

Overflow-Master

Contents

1	Template	2
1.1	C++ Template	2
1.2	Policy Based	2
2	Graph	2
2.1	Dijkstra	2
2.2	Bellman-Ford	2
2.3	Floyd-Warshall	3
2.4	Disjoint Sets	3
2.5	Kruskal	3
2.6	Prim	3
2.7	Tarjan	4
2.8	SCC	4
2.9	Euler-Tour	5
3	Dynamic Programming	6
3.1	Knapsack	6
3.2	LIS	6
3.3	LCS	7
3.4	Edit Distance	7
4	Search	7
4.1	Binary Search	7
4.2	Sliding Window	7
4.3	Count Bits	8
5	Queries	8
5.1	Fenwick Tree (BIT)	8
5.2	Segment Tree	8
5.3	Index Compression	9
6	Math	9
6.1	Sieve	9
6.2	LCM	9
6.3	Binomial Coefficient	9
6.4	Closest Pairs	10
6.5	Distance	10

6.6	Catalan	10
6.7	Binary Exponentiation	10

1 Template

1.1 C++ Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define L(i, j, n) for (int i = (j); i < (int)n; i++)
4 #define LI(i, j, n) for (int i = (j); i <= (int)n; i++)
5 #define R(i, j, n) for (int i = (j); i > (int)n; i--)
6 #define RI(i, j, n) for (int i = (j); i >= (int)n; i--)
7 #define SZ(x) int((x).size())
8 #define ALL(x) begin(x), end(x)
9 #define vec vector
10 #define pb push_back
11
12 using ll = long long;
13 using ld = long double;
14 using pii = pair<int, int>;
15 using pll = pair<ll, ll>;
16
17 const int MOD = (int)1e9 + 7;
18 const int oo = (int)1e9;
19
20 void solve() {}
21
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(nullptr);
25     freopen("input.txt", "r", stdin);
26     freopen("output.txt", "w", stdout);
27     int TT = 1;
28     // cin >> TT;
29     while (TT--) {
30         solve();
31     }
32     return 0;
33 }
```

1.2 Policy Based

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 template <typename Key, typename Val = null_type>
```

```

4 using indexed_set =
5     tree<Key, Val, less<Key>, rb_tree_tag,
6         tree_order_statistics_node_update>;
7 // indexed_set<char> s;
8 // char val = *s.find_by_order(0); // acceso por indice
9 // int idx = s.order_of_key('a'); // busca indice del valor
10 template <class Key, class Val = null_type>
11 using htable = gp_hash_table<Key, Val>;
12 // como unordered_map (o unordered_set si Val es vacio), pero sin metodo
13     count
```

2 Graph

2.1 Dijkstra

```

1 vec<pll> G[N];
2 vec<ll> dijk(ll s) {
3     vec<ll> dist(N, oo);
4     dist[s] = 0;
5     priority_queue<pll, vec<pll>, greater<pll>> pq;
6     pq.push({0ll, s});
7     while (!q.empty()) {
8         auto [d, u] = pq.top();
9         pq.pop();
10        if (d != dist[u]) continue;
11        for (auto [v, w] : G[u]) {
12            if (dist[v] > d + w) {
13                dist[v] = d + w;
14                pq.push({dist[v], v});
15            }
16        }
17    }
18    return dist;
19 }
```

2.2 Bellman-Ford

```

1 void bellmanFord(int n, int source, vec<vec<pii>> &g, vec<int> &d) {
2     d.assign(n, INT_MAX);
3     d[source] = 0;
4
5     for (int i = 0; i < n - 1; ++i) {
6         for (int j = 0; j < n; ++j) {
```

```

7         for (auto &[a, c] : g[j]) {
8             if (d[j] != INT_MAX && d[a] > d[j] + c) {
9                 d[a] = d[j] + c;
10            }
11        }
12    }
13 }
14 }

```

2.3 Floyd-Warshall

```

1 const int N = 10;
2 int G[N][N];
3 L(k, 0, n)
4     L(i, 0, n)
5         L(j, 0, n)
6             G[i][j] = min(G[i][j], G[i][k] + G[k][j]);

