

ICPC Library

Keigo Habara

2019 年 12 月 4 日

1 Template

```
1 #ifndef TEMPLATE
2 #define TEMPLATE
3
4 //include <bits/stdc++.h>
5 using namespace std;
6
7 #define SZ(x) (int)(x.size())
8 #define REP(i, n) for(int i = 0; i < n; i++)
9 #define FOR(i, a, b) for(int i = a; i < b; i++)
10
11 using ll = long long;
12 using ld = long double;
13 using P = pair<int, int>;
14 using vi = vector<int>;
15 using vvi = vector<vi>;
16 using vll = vector<ll>;
17 using vvll = vector<vll>;
18 const double eps = 1e-8;
19 const int MOD = 1000000007;
20
21 //int main() {
22 // cin.tie(0);
23 // ios::sync_with_stdio(false);
24 // cout << fixed << setprecision(10);
25 //
26 //}
27
28 #endif
```

2 Graph

2.1 Dijkstra

```
1 #include "template.cpp"
2
3 template<typename T>
4 vector<T> dijkstra(const Graph<T> &g, int s) {
5     const auto INF = numeric_limits<T>::max();
6     vector<T> d(g.size(), INF);
7
8     using Pi = pair<T, int>;
9     priority_queue<Pi, vector<Pi>, greater<Pi>> que;
10    d[s] = 0;
11    que.emplace(d[s], s);
12    while (!que.empty()) {
13        T cost;
14        int v;
15        tie(cost, v) = que.top();
16        que.pop();
17        if (d[v] < cost) continue;
18        for (auto &e : g[v]) {
19            auto nxt = cost + e.cost;
20            if (d[e.to] > nxt) {
21                d[e.to] = nxt;
22                que.emplace(nxt, e.to);
23            }
24        }
25    }
26}
```

```
23     }
24 }
25 }
26 return d;
27 }
```

2.2 Dinic

```
1 #include "template.cpp"
2
3 template<typename T>
4 struct Dinic {
5     const T INF;
6
7     struct edge {
8         int to;
9         T cap;
10        int rev;
11        bool isrev;
12    };
13
14    vector<vector<edge>> g;
15    vector<int> level, iter;
16
17    Dinic(int V) : INF(numeric_limits<T>::max()), g(V) {}
18
19    void add_edge(int from, int to, T cap) {
20        g[from].push_back({to, cap, (int)g[to].size(), false});
21        g[to].push_back({from, 0, (int)g[from].size()-1, true});
22    }
23
24    bool bfs(int s, int t) {
25        level.assign(g.size(), -1);
26        queue<int> que;
27        level[s] = 0;
28        que.push(s);
29        while (!que.empty()) {
30            int v = que.front();
31            que.pop();
32            for (auto &e : g[v]) {
33                if (e.cap > 0 && level[e.to] == -1) {
34                    level[e.to] = level[v] + 1;
35                    que.push(e.to);
36                }
37            }
38        }
39        return level[t] != -1;
40    }
41
42    T dfs(int v, const int t, T flow) {
43        if (v == t) return flow;
44        for (int &i = iter[v]; i < g[v].size(); i++) {
45            edge &e = g[v][i];
46            if (e.cap > 0 && level[v] < level[e.to]) {
47                T d = dfs(e.to, t, min(flow, e.cap));
48                if (d > 0) {
49                    e.cap -= d;
50                    g[e.to][e.rev].cap += d;
51                    return d;
52                }
53            }
54        }
55        return 0;
56    }
57}
```

```

53     }
54 }
55 return 0;
56 }
57
58 T max_flow(int s, int t) {
59     T flow = 0;
60     while (bfs(s, t)) {
61         iter.assign(g.size(), 0);
62         T f = 0;
63         while((f = dfs(s, t, INF)) > 0) flow += f;
64     }
65     return flow;
66 }
67 };

