
53     return 0;
54 }

```

Topological Sort

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /* Graph with directed edges
5  * 0 1
6  * 1 2
7  * 1 3
8  * 2 4
9  * 3 4
10 */
11
12 vector<vector<int>> graph;
13 vector<int> indegree;
14
15 vector<int> topological_sort(int n) {
16     priority_queue<int, vector<int>, greater<int>> pq;
17     for (int i = 0; i < n; ++i) {
18         if (indegree[i] == 0) {
19             pq.push(i);
20         }
21     }
22
23     vector<int> result;
24     result.reserve(n);
25
26     while (!pq.empty()) {
27         int u = pq.top();
28         pq.pop();
29
30         result.push_back(u);
31
32         for (auto &v: graph[u]) {
33             --indegree[v];
34             if (indegree[v] == 0) pq.push(v);
35         }
36     }
37
38     return result;
39 }
40

```

```

41 int main() {
42     // create a graph
43     int n = 5;
44     graph.resize(n);
45     graph[0].push_back(1);
46     graph[1].push_back(2);
47     graph[1].push_back(3);
48     graph[2].push_back(4);
49     graph[3].push_back(4);
50
51     // Topological sort
52     // Take edges with 0 in-degree
53     // Need to implement the in-degree vector when solving actual problem
54     indegree.resize(n);
55     indegree[0] = 0;
56     indegree[1] = 1;
57     indegree[2] = 1;
58     indegree[3] = 1;
59     indegree[4] = 2;
60
61     vector<int> result = topological_sort(n);
62     for (auto x: result) cout << x << " ";
63     return 0;
64 }

```

Cycle Check

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 enum {UNVISITED = -1, EXPLORED = -2, VISITED = -3};
5 vector<vector<int>> graph;
6 vector<int> status;
7 vector<int> parents;
8
9 bool cyclecheck(int u) {
10     status[u] = EXPLORED;
11     bool hascycle = false;
12     for (auto v: graph[u]) {
13         if (status[v] == UNVISITED) {
14             parents[v] = u;
15             hascycle |= cyclecheck(v);
16         } else if (status[v] == EXPLORED) {
17             if (v == parents[u]) // bidirectional
18                 continue;
19             else // back edge (cycle)
20                 hascycle = true;
21         } else if (status[v] == VISITED) {
22             continue;
23         }
24     }
25     status[u] = VISITED;
26     return hascycle;
27 }
28
29 /* Graph with bidirectional edges
30 * 0 4
31 * 4 5

```

```

32 * 1 2
33 * 1 3
34 * 2 3
35 */
36
37 int main() {
38     // create a graph
39     int n = 6;
40     graph.resize(n);
41     graph[0].push_back(4);
42     graph[4].push_back(0);
43     graph[4].push_back(5);
44     graph[5].push_back(4);
45     graph[1].push_back(2);
46     graph[2].push_back(1);
47     graph[1].push_back(3);
48     graph[3].push_back(1);
49     graph[2].push_back(3);
50     graph[3].push_back(2);
51
52     status.assign(n, UNVISITED);
53     parents.assign(n, -1);
54     for (int i = 0; i < n; ++i) {
55         if (status[i] == UNVISITED) {
56             if (cyclecheck(i)) cout << i << ": cycle\n";
57             else cout << i << ": no cycle\n";
58         }
59     }
60
61     return 0;
62 }

```

Fenwick

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef vector<int> vi;
5 typedef vector<ll> vll;
6
7 // Reference:
8 // https://github.com/mission-peace/interview/blob/master/src/com/
9 // interview/tree/FenwickTree.java
10 class FenwickTree {
11 private:
12     vll ft;
13
14     /**
15      * To get next
16      * 1) 2's complement of get minus of index
17      * 2) AND this with index
18      * 3) Add it to index
19      */
20     int getNext(int idx) { return idx + (idx & -idx); }
21
22     /**
23      * To get parent
24      * 1) 2's complement to get minus of index

```

```

24  * 2) AND this with index
25  * 3) Subtract that from index
26  */
27  int getParent(int idx) { return idx - (idx & -idx); }
28
29 public:
30  FenwickTree(int n) { ft.assign(n + 1, 0); }
31  FenwickTree(const vll &input) { create(input); }
32
33  void update(ll val, int idx) {
34      idx++;
35      while (idx < ft.size()) {
36          ft[idx] += val;
37          idx = getNext(idx);
38      }
39  }
40
41  void create(const vll &input) {
42      int n = input.size();
43      ft.assign(n + 1, 0);
44      for (int i = 0; i < n; i++) {
45          update(input[i], i);
46      }
47  }
48
49  // get sum from [0, idx-1]
50  ll getSum(int idx) {
51      ll sum = 0;
52      while (idx > 0) {

```

```

53          sum += ft[idx];
54          idx = getParent(idx);
55      }
56      return sum;
57  }
58 };
59
60 int main() {
61     ios_base::sync_with_stdio(false);
62     cin.tie(NULL);
63
64     FenwickTree ft(5);
65     ft.update(2, 0);
66     ft.update(1, 1);
67     ft.update(-5, 2);
68     ft.update(20, 3);
69     ft.update(1, 4);
70     cout << "Fenwick Tree after update: [2,1,-5,20,1] \n";
71     cout << "sum [0, 4] = " << ft.getSum(5) << '\n';
72     cout << "sum [0, 2] = " << ft.getSum(3) << '\n';
73
74     cout << "Fenwick Tree with initial vector [1,2,-5,3,4]\n";
75     vll v = {1, 2, -5, 3, 4};
76     FenwickTree t(v);
77     cout << "sum [0, 3] = " << t.getSum(4) << '\n';
78     cout << "sum [1, 4] = " << (t.getSum(5) - t.getSum(1)) << '\n';
79
80     return 0;
81 }

```