```

2.4 Disjoint Sets

```

1 struct UFDS {
2     vi p, size;
3     int numSets, n;
4     UFDS(int n) : p(n), size(n, 1), n(n) {
5         for (int i = 0; i < n; i++) p[i] = i;
6         numSets = n;
7     }
8     int find(int i) { return (p[i] == i) ? i : (p[i] = find(p[i])); }
9     void join(int i, int j) {
10         int a = find(i), b = find(j);
11         if (a != b) {
12             if (size(b) && size(a)) swap(a, b);
13             p[b] = a;
14             size[a] += size[b];
15             numSets--;
16         }
17     }
18 };

```

2.5 Kruskal

```

1 struct Edge {
2     int w, u, v;
3     Edge(int wx, int ux, int vx) { w = wx, u = ux, v = vx; }

```

```

4     bool operator<(const Edge &other) const { return w < other.w; }
5 };
6
7 int main() {
8     int V, E;
9     cin >> V >> E;
10    vector<Edge> EL(E);
11    for (int i = 0; i < E; i++) {
12        int u, v, w;
13        cin >> u >> v >> w;
14        EL[i] = Edge(w, u, v);
15    }
16    sort(EL.begin(), EL.end());
17    int mst_cost = 0, num_taken = 0;
18    UFDS UF(V);
19    for (auto &[w, u, v] : EL) {
20        if (UF.isSameSet(u, v)) continue;
21        mst_cost += w;
22        UF.unionSet(u, v);
23        ++num_taken;
24        if (num_taken == V - 1) break;
25    }
26
27    return 0;
28 }

```

2.6 Prim

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 typedef pair<int, int> pii;
5 vector<vector<pii>> AL;
6 vector<int> taken;
7 priority_queue<pii, vector<pii>, greater<pii>> pq;
8
9 void process(int u) {
10     taken[u] = 1;
11     for (auto &[v, w] : AL[u]) {
12         if (!taken[v]) {
13             pq.emplace(w, v);
14         }
15     }

```

```

16 }
17
18 int main() {
19     int V, E;
20     cin >> V >> E;
21     AL.assign(V, vector<pii>());
22     for (int i = 0; i < E; i++) {
23         int u, v, w;
24         cin >> u >> v >> w;
25         AL[u].emplace_back(v, w);
26         AL[v].emplace_back(u, w);
27     }
28     taken.assign(V, 0);
29     process(0);
30     int mst_cost = 0, num_taken = 0;
31     while (!pq.empty()) {
32         auto [w, u] = pq.top();
33         pq.pop();
34         if (taken[u]) continue;
35         mst_cost += w;
36         process(u);
37         ++num_taken;
38         if (num_taken == V - 1) break;
39     }
40     cout << "MST cost: " << mst_cost << endl;
41     return 0;
42 }

```

2.7 Tarjan

```

1 vector<int> G[N];
2 vector<int> dfs_low(N, -1), dfs_num(N, -1),
3   ap(N, 0); // ap for Articulation Points
4 int dfs_count = 0;
5 int root = -1; // For AP
6 void dfs(int u, int p = -1) {
7     dfs_low[u] = dfs_num[u] = dfs_count++;
8     int child = 0;
9     for (int v : G[u]) {
10         if (v == p) continue;
11         if (dfs_num[v] == -1) {
12             child++;
13             dfs(v, u);

```

```

14         dfs_low[u] = min(dfs_low[u], dfs_low[v]);
15         if (dfs_low[v] > dfs_num[u]) {
16             // Bridge from u -> v
17             cout << "Bridge " << u << " -> " << v << "\n";
18         }
19         if (dfs_low[v] >= dfs_num[u]) {
20             // u is AP
21             ap[u] = 1;
22         }
23     } else
24         dfs_low[u] = min(dfs_low[u], dfs_num[v]);
25 }
26 if (u == root) {
27     ap[u] = child > 1;
28 }
29 }

```

2.8 SCC