```

2.3 Hopcroft Karp

```

1 #include "template.cpp"
2
3 struct HopcroftKarp {
4     vector<vector<int>> g;
5     vector<int> d, mch;
6     vector<bool> used, vv;
7
8     HopcroftKarp(int n, int m) : g(n), mch(m, -1), used(n) {}
9
10    void add_edge(int u, int v) {
11        g[u].push_back(v);
12    }
13
14    void bfs() {
15        d.assign(g.size(), -1);
16        queue<int> que;
17        for (int i = 0; i < g.size(); i++) {
18            if (!used[i]) {
19                que.emplace(i);
20                d[i] = 0;
21            }
22        }
23
24        while (!que.empty()) {
25            int a = que.front();
26            que.pop();
27            for (auto &b : g[a]) {
28                int c = mch[b];
29                if (c >= 0 && d[c] == -1) {
30                    d[c] = d[a] + 1;
31                    que.emplace(c);
32                }
33            }
34        }
35    }
36
37    bool dfs(int a) {
38        vv[a] = true;
39        for (auto &b : g[a]) {
40            int c = mch[b];
41            if (c < 0 || (!vv[c] && d[c] == d[a] + 1 && dfs(c))) {
42                mch[b] = a;

```

```

43                used[a] = true;
44                return (true);
45            }
46        }
47        return (false);
48    }
49
50    int bipartite_matching() {
51        int ret = 0;
52        while (true) {
53            bfs();
54            vv.assign(g.size(), false);
55            int flow = 0;
56            for (int i = 0; i < g.size(); i++) {
57                if (!used[i] && dfs(i)) ++flow;
58            }
59            if (flow == 0) return ret;
60            ret += flow;
61        }
62    }
63 };

```

2.4 Kruskal

```

1 #include "template.cpp"
2
3 #include "../structure/union_find.cpp"
4
5 template<typename T>
6 T kruskal(vector<edge<T>> &es, int V) {
7
8     UnionFind uf(V);
9     T ret = 0;
10
11     // sort destructively
12     sort(es.begin(), es.end(), [](edge<T> &a, edge<T> &b){
13         return a.cost < b.cost;
14     });
15     for (auto &e : es) {
16         if (!uf.issame(e.src, e.to)) {
17             ret += e.cost;
18             uf.merge(e.src, e.to);
19         }
20     }
21
22     // // sort only the order to check
23     // vector<int> ord(es.size());
24     // iota(ord.begin(), ord.end(), 0);
25     // sort(ord.begin(), ord.end(), [&](int i, int j){
26     //     return es[i].cost < es[j].cost;
27     // });
28     // for (auto i : ord) {
29     //     auto &e = es[i];
30     //     if (!uf.issame(e.src, e.to)) {
31     //         ret += e.cost;
32     //         uf.merge(e.src, e.to);
33     //     }
34     // }
35
36     return ret;

```

2.5 LCA

```

1 #include "../template.cpp"
2
3 struct LCA {
4     int n, log2_n;
5     vector<int> depth;
6     vector<vector<int>> par;
7
8     void dfs(const vector<vector<int>>& G, int v, int p, int d) {
9         depth[v] = d;
10        par[0][v] = p;
11        for (auto to : G[v]) {
12            if (to != p) dfs(G, to, v, d+1);
13        }
14    }
15
16    LCA(const vector<vector<int>>& G, int root=0) :
17        n(G.size()), log2_n(log2(n)), depth(n),
18        par(log2_n+1, vector<int>(n,-1)) {
19
20        dfs(G, root, -1, 0);
21
22        for (int k = 0; k < log2_n; ++k) {
23            for (int v = 0; v < n; ++v) {
24                if (par[k][v] != -1) {
25                    par[k+1][v] = par[k][par[k][v]];
26                }
27            }
28        }
29    }
30
31    int query(int u, int v) {
32        if (depth[u] > depth[v]) swap(u, v);
33
34        // align the depth of u and v
35        for (int k = 0; k <= log2_n; ++k) {
36            if ((depth[v] - depth[u]) >= k & 1) {
37                v = par[k][v];
38            }
39        }
40        if (u == v) return u;
41
42        // go back until u and v's parents do not match
43        for (int k = log2_n; k >= 0; --k) {
44            if (par[k][u] != par[k][v]) {
45                u = par[k][u];
46                v = par[k][v];
47            }
48        }
49        return par[0][u];
50    }
51 };

