

Team Notebook



template

```
#include <bits/stdc++.h>
using namespace std;

template <typename T> using vec = vector<T>;
using ll = long long;
#define sz(x) (int)x.size()
#define all(x) x.begin(), x.end()

int main() {
    cin.tie(0) -> sync_with_stdio(0);
    return 0;
}
```

Data Structures

hashmap

description: Hash map with mostly the same API as `unordered_map`, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

struct chash {
    const uint64_t C = ll(4e18 * acos(0)) | 71;
    ll operator()(ll x) const { return __builtin_bswap64(x * C); }
};

gp_hash_table<ll, int, chash> h({}, {}, {}, {}, {1 << 16});
```

order_statistic_tree

time: $O(\log n)$

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

template <class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
                tree_order_statistics_node_update>;
```

union_find

time: $O(\alpha(n))$

```
struct union_find {
    vec<int> e;
    union_find(int n) : e(n, -1) {}
    bool same(int a, int b) { return find(a) == find(b); }
    int size(int x) { return -e[find(x)]; }
    int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b)
            return false;
        if (e[a] > e[b])
            swap(a, b);
        e[a] += e[b];
        e[b] = a;
        return true;
    }
};
```

convex_hull_trick

description: Container where you can add lines of the form $kx + m$, and query maximum values at points x .

time: $O(\log n)$

```
struct line {
    mutable ll k, m, p;
    bool operator<(const line &o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct line_container : multiset<line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b);
    }
    bool isect(iterator x, iterator y) {
        if (y == end())
            return x->p = inf, 0;
        if (x->k == y->k)
            x->p = x->m > y->m ? inf : -inf;
        else
            x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
};
```

```

}
void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z))
        z = erase(z);
    if (x != begin() && isect(--x, y))
        isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
        isect(x, erase(y));
}
ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
}
};

```

sparse_table

time: $O(n \log n)$ preprocessing and $O(1)$ queries

```

template <class T, T F(T, T)> struct sparse_table {
    vec<vec<T>> jmp;
    sparse_table(const vec<T> &V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
            jmp.emplace_back(sz(V) - pw * 2 + 1);
            for (int j = 0; j < sz(jmp[k]); j++)
                jmp[k][j] = F(jmp[k - 1][j], jmp[k - 1][j + pw]);
        }
    }
    T query(int b, int e) { // query [b, e)
        assert(b < e); // or return inf if b == e
        int dep = 31 - __builtin_clz(e - b);
        return F(jmp[dep][b], jmp[dep][e - (1 << dep)]);
    }
};

```

iterative_segment_tree

```

template <class T, T unit, T F(T, T)> struct segment_tree {
    vector<T> s;
    int n;
    segment_tree(int n = 0, T def = unit) : s(2 * n, def), n(n) {}
    void update(int pos, T val) {
        for (s[pos += n] = val; pos /= 2;)
            s[pos] = F(s[pos * 2], s[pos * 2 + 1]);
    }
};

```

```

}
T query(int b, int e) { // query [b, e)
    T ra = unit, rb = unit;
    for (b += n, e += n; b < e; b /= 2, e /= 2) {
        if (b % 2)
            ra = F(ra, s[b++]);
        if (e % 2)
            rb = F(s[--e], rb);
    }
    return F(ra, rb);
}
};

```

Strings

kmp

description: $\text{pi}[x]$ computes the length of the longest prefix of s that ends at x , other than $s[0 \dots x]$ itself (abacaba \rightarrow 0010123). Can be used to find all occurrences of a string.

time: $O(n)$

```

vec<int> pi(const string &s) {
    vec<int> p(s.size());
    for (int i = 0; i < s.size(); i++) {
        int g = p[i - 1];
        while (g && s[i] != s[g])
            g = p[g - 1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}

vec<int> match(const string &s, const string &pat) {
    vec<int> p = pi(pat + '\0' + s), res;
    for (int i = p.size() - s.size(); i < p.size(); i++) {
        if (p[i] == pat.size())
            res.push_back(i - 2 * pat.size());
    }
    return res;
}

```

Graphs

topological_sort

time: $O(V + E)$

```

vec<int> topoSort(const vec<vec<int>> &gr) {
    vec<int> indeg(gr.size()), ret;
    for (auto &li : gr)
        for (int x : li)
            indeg[x]++;
    queue<int> q;
    for (int i = 0; i < gr.size(); i++)
        if (indeg[i] == 0)
            q.push(i);
    while (!q.empty()) {
        int i = q.front();
        ret.push_back(i);
        q.pop();
        for (int x : gr[i])
            if (--indeg[x] == 0)
                q.push(x);
    }
    return ret;
}

```

dinic

time: $O(VE \log U)$

```

struct dinic {
    struct edge_flow {
        int to, rev;
        ll c, oc;
        ll flow() { return max(oc - c, 0LL); } // if you need flows
    };
    vec<int> lvl, ptr, q;
    vec<vec<edge_flow>> adj;
    dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
    void addEdge(int a, int b, ll c, ll rcap = 0) {
        adj[a].push_back({b, sz(adj[b]), c, c});
        adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
    }
    ll dfs(int v, int t, ll f) {
        if (v == t || !f)
            return f;
        for (int &i = ptr[v]; i < sz(adj[v]); i++) {
            edge_flow &e = adj[v][i];
            if (lvl[e.to] == lvl[v] + 1)
                if (ll p = dfs(e.to, t, min(f, e.c))) {
                    e.c -= p, adj[e.to][e.rev].c += p;

```

```

                return p;
            }
        }
        return 0;
    }
    ll calc(int s, int t) {
        ll flow = 0;
        q[0] = s;
        for (int L = 0; L < 31; L++) {
            do { // 'int L=30' maybe faster for random data
                lvl = ptr = vec<int>(sz(q));
                int qi = 0, qe = lvl[s] = 1;
                while (qi < qe && !lvl[t]) {
                    int v = q[qi++];
                    for (edge_flow e : adj[v])
                        if (!lvl[e.to] && e.c >> (30 - L))
                            q[qi++] = e.to, lvl[e.to] = lvl[v] + 1;
                }
                while (ll p = dfs(s, t, LLONG_MAX))
                    flow += p;
            } while (lvl[t]);
        }
        return flow;
    }
    bool leftOfMinCut(int a) { return lvl[a] != 0; }
};

```