```

1 struct SCC {
2     int n;
3     vec<vec<int>> G, G2;
4     vec<int> order, sccId, vi;
5     vec<vec<int>> components;
6     int sccCount;
7
8     SCC(int n) : n(n) {
9         G.assign(n, vec<int>());
10        G2.assign(n, vec<int>());
11        sccId.assign(n, -1);
12        sccCount = 0;
13    }
14
15    void addEdge(int u, int v) {
16        G[u].pb(v);
17        G2[v].pb(u);
18    }
19
20    void dfs1(int u) {
21        vi[u] = 1;
22        for (int v : G[u]) {
23            if (!vi[v]) dfs1(v);
24        }

```

```

25     order.pb(u);
26 }
27
28 void dfs2(int u, int id) {
29     vi[u] = 1;
30     sccId[u] = id;
31     components[id].pb(u);
32     for (int v : G2[u]) {
33         if (!vi[v]) dfs2(v, id);
34     }
35 }
36
37 void findSCC() {
38     vi.assign(n, 0);
39     order.clear();
40     L(i, 0, n) {
41         if (!vi[i]) dfs1(i);
42     }
43
44     vi.assign(n, 0);
45     sccCount = 0;
46     components.clear();
47
48     reverse(ALL(order));
49     for (int u : order) {
50         if (!vi[u]) {
51             components.pb(vec<int>());
52             dfs2(u, sccCount++);
53         }
54     }
55 }
56
57 vec<vec<int>> getCondensedGraph() {
58     vec<vec<int>> sccGraph(sccCount);
59     set<pii> edges;
60
61     L(u, 0, n) {
62         for (int v : G[u]) {
63             int fromScc = sccId[u], toScc = sccId[v];
64             if (fromScc != toScc &&
65                 edges.find({fromScc, toScc}) == edges.end()) {
66                 sccGraph[fromScc].pb(toScc);
67                 edges.insert({fromScc, toScc});

```

```

68     }
69     }
70 }
71     return sccGraph;
72 }
73
74 int getSCCId(int u) { return sccId[u]; }
75 vec<int> getSCC(int i) { return components[i]; }
76 int getCount() { return sccCount; }
77 };

```

2.9 Euler-Tour

```

1 struct edge {
2     int y;
3     list<edge>::iterator rev; // NO DIRIGIDOS: iterador para arista
4     reversa
5     edge(int y) : y(y) {}
6 };
7
8 list<edge> g[N];
9
10 void add_edge(int a, int b) {
11     g[a].push_front(edge(b));
12     auto ia = g[a].begin(); // NO DIRIGIDOS
13     g[b].push_front(edge(a)); // NO DIRIGIDOS
14     auto ib = g[b].begin(); // NO DIRIGIDOS
15     ia->rev = ib; // NO DIRIGIDOS
16     ib->rev = ia; // NO DIRIGIDOS
17 }
18
19 vec<int> p;
20
21 void go(int x) {
22     while (SZ(g[x])) {
23         int y = g[x].front().y;
24         g[y].erase(g[x].front().rev); // NO DIRIGIDOS: eliminar
25         g[x].pop_front();
26         go(y);
27     }
28     p.pb(x);
29 }

```

```

30 vec<int> get_path(int x) {
31     p.clear();
32     go(x);
33     reverse(ALL(p));
34     return p;
35 }
36
37 void solve() {
38     int n, m;
39     cin >> n >> m;
40
41     // vec<int> inDeg(n, 0), outDeg(n, 0); // DIRIGIDOS
42     vec<int> deg(n, 0); // NO DIRIGIDOS
43
44     L(i, 0, m) {
45         int a, b;
46         cin >> a >> b;
47         a--;
48         b--;
49         add_edge(a, b);
50         // inDeg[b]++; // DIRIGIDOS
51         // outDeg[a]++; // DIRIGIDOS
52         deg[a]++; // NO DIRIGIDOS
53         deg[b]++; // NO DIRIGIDOS
54     }
55
56     // DIRIGIDOS (camino euleriano):
57     // Nodo 0: outDeg[0] = inDeg[0] + 1 (nodo inicial)
58     // Nodo n-1: inDeg[n-1] = outDeg[n-1] + 1 (nodo final)
59     // Resto: inDeg[i] = outDeg[i]
60     // L(i, 1, n - 1) {
61     //     if (inDeg[i] != outDeg[i]) {
62     //         cout << "IMPOSSIBLE\n";
63     //         return;
64     //     }
65     // }
66     // if (outDeg[0] != inDeg[0] + 1 || inDeg[n - 1] != outDeg[n - 1] +
67     //     1) {
68     //     cout << "IMPOSSIBLE\n";
69     //     return;
70     // }
71
72     // NO DIRIGIDOS: verificar que todos los grados sean pares

```