```

2.6 Lowlink

```

1 #include "../template.cpp"
2
3 template<typename T>
4 struct LowLink {
5     const int inf = 1000000000;
6     int sz;
7     std::vector<int> pre, low;
8     std::vector<bool> sel;
9     std::vector<std::pair<int, int>> bridge;
10    std::vector<int> articulation;
11
12    LowLink(const Graph<T> &g) {
13        sz = g.size();
14        pre.resize(sz, inf);
15        low.resize(sz, inf);
16        sel.resize(sz, false);
17        int cnt = 0;
18        dfs(g, 0, -1, cnt);
19    }
20
21    void dfs(const Graph<T> &g, int now, int prev, int &cnt) {
22        if (pre[now] != inf) {
23            low[prev] = min(low[prev], pre[now]);
24            return;
25        }
26        pre[now] = cnt;
27        low[now] = cnt;
28        cnt++;
29        for (int i=0; i<(int)(g[now].size()); ++i) {
30            int nxt = g[now][i].to;
31            //if g is an undirected graph
32            if (nxt == prev) continue;
33            dfs(g, nxt, now, cnt);
34        }
35        if (prev != -1) low[prev] = min(low[prev], low[now]);
36        if (prev != -1 && pre[prev] < low[now]) {
37            bridge.emplace_back(make_pair(prev, now));
38        }
39    }
40
41    void get_articulation(const Graph<T> &g, int now, int prev) {
42        sel[now] = true;
43        int art = 0;
44        for (int i=0; i<(int)(g[now].size()); ++i) {
45            int nxt = g[now][i].to;
46            //cout << now << ":" << nxt << endl;
47            if (sel[nxt]) continue;
48            // if g is an undirected graph
49            if (nxt == prev) continue;
50            if (now == 0 || pre[now] <= low[nxt]) art++;
51            get_articulation(g, nxt, now);
52        }
53        if ((now == 0 && art >= 2) || (now != 0 && art >= 1)) {
54            articulation.push_back(now);
55        }
56    }
57 };

```

2.7 Maximum Clique

```
1 #include "template.cpp"
2
3 int maximum_clique(const vector<vector<bool>>& G) {
4     // G: 隣接行列, 無向グラフ
5     int n = G.size();
6     vector<int> deg(n);
7     int M = 0;
8     for (int i = 0; i < n; ++i) {
9         for (int j = i+1; j < n; ++j) {
10             ++deg[i], ++deg[j], ++M;
11         }
12     }
13     vector<vector<bool>> g = G;
14     vector<bool> used(n);
15
16     int lim = sqrt(2*M), ret = 0;
17
18     for (int t = 0; t < n; ++t) {
19         int u = -1;
20         for (int i = 0; i < n; ++i) {
21             if (!used[i] && deg[i] < lim) {
22                 u = i;
23                 used[u] = true;
24                 break;
25             }
26         }
27
28         vector<int> neighbor;
29         if (u != -1) neighbor.push_back(u);
30         for (int v = 0; v < n; ++v) if (!used[v]) {
31             if (u == -1 || g[u][v]) {
32                 neighbor.push_back(v);
33             }
34         }
35
36         int sz = neighbor.size();
37         vector<int> bit(sz);
38         for (int i = 0; i < sz; i++) {
39             for (int j = i+1; j < sz; j++) {
40                 if (!g[neighbor[i]][neighbor[j]]) {
41                     bit[i] |= 1 << j;
42                     bit[j] |= 1 << i;
43                 }
44             }
45         }
46
47         vector<int> dp(1<<sz);
48         dp[0] = 1;
49         for (int s = 1; s < 1<<sz; ++s) {
50             int i = __builtin_ffs(s) - 1;
51
52             if (dp[s] = dp[s & ~(1<<i)] && (bit[i] & s) == 0) {
53                 ret = max(ret, __builtin_popcount(s));
54             }
55         }
56
57         if (u == -1) break;
58
59         for (auto v : neighbor) {
60             --deg[v], --deg[u];
61             g[u][v] = g[v][u] = false;
62         }
```

```
63     }
64
65     return ret;
66 }
```

