1 データ構造

Union-Find

```
struct UnionFind{
    vector<int> dat;

    UnionFind(int sz){
        dat.assign(sz, -1);
    }

    bool unite(int x, int y){
        x = root(x), y = root(y);
        if(x == y) return(false);
        if(dat[x] > dat[y]) swap(x, y);
        dat[x] += dat[y];
        dat[y] = x;
        return(true);
    }

    int root(int k){
        if(dat[k] < 0) return(k);
        return(dat[k] = root(dat[k]));
    }

    int size(int k){
        return(-dat[root(k)]);
    }
};
```

Binary Indexed Tree

```
template< class T >
struct BinaryIndexedTree {
    vector< T > dat;

    BinaryIndexedTree(int sz) {
        dat.assign(++sz, 0);
    }

    T sum(int k){
        T ret = 0;
        for(++k; k > 0; k -= k & -k) ret += data[k];
        return ret;
    }

    void add(int k, T x){
        for(++k; k < data.size(); k += k & -k) data[k] += x;
    }
};
```

Segment Tree

```cpp
/*
!) 0-indexed
!) [a, b)に対する演算
!) merge, updateNodeを書く
*/
template<typename Monoid>
struct Segtree{
    int n;
    vector<Monoid> dat;
    Monoid m0; // データの初期化値
    Segtree(int sz, Monoid m0) : m0(m0){
        n = 1;
        while(n < sz) n *= 2;
        dat.assign(2*n-1, m0);
    }

    Monoid merge(Monoid a, Monoid b) // 区間をマージする二項演算
    void updateNode(int k, Monoid x) // 区間を操作する二項演算

    void update(int k, Monoid x){
        k += n-1;
        updateNode(k, x);
        while(k > 0) {
            k = (k-1)/2;
            dat[k] = merge(dat[k*2+1], dat[k*2+2]);
        }
    }
    Monoid query(int a, int b){
        Monoid L = m0, R = m0;
        int A = a+n-1;
        int B = b+n-1;
        while(A < B) {
            if((A&1) == 0) L = merge(L, dat[A++]);
            if((B&1) == 0) R = merge(dat[--B], R);
            A >>= 1;
            B >>= 1;
        }
        return merge(L, R);
    }
    Monoid operator[](const int &k) const { return dat[k+n-1]; }
};

//########## 例 ##########
// Range min (AOJ DSL_2_A)
Monoid merge(Monoid a, Monoid b){ return min(a,b); }
void updateNode(int k, Monoid x){ dat[k] = x; }
Segtree<LL> a(n, (1LL<<31)-1);

// Range add (AOJ DSL_2_B)
Monoid merge(Monoid a, Monoid b){ return a + b; }
void updateNode(int k, Monoid x){ dat[k] += x; }
Segtree<LL> a(n, 0);
```