```

72     L(i, 0, n) {
73         if (deg[i] % 2) {
74             cout << "IMPOSSIBLE\n";
75             return;
76         }
77     }
78
79     vec<int> path = get_path(0);
80
81     if (SZ(path) != m + 1) {
82         cout << "IMPOSSIBLE\n";
83     } else {
84         for (auto x : path) {
85             cout << x + 1 << " ";
86         }
87         cout << "\n";
88     }
89 }

```

3 Dynamic Programming

3.1 Knapsack

```

1 void solve() {
2     vec<int> prices(n);
3     vec<int> pages(n);
4     vec<vec<int>> dp(n + 1, vec<int>(x + 1, 0));
5     for (int i = 0; i < n; i++) {
6         for (int j = 0; j <= x; j++) {
7             if (prices[i] <= j) {
8                 dp[i + 1][j] = max(dp[i][j], pages[i] + dp[i][j - prices
9                     [i]]);
10            } else {
11                dp[i + 1][j] = dp[i][j];
12            }
13        }
14    }

```

3.2 LIS

```

1 int lis(vector<int>& arr) {
2     int n = arr.size();

```

```

3   vector<int> lis(n, 1);
4   for (int i = 1; i < n; i++) {
5       for (int prev = 0; prev < i; prev++) {
6           if (arr[i] > arr[prev] && lis[i] < lis[prev] + 1) {
7               lis[i] = lis[prev] + 1;
8           }
9       }
10  }
11  return *max_element(lis.begin(), lis.end());
12 }

```

3.3 LCS

```

1  int lcs(string &S1, string &S2) {
2      vec<vec<int>> dp(m + 1, vec<int>(n + 1, 0));
3      for (int i = 1; i <= m; ++i) {
4          for (int j = 1; j <= n; ++j) {
5              if (S1[i - 1] == S2[j - 1])
6                  dp[i][j] = dp[i - 1][j - 1] + 1;
7              else
8                  dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
9          }
10     }
11     return dp[m][n];
12 }

```

3.4 Edit Distance

```

1  int editDistance(string& s1, string& s2) {
2      int n = s1.length(), m = s2.length();
3      vector<vector<int>> dp(n + 1, vector<int>(m + 1));
4
5      // Base cases
6      for (int i = 0; i <= n; i++) dp[i][0] = i;
7      for (int j = 0; j <= m; j++) dp[0][j] = j;
8
9      for (int i = 1; i <= n; i++) {
10         for (int j = 1; j <= m; j++) {
11             if (s1[i - 1] == s2[j - 1]) {
12                 dp[i][j] = dp[i - 1][j - 1];
13             } else {
14                 dp[i][j] = 1 + min({dp[i - 1][j],           // deletion
15                                     dp[i][j - 1],           // insertion
16                                     dp[i - 1][j - 1]});      // replacement
17             }
18         }
19     }
20 }

```

```

17     }
18 }
19 }
20 return dp[n][m];
21 }

```

4 Search

4.1 Binary Search

```

1  int binSearch(int arr[], int low, int high, int x) {
2      while (low <= high) {
3          int mid = low + (high - low) / 2;
4          if (arr[mid] == x) return mid;
5          if (arr[mid] < x)
6              low = mid + 1;
7          else
8              high = mid - 1;
9      }
10     return -1;
11 }

```

4.2 Sliding Window

```

1  int main() {
2      int cant = 0, start = 0, end = 0, sum = 0;
3      while (end < n) {
4          while (end < n && sum < x) {
5              sum += arr[end];
6              end++;
7          }
8          while (start <= end && sum > x) {
9              sum -= arr[start];
10             start++;
11         }
12         if (sum == x) {
13             cant++;
14             sum -= arr[start];
15             start++;
16         }
17     }
18     cout << cant;
19     return 0;

```

20 }
 21

4.3 Count Bits