2.8 Primal Dual

```
1 #include "template.cpp"
2
3 template<typename flow_t, typename cost_t>
4 struct PrimalDual {
5     const cost_t INF;
6
7     struct edge {
8         int to;
9         flow_t cap;
10        cost_t cost;
11        int rev;
12    };
13    vector<vector<edge>> g;
14    vector<cost_t> h, d;
15    vector<int> prevv, preve;
16
17    PrimalDual(int V) : g(V), INF(numeric_limits< cost_t >::max()) {}
18
19    void add_edge(int from, int to, flow_t cap, cost_t cost) {
20        g[from].push_back({to, cap, cost, (int)g[to].size()});
21        g[to].push_back({from, 0, -cost, (int)g[from].size()-1});
22    }
23
24    cost_t min_cost_flow(int s, int t, flow_t f) {
25        int V = (int)g.size();
26        cost_t ret = 0;
27        using Pi = pair<cost_t, int>;
28        priority_queue<Pi, vector<Pi>, greater<Pi>> que;
29        h.assign(V, 0);
30        preve.assign(V, -1);
31        prevv.assign(V, -1);
32
33        while (f > 0) {
34            d.assign(V, INF);
35            que.emplace(0, s);
36            d[s] = 0;
37            while (!que.empty()) {
38                Pi p = que.top(); que.pop();
39                if (d[p.second] < p.first) continue;
40                for (int i = 0; i < g[p.second].size(); i++) {
41                    edge &e = g[p.second][i];
42                    cost_t nextCost = d[p.second] + e.cost +
43                        h[p.second] - h[e.to];
44                    if (e.cap > 0 && d[e.to] > nextCost) {
45                        d[e.to] = nextCost;
46                        prevv[e.to] = p.second, preve[e.to] = i;
47                        que.emplace(d[e.to], e.to);
48                    }
49                }
50            }
51            if (d[t] == INF) return -1;
52            for (int v = 0; v < V; v++) h[v] += d[v];
53            flow_t addflow = f;
```

```

54     for (int v = t; v != s; v = prevv[v]) {
55         addflow = min(addflow, g[prevv[v]][preve[v]].cap);
56     }
57     f -= addflow;
58     ret += addflow * h[t];
59     for (int v = t; v != s; v = prevv[v]) {
60         edge &e = g[prevv[v]][preve[v]];
61         e.cap -= addflow;
62         g[v][e.rev].cap += addflow;
63     }
64 }
65 return ret;
66 }
67 };

```

2.9 SCC

```

1 #include "template.cpp"
2
3 template<typename T>
4 struct SCC {
5     int sz, cnt, num;
6     vi post, comp;
7     vector<pair<int, int>> vp;
8     vector<bool> sel;
9     Graph<T> revg;
10
11     SCC(const Graph<T> &g) {
12         sz = g.size();
13         cnt = 0;
14         num = 0;
15         post.resize(sz, -1);
16         comp.resize(sz, -1);
17         sel.resize(sz, false);
18         revg.resize(sz);
19     }
20
21     void build(const Graph<T> &g) {
22         for(int i=0; i<sz; ++i) {
23             if(sel[i]) continue;
24             sel[i] = true;
25             dfs1(g, i);
26         }
27
28         rev(g, revg);
29
30         for(int i=0; i<sz; ++i) {
31             vp.emplace_back(make_pair(post[i], i));
32         }
33         sort(vp.begin(), vp.end());
34         reverse(vp.begin(), vp.end());
35         sel.clear();
36         sel.resize(sz, false);
37         for(int i=0; i<sz; ++i) {
38             if(sel[vp[i].second]) continue;
39             sel[vp[i].second] = true;
40             comp[vp[i].second] = num;
41             dfs2(revg, vp[i].second);
42             num++;
43         }