遅延伝搬 Segment Tree

```
/*
!) 0-indexed 遅延伝搬セグメント木
!) [a, b)に対する演算
!) M0, L0, merge, updateNode, propagateを書く
*/
template <typename Monoid>
struct LazySegtree {
    int n;
    vector<Monoid> dat, lazy;

    Monoid M0, L0 // データと遅延配列の初期化値 queryに合わせて選択する

    LazySegtree(int sz, Monoid dat_init){
        n = 1;
        while (n < sz) n *= 2;
        dat.assign(2*n-1, dat_init);
        lazy.assign(2*n-1, L0);
    }

    Monoid merge(Monoid a, Monoid b) // 区間をマージする二項演算
    void updateNode(int k, Monoid x) // 区間を操作する二項演算
    void propagate(int k, int l, int r) // 遅延配列の伝搬のさせ方

    void eval(int k, int l, int r) {
        if(lazy[k] == L0) return;
        propagate(k, l, r);
        if(r-l > 1) {
            updateNode(2*k+1, lazy[k]);
            updateNode(2*k+2, lazy[k]);
        }
        lazy[k] = L0;
    }

    void update(int a, int b, Monoid x, int k, int l, int r) {
        eval(k, l, r);
        if (r <= a || b <= l) return;
        if (a <= l && r <= b) {
            updateNode(k, x);
            eval(k, l, r);
        }else{
            update(a, b, x, k*2+1, l, (l+r)/2);
            update(a, b, x, k*2+2, (l+r)/2, r);
            dat[k] = merge(dat[2*k+1], dat[2*k+2]);
        }
    }

    void update(int a, int b, Monoid x) {
        update(a, b, x, 0, 0, n);
    }

    Monoid query(int a, int b, int k, int l, int r) {
        eval(k, l, r);
        if (r <= a || b <= l) return M0;
        if (a <= l && r <= b) return dat[k];
        Monoid L = query(a, b, k*2+1, l, (l+r)/2);
```

```
        Monoid R = query(a, b, k*2+2, (l+r)/2, r);
        return merge(L, R);
    }

    Monoid query(int a, int b){
        return query(a, b, 0, 0, n);
    }
};

//########## 例 ##########
// Range update - min (AOJ DSL_2_F)
Monoid M0 = LLINF, L0 = LLINF;
Monoid merge(Monoid a, Monoid b){ return min(a, b); }
void updateNode(int k, Monoid x){ lazy[k] = x; }
void propagate(int k, int l, int r){ dat[k] = lazy[k]; }
LazySegtree<LL> seg(n+1, (1LL<<31)-1);

// Range update - sum (AOJ DSL_2_I)
Monoid M0 = 0, L0 = LLINF;
Monoid merge(Monoid a, Monoid b){ return a + b; }
void updateNode(int k, Monoid x){ lazy[k] = x; }
void propagate(int k, int l, int r){ dat[k] = lazy[k]*(r-l); }
LazySegtree<LL> seg(n+1, 0);

// Range add - min (AOJ DSL_2_H)
Monoid M0 = LLINF, L0 = 0;
Monoid merge(Monoid a, Monoid b){ return min(a, b); }
void updateNode(int k, Monoid x){ lazy[k] += x; }
void propagate(int k, int l, int r){ dat[k] += lazy[k]; }
LazySegtree<LL> seg(n+1, 0);

// Range add - sum (AOJ DSL_2_G)
Monoid M0 = 0, L0 = 0;
Monoid merge(Monoid a, Monoid b){ return a + b; }
void updateNode(int k, Monoid x){ lazy[k] += x; }
void propagate(int k, int l, int r){ dat[k] += lazy[k]*(r-l); }
LazySegtree<LL> seg(n+1, 0);
```

2 グラフ

Grid 上での BFS

```
int W, H;
vector<vector<char>> s;
vector<vector<int>> cost;
int bfs(){
    int dx[] = {0, 1, 0, -1}, dy[] = {1, 0, -1, 0};
    queue<pair<int, int>> que;
    que.push(make_pair(0, 0));
    cost[0][0] = 0;

    while(!que.empty()) {
        pair<int, int> p = que.front();
        que.pop();
        if(p == make_pair(H-1, W-1)){
            // ゴールに到達
            return cost[p.first][p.second];
        }
        for(int i = 0; i < 4; i++) {
            int ny = p.first + dy[i], nx = p.second + dx[i];
            if(nx < 0 || ny < 0 || nx >= W || ny >= H) continue;
            if(s[ny][nx] == '#') continue;
            if(cost[ny][nx] != -1) continue;

            cost[ny][nx] = cost[p.first][p.second] + 1;
            que.push(make_pair(ny, nx));
        }
    }
    return -1;
}
```

Dijkstra 法 (単一始点最短経路)

```
struct edge{
    int to, cost;
};
using WeightedGraph = vector<vector<edge>>;

vector<int> dijkstra(WeightedGraph &G, int st){
    vector<int> dist(G.size(), INF);
    using pi = pair<int, int>;
    priority_queue<pi, vector<pi>, greater<pi>> que;
    dist[st] = 0;
    que.push(mp(dist[st], st));
    while(!que.empty()){
        int cost, idx;
        tie(cost, idx) = que.top(); que.pop();
        if(dist[idx] < cost) continue;
        for(auto &e: G[idx]){
            if(dist[e.to] <= cost+e.cost) continue;
            dist[e.to] = cost+e.cost;
            que.push(mp(dist[e.to], e.to));
        }
```

```
    }
    return dist;
}
```