```
1 void update_bits_and_sum(long mask, vec<int> &bits_used, long long &sum)
  {
2   for (long j = mask; j > 0; j &= j - 1) {
3       int bit = __builtin_ctzll(j); // lowest bit ON (0-index)
4       if (bits_used[bit] == 0) {
5           sum += (1LL << bit);
6       }
7       bits_used[bit]++;
8   }
9 }
```

5 Queries

5.1 Fenwick Tree (BIT)

```
1 #include <bits/stdc++.h>
2 #define ll long long
3 #define MOD 1000000007
4 using namespace std;
5
6 const int MAXN = 200000;
7 ll BIT[MAXN + 1]; // Array para el BIT
8 ll arr[MAXN + 1]; // Array original
9
10 void update(int idx, ll delta, int n) {
11     while (idx <= n) {
12         BIT[idx] += delta;
13         idx += idx & -idx;
14     }
15 }
16
17 ll query(int idx) {
18     ll sum = 0;
19     while (idx > 0) {
20         sum += BIT[idx];
21         idx -= idx & -idx;
22     }
23     return sum;
24 }
```

25

```
26 ll rangeQuery(int L, int R) { return query(R) - query(L - 1); }
27
28 int main() {
29     int n, q;
30     cin >> n >> q;
31     for (int i = 1; i <= n; i++) {
32         cin >> arr[i];
33         update(i, arr[i], n); // init
34     }
35
36     while (q--) {
37         ios::sync_with_stdio(0);
38         cin.tie(0);
39         ll type, a, b;
40         cin >> type >> a >> b;
41         if (type == 1) {
42             ll delta = b - arr[a];
43             arr[a] = b;
44             update(a, delta, n);
45         } else {
46             cout << rangeQuery(a, b) << "\n";
47         }
48     }
49
50     return 0;
51 }
```

5.2 Segment Tree

```
1 struct SegTree {
2     int n;
3     vec<int> A, st, lazy;
4     int l(int p) { return p << 1; }
5     int r(int p) { return (p << 1) + 1; }
6     int conquer(int a, int b) {
7         if (a == -1) return b;
8         if (b == -1) return a;
9         return a + b;
10    }
11    void build(int p, int L, int R) {
12        if (L == R)
13            st[p] = A[L];
```



```

14     else {
15         int m = L + (R - L) / 2;
16         build(l(p), L, m);
17         build(r(p), m + 1, R);
18         st[p] = conquer(st[l(p)], st[r(p)]);
19     }
20 }
21 void propagate(int p, int L, int R) {
22     if (lazy[p] != -1) {
23         st[p] = lazy[p];
24         if (L != R) {
25             lazy[l(p)] = lazy[r(p)] = lazy[p];
26         }
27         lazy[p] = -1;
28     }
29 }
30 int query(int p, int L, int R, int i, int j) {
31     if (i > j || L > j || R < i) return 0;
32     propagate(p, L, R);
33     if (L >= i && R <= j) return st[p];
34     int m = L + (R - L) / 2;
35     return conquer(query(l(p), L, m, i, j), query(r(p), m + 1, R, i,
36         j));
37 }
38 void update(int p, int L, int R, int i, int j, int v) {
39     if (i > j || L > j || R < i) return;
40     propagate(p, L, R);
41     if (L >= i && R <= j) {
42         lazy[p] = v;
43         propagate(p, L, R);
44     } else {
45         int m = L + (R - L) / 2;
46         update(l(p), L, m, i, j, v);
47         update(r(p), m + 1, R, i, j, v);
48         st[p] = conquer(st[l(p)], st[r(p)]);
49     }
50 }
51 SegTree(int sz) : n(sz), st(4 * n), lazy(4 * n, -1) {}
52 SegTree(const vec<int> &init) : SegTree((int)init.size()) {
53     A = init;
54     build(1, 0, n - 1);
55 }
56 void update(int i, int j, int val) { update(1, 0, n - 1, i, j, val);

```