```

```

44     }
45
46     vi get_comp() {return comp;}
47
48     Graph<T> build_graph(const Graph<T> &g) {
49         build(g);
50         vector<set<int>> s(sz);
51         Graph<T> res(sz);
52         for(int i=0; i<sz; ++i) {
53             for(int j=0; j<(int)g[i].size(); ++j) {
54                 s[comp[i]].insert(comp[g[i][j].to]);
55             }
56         }
57         for(int i=0; i<sz; ++i) {
58             for(auto j: s[i]) {
59                 if(i != j) res[i].push_back(edge<int>(i, j, 1));
60             }
61         }
62         return res;
63     }
64
65     void dfs1(const Graph<T> &g, int now) {
66         for(int i=0; i<(int)g[now].size(); ++i) {
67             int nxt = g[now][i].to;
68             if(sel[nxt]) continue;
69             sel[nxt] = true;
70             dfs1(g, nxt);
71         }
72         post[now] = cnt;
73         cnt++;
74     }
75
76     void rev(const Graph<T> &g, Graph<T> &revg) {
77         for(int i=0; i<sz; ++i) {
78             for(int j=0; j<(int)g[i].size(); ++j) {
79                 revg[g[i][j].to].push_back({
80                     g[i][j].to, g[i][j].src, g[i][j].cost});
81             }
82         }
83     }
84
85     void dfs2(const Graph<T> &revg, int now) {
86         for(int i=0; i<(int)revg[now].size(); ++i) {
87             int nxt = revg[now][i].to;
88             if(sel[nxt]) continue;
89             sel[nxt] = true;
90             comp[nxt] = num;
91             dfs2(revg, nxt);
92         }
93     }
94 };

```

2.10 Topological Sort

```

1 #include "template.cpp"
2
3 void topological_sort(const vector<vector<int>>& G, vector<int>& ord)
4 {
5     int n = G.size();
6     vector<int> num(n, 0);

```

```

7   ord.assign(n, 0);
8   for (int i = 0; i < n; ++i) {
9       for (auto u : G[i]) {
10          ++num[u];
11      }
12  }
13  stack<int> st;
14  for(int i = 0; i < n; ++i) {
15      if (num[i] == 0) {
16          st.push(i);
17      }
18  }
19  for (int k = 0; !st.empty(); ++k) {
20      int i = st.top(); st.pop();
21      ord[k] = i;
22      for (auto u : G[i]) {
23          if (--num[u] == 0) {
24              st.push(u);
25          }
26      }
27  }
28 }

```

2.11 Warshall Floyd

```

1 #include "template.cpp"
2
3 template<typename T>
4 void warshall_floyd(vector<vector<T>> &g) {
5     const auto INF = numeric_limits<T>::max();
6     int n = g.size();
7     for(int k = 0; k < n; k++) {
8         for(int i = 0; i < n; i++) {
9             for(int j = 0; j < n; j++) {
10                if(g[i][k] == INF || g[k][j] == INF) continue;
11                g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
12            }
13        }
14    }
15 }

```

3 Number

3.1 Mod

```

1 #include "../template.cpp"
2
3 ll powm(ll a, ll n, ll m) {
4     ll ret = 1;
5     while (n > 0) {
6         if (n & 1) (ret *= a) %= m;
7         (a *= a) %= m;
8         n >>= 1;

```

```

9     }
10    return ret;
11 }
12
13 ll invm(ll a, ll m) {
14     return powm(a, m-2, m);
15 }

```

3.2 ExtendedGCD

```

1 #include "../template.cpp"
2
3 ll extended_gcd(ll a, ll b, ll& x, ll& y) {
4     // solve ax + by = gcd(a, b)
5     if (b == 0) { x = 1; y = 0; return a; }
6     ll X, Y;
7     ll g = extended_gcd(b, a % b, X, Y);
8     x = Y; y = X - a/b * Y;
9     return g;
10 }