Bellman-Ford 法 (負路あり単一始点最短経路)

```
struct edge{
    int src, to, cost;
};
using Edges = vector<edge>;

vector<int> bellman_ford(Edges &E, int V, int st){
    vector<int> dist(V, INF);
    dist[st] = 0;
    rep(i, V-1){
        for(auto &e: E){
            if(dist[e.src] == INF) continue;
            dist[e.to] = min(dist[e.to], dist[e.src]+e.cost);
        }
    }
    for(auto &e: E){
        if(dist[e.src] == INF) continue;
        if(dist[e.to] > dist[e.src]+e.cost){
            // 負閉路が存在
            return vector<int>();
        }
    }
    return dist;
}
```

Warshall-Floyd 法 (全点対間最短経路)

```
using Graph = vector<vector<int>>;

void warshall_floyd(Graph &G){
    int V = G.size();
    rep(k, V)rep(i, V)rep(j, V){
        if(G[i][k] == INF || G[k][j] == INF) continue;
        G[i][j] = min(G[i][j], G[i][k]+G[k][j]);
    }
    // G[i][i] < 0が存在 <=> 負閉路が存在
}
```

Kruskal 法 (最小全域木)

```
// UnionFindが必要

struct edge{
    int src, to, cost;
};
using Edges = vector<edge>;

int kruskal(Edges &E, int V)
{
    sort(all(E), [](const edge &a, const edge &b)
    {
```

```
        return (a.cost < b.cost);
    });
    UnionFind tree(V);
    int res = 0;
    for(auto &e : E) {
        if(tree.unite(e.src, e.to)) res += e.cost;
    }
    return (res);
}
```

トポロジカルソート

```
struct edge{
    int to, cost;
};
using WeightedGraph = vector<vector<edge>>;

vector<int> tsort(WeightedGraph &G){
    vector<int> tsorted;
    vector<int> used(G.size(), 0);
    bool f = false;
    function<void(int)> dfs = [&](int u){
        if(used[u] > 0){
            if(used[u] == 1) f = true;
            return;
        }
        used[u] = 1;
        for(auto &e : G[u]) dfs(e.to);
        used[u] = 2;
        tsorted.pb(u);
    };
    rep(i, G.size()) dfs(i);
    if(f){
        // 閉路が存在
        return vector<int>();
    }
    reverse(all(tsorted));
    return tsorted;
}
```

Dinic 法 (最大流)

```
template<typename flow_t>
struct Dinic{
    const flow_t INF_flow_t = INF;   // WRITE HERE

    struct edge{
        int to;
        flow_t cap;
        int rev;
    };
    using WeightedGraph = vector<vector<edge>>;
    int V;
    WeightedGraph G;
    vector<int> itr, level;

```

```
    Dinic(int V) : V(V) { G.assign(V, vector<edge>()); }

    void add_edge(int from, int to, int cap) {
        G[from].push_back((edge){to, cap, (int)G[to].size()});
        G[to].push_back((edge){from, 0, (int)G[from].size()-1});
    }

    void bfs(int s) {
        level.assign(V, -1);
        queue<int> que;
        level[s] = 0;
        que.push(s);
        while (!que.empty()) {
            int v = que.front(); que.pop();
            for(auto &&e: G[v]){
                if (e.cap > 0 && level[e.to] < 0) {
                    level[e.to] = level[v] + 1;
                    que.push(e.to);
                }
            }
        }
    }

    flow_t dfs(int v, int t, flow_t f) {
        if(v == t) return f;
        for(int &i = itr[v]; i < (int)G[v].size(); i++) {
            edge &e = G[v][i];
            if (e.cap > 0 && level[v] < level[e.to]) {
                flow_t d = dfs(e.to, t, min(f, e.cap));
                if (d > 0) {
                    e.cap -= d;
                    G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }

    flow_t max_flow(int s, int t) {
        flow_t res = 0, f;
        while(bfs(s), level[t] >= 0) {
            itr.assign(V, 0);
            while((f = dfs(s, t, INF_flow_t)) > 0) res += f;
        }
        return res;
    }
};

// 最小流量制限付き最大流
// 各辺に[lb, ub]の容量の辺を張る
template<typename flow_t>
struct DinicWithLowerBound{
    Dinic<flow_t> flow;
    int S, T;
    flow_t sum_lb;