```

    }
56     int query(int i, int j) { return query(1, 0, n - 1, i, j); }
57 };

```

5.3 Index Compression

```

1 template <class T>
2 struct Index { // If only 1 use Don't need to copy T type
3     vec<T> d;
4     int sz;
5     Index(vec<T> &a) : d(ALL(a)) {
6         sort(ALL(d)); // Sort
7         d.erase(unique(ALL(d)), end(d)); // Erase continuous duplicates
8         sz = SZ(d);
9     }
10    int of(T e) { return lower_bound(ALL(d), e) - begin(d); } // get
11        index
12    T at(int i) { return d[i]; } // get value of index

```

6 Math

6.1 Sieve

```

1 void solve(int n) {
2     for (int x = 2; x <= n; x++) {
3         if (sieve[x]) continue;
4         for (int u = 2 * x; u <= n; u += x) {
5             sieve[u] = 1;
6         }
7     }
8 }

```

6.2 LCM

```

1 int lcm(int a, int b) { return (a * b) / __gcd(a, b); }

```

6.3 Binomial Coefficient

```

1 using ll = long long;
2 const int MAXN = 1e6 + 5;
3 const ll MOD = 1e9 + 7;
4 ll factorial[MAXN];
5

```

```

6 void build_factorials() {
7     factorial[0] = 1;
8     for (int i = 1; i < MAXN; i++) {
9         factorial[i] = factorial[i - 1] * i % MOD;
10    }
11 }
12 ll binomial_coefficient(int n, int k) {
13     if (k < 0 || k > n) return 0;
14     ll denom = factorial[k] * factorial[n - k] % MOD;
15     return factorial[n] * exp(denom, MOD - 2) % MOD;
16 }

```

6.4 Closest Pairs

```

1 using ld = long double;
2 vec<pair<ld, ld>> closestPair(vector<pair<ld, ld>> coord, int n) {
3     sort(ALL(coord));
4     set<pair<ld, ld>> s;
5     ld squaredDistance = LLONG_MAX;
6     vec<pair<ld, ld>> ans;
7     int j = 0;
8     for (int i = 0; i < n; ++i) {
9         ld D = ceil(sqrt(squaredDistance));
10        while (coord[i].first - coord[j].first >= D) {
11            s.erase({coord[j].second, coord[j].first});
12            j += 1;
13        }
14
15        auto start = s.lower_bound({coord[i].second - D, coord[i].first});
16        auto end = s.upper_bound({coord[i].second + D, coord[i].first});
17
18        for (auto it = start; it != end; ++it) {
19            ld dx = coord[i].first - it->second;
20            ld dy = coord[i].second - it->first;
21            ld preDist = min(squaredDistance, dx * dx + dy * dy);
22            if (preDist < squaredDistance) {
23                pair<ld, ld> one = {it->second, it->first};
24                pair<ld, ld> two = {coord[i].first, coord[i].second};
25                ans = {one, two};
26                squaredDistance = preDist;
27            }
28        }
29    }
30 }

```

```

29
30 // Insert the point as {y-coordinate, x-coordinate}
31 s.insert({coord[i].second, coord[i].first});
32 }
33 return ans;
34 }

```

6.5 Distance

```

1 double dist(double x1, double y1, double x2, double y2) {
2     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
3 }

```

6.6 Catalan

```

1 const int MOD = ....const int MAX = ....int catalan[MAX];
2 void init() {
3     catalan[0] = catalan[1] = 1;
4     for (int i = 2; i <= n; i++) {
5         catalan[i] = 0;
6         for (int j = 0; j < i; j++) {
7             catalan[i] += (catalan[j] * catalan[i - j - 1]) % MOD;
8             if (catalan[i] >= MOD) {
9                 catalan[i] -= MOD;
10            }
11        }
12    }
13 }

```

6.7 Binary Exponentiation

```

1 ll power(ll a, ll b, ll m) {
2     a %= m;
3     ll res = 1;
4     while (b > 0) {
5         if (b & 1) res = res * a % m;
6         a = a * a % m;
7         b >>= 1;
8     }
9     return res;
10 }

```