```

3.3 Combination

```

1 #include "mod.cpp"
2
3 vector<ll> fact;
4 void init_fact(int n, ll m) {
5     fact.assign(n+1, 1);
6     for (int i = 2; i <= n; ++i) {
7         (fact[i] = fact[i-1] * i) %= m;
8     }
9 }
10
11 // require init_fact(GREATER THAN OR EQUAL TO n, m)
12 ll C(ll n, ll r, ll m) {
13     return (fact[n] * invm((fact[r] * fact[n-r]) % m, m)) % m;
14 }
15
16 // Stirling number
17 // Stirling(n, k) := the number of cases
18 // to split n balls(distinguished)
19 // into k boxes(not distinguished)
20 // s.t. each box contains at least one ball.
21 //
22 // require init_fact(GREATER THAN OR EQUAL TO k, m)
23 ll Stirling(ll n, ll k, ll m) {
24     ll ret = 0;
25     for (ll l = 0; l <= k; ++l) {
26         ll tmp = (C(k, l, m) * powm((k-l) % m, n, m)) % m;
27         if (l & 1) tmp = (-tmp + m) % m;
28         (ret += tmp) %= m;
29     }
30     return (ret * invm(fact[k], m)) % m;
31 }
32

```

```

33 // Bell number
34 // Bell(n, k) := the number of cases
35 // to split n balls(distinguished)
36 // into k boxes(not distinguished)
37 //
38 // require init_fact(GREATER THAN OR EQUAL TO k, m)
39 ll Bell(ll n, ll k, ll m) {
40     ll ret = 0;
41     for (ll l = 0; l <= k; ++l) {
42         (ret += Stirling(n, l, m)) %= m;
43     }
44     return ret;
45 }
46
47 // Partition function
48 // Partition[k][n] := the number of cases
49 // to split n balls(not distinguished)
50 // into k boxes(not distinguished)
51 vector<vector<ll>> Part;
52 void init_partition(ll k, ll n, ll m) {
53     Part.assign(k+1, vector<ll>(n+1, 0));
54     Part[0][0] = 1;
55     for (int i = 1; i <= k; ++i) {
56         for (int j = 0; j <= n; ++j) {
57             if (j-i >= 0) {
58                 Part[i][j] = (Part[i-1][j] + Part[i][j-i]) % m;
59             } else {
60                 Part[i][j] = Part[i-1][j];
61             }
62         }
63     }
64 }

```

4 String

4.1 Rolling Hash

```

1 #include "../template.cpp"
2
3 struct RollingHash {
4     const int base = 9973;
5     const int mod[2] = {999999937, 1000000007};
6     vector<int> s;
7     vector<ll> hash[2], pow[2];
8
9     RollingHash(const vector<int> &cs) : s(cs) {
10         int n = s.size();
11         for (int id = 0; id < 2; ++id) {
12             hash[id].assign(n+1, 0);
13             pow[id].assign(n+1, 1);
14             for (int i = 0; i < n; ++i) {
15                 hash[id][i+1] = (hash[id][i] * base + s[i]) % mod[id];
16                 pow[id][i+1] = pow[id][i] * base % mod[id];
17             }
18         }
19     }
20
21     // get hash of s[l:r)

```

```

22     ll get(int l, int r, int id = 0) {
23         ll res = hash[id][r] - hash[id][l] * pow[id][r-l] % mod[id];
24         if (res < 0) res += mod[id];
25         return res;
26     }
27 };

```

4.2 Z Algorithm

```

1 #include "../template.cpp"
2
3 // GET A[i]: the longest common prefix size of S and S[i:n-1]
4 template<typename S>
5 void z_algorithm(const S& s, vector<int>& A) {
6     int n = s.size();
7     A.resize(n);
8     A[0] = n;
9     int i = 1, j = 0;
10    while (i < n) {
11        while (i+j < n && s[j] == s[i+j]) ++j;
12        A[i] = j;
13        if (j == 0) { ++i; continue; }
14        int k = 1;
15        while (i+k < n && k+A[k] < j) { A[i+k] = A[k]; ++k; }
16        i += k; j -= k;
17    }
18 }