```

```
    DinicWithLowerBound(int V) : flow(V+2), S(V), T(V+1), sum_lb(0) {}

    void add_edge(int from, int to, flow_t lb, flow_t ub) {
        flow.add_edge(from, to, ub-lb);
        flow.add_edge(S, to, lb);
        flow.add_edge(from, T, lb);
        sum_lb += lb;
    }

    flow_t max_flow(int s, int t) {
        auto a = flow.max_flow(S, T);
        auto b = flow.max_flow(s, T);
        auto c = flow.max_flow(S, t);
        auto d = flow.max_flow(s, t);
        return (b == c && a + b == sum_lb) ? b+d : -1;
    }
};
```

二部マッチング

```
struct BipartiteMatching {
    using Graph = vector<vector<int>>;
    Graph G;
    vector<int> match, alive, used;
    int timestamp;

    BipartiteMatching(int n) : G(n), alive(n, 1),
                               used(n, 0), match(n, -1), timestamp(0) {}

    void add_edge(int u, int v) {
        G[u].push_back(v);
        G[v].push_back(u);
    }

    int dfs(int idx) {
        used[idx] = timestamp;
        for(auto &&to : G[idx]) {
            int w = match[to];
            if(alive[to] == 0) continue;
            if(w < 0 || (used[w] != timestamp && dfs(w))) {
                match[idx] = to;
                match[to] = idx;
                return 1;
            }
        }
        return 0;
    }

    int bipartite_matching() {
        int res = 0;
        for(int i = 0; i < G.size(); i++) {
            if(alive[i] == 0) continue;
            if(match[i] == -1) {
                ++timestamp;
                res += dfs(i);
            }
        }
```

```
        return res;
    }

    void output() {
        for(int i = 0; i < G.size(); i++) {
            if(i < match[i]) {
                cout << i << "-" << match[i] << endl;
            }
        }
    }
};
```

3 木

木の直径

```
struct edge{
    int to, cost;
};
using WeightedGraph = vector<vector<edge>>;
using pi = pair<int, int>;

pi dfs(WeightedGraph &G, int idx, int src){
    pi res(0, idx);
    for(auto &e : G[idx]) {
        if(e.to == src) continue;
        pi cost = dfs(G, e.to, idx);
        cost.first += e.cost;
        res = max(res, cost);
    }
    return res;
}

int tree_diameter(WeightedGraph &G)
{
    auto far = dfs(G, 0, -1);
    auto res = dfs(G, far.second, -1);
    return (res.first);
}
```

4 数学

GCD・LCM

```
LL gcd(LL a, LL b){
    if(a < b) swap(a, b);
    if(b == 0) return a;
    return gcd(b, a%b);
}

LL lcm(LL a, LL b){
    return a*b/gcd(a,b);
}
```

5 文字列

KMP 法

文字列 $S[0, i-1]$ の接頭辞と接尾辞が最大何文字一致しているかを記録した配列を $O(|S|)$ で構築する

```
vector<int> A(s.size()+1);
A[0] = -1;
int j = -1;
for (int i = 0; i < s.size(); i++) {
    while (j >= 0 && s[i] != s[j]) j = A[j];
    j++;
    A[i+1] = j;
}
```

6 テクニック

座標圧縮

```
vector<int> unzip = a;
map<int, int> zip;
sort(all(unzip));
unzip.erase(unique(all(unzip)), unzip.end());
for(int i=0; i<unzip.size(); i++) zip[unzip[i]] = i;
```

スライド最小値 $[i-k, i]$ の最小値を格納した vector を返す

```
vector<int> slide_min(vector<int> &a, int k){
    deque<int> deq;
    vector<int> b;
    rep(i, a.size()){
        // maxはここの不等号の向きを変える
        while(!deq.empty() && a[deq.back()] >= a[i]) deq.pop_back();
        deq.push_back(i);
        b.push_back(a[deq.front()]);
        if(i-k+1 >= 0 && deq.front() == i-k+1) deq.pop_front();
    }
    return b;
}
```