```

5 Structure

5.1 Binary Indexed Tree

```

1 #include "../template.cpp"
2
3 template<typename T>
4 struct BIT {
5     vector<T> bit;
6     int sz;
7     BIT(int n) : sz(n+1), bit(n+1) {}
8     void add(int i, T x) {
9         i += 1;
10        while (i < sz) { bit[i] += x; i += i & -i; }
11    }
12    T sum(int i) {
13        i += 1; T s = 0;
14        while (i > 0) { s += bit[i]; i -= i & -i; }
15        return s;
16    }
17 };

```


5.2 Segment Tree

```
1 #include "../template.cpp"
2
3 template<typename M>
4 struct SegmentTree {
5     int sz;
6     vector<M> data;
7
8     // RMQ
9     const M e = numeric_limits<M>::max();
10    const function<M(M,M)> f = [](M a,M b){ return min(a,b); };
11
12    SegmentTree(int n) {
13        sz = 1;
14        while (sz < n) sz <= 1;
15        data.assign(2*sz, e);
16    }
17
18    void update(int k, const M &x) {
19        k += sz;
20        data[k] = x;
21        while (k >= 1) {
22            data[k] = f(data[2*k], data[2*k+1]);
23        }
24    }
25
26    M query(int a, int b, int k, int l, int r) {
27        if (r <= a || b <= l) {
28            return e;
29        } else if (a <= l && r <= b) {
30            return data[k];
31        } else {
32            return f(query(a,b,2*k, l,(l+r)/2),
33                    query(a,b,2*k+1,(l+r)/2,r));
34        }
35    }
36
37    M query(int a, int b) {
38        // return f[a,b)
39        return query(a, b, 1, 0, sz);
40    }
41
42    M operator[](int k) {
43        return data[k + sz];
44    }
45};
```

5.3 Lazy Segment Tree

```
1 #include "../template.cpp"
2
3 template<typename M, typename OM = M>
4 struct LazySegmentTree {
5     int sz;
6     vector<M> data;
7     vector<OM> lazy;
8
9     // RangeSumRangeAdd
```

```
10    const function<M(M,M)> f = [](M a,M b){ return a+b; };
11    const function<M(M,OM,int)> g = [](M a,OM b,int l){ return a+b*1; };
12    const function<OM(OM,OM)> h = [](OM a,OM b){ return a+b; };
13    const M e = 0;
14    const OM oe = 0;
15
16    LazySegmentTree(int n) {
17        sz = 1;
18        while (sz < n) sz <= 1;
19        data.assign(2*sz, e);
20        lazy.assign(2*sz, oe);
21    }
22
23    void propagate(int k, int len) {
24        if (lazy[k] == oe) return;
25        if (k < sz) {
26            lazy[2*k] = h(lazy[2*k], lazy[k]);
27            lazy[2*k+1] = h(lazy[2*k+1], lazy[k]);
28        }
29        data[k] = g(data[k], lazy[k], len);
30        lazy[k] = oe;
31    }
32
33    M update(int a, int b, const OM &x, int k, int l, int r) {
34        propagate(k, r - l);
35        if (r <= a || b <= l) {
36            return data[k];
37        } else if (a <= l && r <= b) {
38            lazy[k] = h(lazy[k], x);
39            propagate(k, r - l);
40            return data[k];
41        } else {
42            return data[k] = f(
43                update(a, b, x, 2*k, l, (l+r)/2),
44                update(a, b, x, 2*k+1, (l+r)/2, r));
45        }
46    }
47
48    void update(int a, int b, const OM &x) {
49        // update [a, b) with x.
50        update(a, b, x, 1, 0, sz);
51    }
52
53    M query(int a, int b, int k, int l, int r) {
54        propagate(k, r - l);
55        if (r <= a || b <= l) {
56            return e;
57        } else if (a <= l && r <= b) {
58            return data[k];
59        } else {
60            return f(
61                query(a, b, 2*k, l, (l+r)/2),
62                query(a, b, 2*k+1, (l+r)/2, r));
63        }
64    }
65
66    M query(int a, int b) {
67        // return f[a, b).
68        return query(a, b, 1, 0, sz);
69    }
70};
```

5.4 Union Find

```
1 #include "../template.cpp"
2
3 struct UnionFind
4 {
5     vector<int> par, sz;
6     UnionFind(int n) : par(n), sz(n, 1) {
7         for (int i = 0; i < n; ++i) par[i] = i;
8     }
9     int root(int x) {
10         if (par[x] == x) return x;
11         return par[x] = root(par[x]);
12     }
13     void merge(int x, int y) {
14         x = root(x);
15         y = root(y);
16         if (x == y) return;
17         if (sz[x] < sz[y]) swap(x, y);
18         par[y] = x;
19         sz[x] += sz[y];
20         sz[y] = 0;
21     }
22     bool issame(int x, int y) {
23         return root(x) == root(y);
24     }
25     int size(int x) {
26         return sz[root(x)];
27     }
28 };
```

5.5 Weighted Union Find

```
1 #include "../template.cpp"
2
3 template<typename A>
4 struct WeightedUnionFind
5 {
6     vector<int> par, sz;
7     vector<A> data; // data[x]: diff from root to x
8     WeightedUnionFind(int n, A e=0) :
9         par(n), sz(n, 1), data(n, e) {
10         for (int i = 0; i < n; ++i) par[i] = i;
11     }
12
13     int root(int x) {
14         if (par[x] == x) return x;
15         int r = root(par[x]);
16         data[x] += data[par[x]];
17         return par[x] = r;
18     }
19
20     A weight(int x) {
21         root(x);
22         return data[x];
23     }
24
25     A diff(int x, int y) {
26         // diff from x to y
```

```
27         return data[y] - data[x];
28     }
29
30     void merge(int x, int y, A w) {
31         // merge so that "diff from x to y" will be w.
32         w += weight(x); w -= weight(y);
33         x = root(x); y = root(y);
34         if (x == y) return;
35         if (sz[x] < sz[y]) swap(x, y), w = -w;
36         par[y] = x;
37         sz[x] += sz[y];
38         sz[y] = 0;
39         data[y] = w;
40     }
41
42     bool issame(int x, int y) {
43         return root(x) == root(y);
44     }
45 };
```

6 Others

6.1 Inversion

```
1 #include "../template.cpp"
2
3 #include "../structure/binary_indexed_tree.cpp"
4
5 template<typename T>
6 ll inversion(const vector<T>& a)
7 {
8     map<T,int> mp;
9     for (auto e : a) mp[e] = -1;
10     int sz = 0;
11     for (auto &q : mp) {
12         q.second = sz++;
13     }
14
15     BIT<int> bit(sz+1);
16     ll res = 0;
17     for (int i = 0; i < a.size(); ++i) {
18         res += i - bit.sum(mp[a[i]]);
19         bit.add(mp[a[i]], 1);
20     }
21     return res;
22 }
```

6.2 LIS

```
1 #include "../template.cpp"
2
3 template<typename T>
4 int LIS(const vector<T>& a) {
```

```
5  int n = a.size();
6  T INF = numeric_limits<T>::max();
7  vector<T> dp(n, INF);
8  for(int i = 0; i < n; ++i) {
9      *lower_bound(dp.begin(), dp.end(), a[i]) = a[i];
10 }
11 return distance(dp.begin(), lower_bound(dp.begin(), dp.end(), INF));
12 }
```

7 Vimrc

```
1 syntax enable
2 set number
3 set autoindent
4 set expandtab
5 set tabstop=4
6 set shiftwidth=4
```

8 Emacs

```
1 (package-initialize)
2
3 (setq inhibit-startup-message t)
4
5 (setq make-backup-files nil)
6
7 (setq delete-auto-save-files t)
8
9 (setq-default tab-width 4 indent-tabs-mode nil)
10
11 (setq eol-mnemonic-dos "(CRLF)")
12 (setq eol-mnemonic-mac "(CR)")
13 (setq eol-mnemonic-unix "(LF)")
14
15 (tool-bar-mode -1)
16 (menu-bar-mode -1)
17
18 (column-number-mode t)
19 (global-linum-mode t)
20 (blink-cursor-mode t)
21 (show-paren-mode 1)
22 (setq mouse-wheel-scroll-amount '(1 ((shift) . 5)))
23 (load-theme 'monokai t)
```
