目次

1 テンプレート 2

2 グラフ 2

3 フロー 2

3.1 dinic 2

3.2 最小費用流 3

4 数学 4

4.1 modint 4

4.2 FFT 5

5 データ構造 6

5.1 セグメント木 6

5.2 遅延評価セグメント木 8

1 テンプレート

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

using ll = long long;
using P = pair<int, int>;
constexpr ll MOD = 1000000007;
constexpr int INF = 1 << 30;
#define REP(i, n) for (int i = 0, i_len = (n); i < i_len; i++)
#define ALL(v) (v).begin(), (v).end()
```

2 グラフ

3 フロー

3.1 dinic

最大流問題を解くアルゴリズム。計算量は $O(VE^2)$ だが実用上かなり高速なことが多い。

- Dinic flow(V): 構造体の宣言。V は頂点数。
- flow.add_edge(u, v, c): u → v に容量 c の辺を追加する
- flow.max_flow(s, t): s → t の最大流を返す

```cpp
//Dinic法 O(V^2E)
struct Dinic {
    int V;                                  // 頂点数
    vector<vector<vector<long long>>> graph; // グラフ
    vector<int> dis;                        // 始点からの距離
    vector<int> next;                       // 次に処理する頂点のメモ
    Dinic(int v) : V(v) { graph.resize(V); }
    void add_edge(int from, int to, long long capacity) {
        graph[from].push_back({to, capacity, (int)graph[to].size()});
        graph[to].push_back({from, 0, (int)graph[from].size() - 1});
    }
    void bfs(int s) {
        dis.assign(V, -1);
        dis[s] = 0;
        deque<int> pos = {s};
        while (pos.size()) {
            int now = pos[0];
            pos.pop_front();
            for (auto& to : graph[now]) {
                if (dis[to[0]] < 0 and to[1] > 0) {
                    dis[to[0]] = dis[now] + 1;
                    pos.emplace_back(to[0]);
                }
            }
        }
    }
    long long dfs(int v, int t, long long f) {
        if (v == t) return f;
        for (int& i = next[v]; i < graph[v].size(); i++) {
            int to = graph[v][i][0];
            long long& cap = graph[v][i][1];
            int rev = graph[v][i][2];
            if (cap > 0 and dis[v] < dis[to]) {
                long long d = dfs(to, t, min(f, cap));
                if (d > 0) {
                    cap -= d;
                    graph[to][rev][1] += d;
                    return d;
```

```cpp
                }
            }
        }
        return 0;
    }
    long long max_flow(int s, int t) {
        long long flow = 0;
        while (1) {
            bfs(s);
            if (dis[t] < 0) return flow;
            next.assign(V, 0);
            long long f;
            while ((f = dfs(s, t, LLONG_MAX)) > 0) flow += f;
        }
    }
};
```

3.2 最小費用流

最小費用流問題を解くアルゴリズム。計算量は $O(FE\log V)$

- MinCostFlow flow(V): 構造体の宣言。V は頂点数。
- flow.add_edge(u, v, c, d): u → v に容量 c, コスト d の辺を追加する
- flow.min_cost_flow(s, t, F): s → t に流量 F を流すときの最小コストを返す。流せない場合は-1 を返す。

```cpp
// 最小費用流 O(FElogV)
struct MinCostFlow {
    int V;
    vector<vector<vector<long long>>> g;  // g[from] = {{to, 容量, コスト, 逆辺のindex} ... }
    vector<long long> h, dis;             // ポテンシャル, 最短距離
    vector<int> prevv, preve;             // 直前の頂点, 辺

    MinCostFlow(int v) : V(v), g(v), dis(v), prevv(v), preve(v) {
    }

    void add_edge(int u, int v, long long c, long long d) {
        g[u].push_back({v, c, d, (int)g[v].size()});
        g[v].push_back({u, 0, -d, (int)g[u].size() - 1});
    }

    long long min_cost_flow(int s, int t, long long f) {
        long long res = 0;
        h.assign(V, 0);
        using Q = pair<long long, int>;
        while (f > 0) {
            priority_queue<Q, vector<Q>, greater<Q>> que;
            dis.assign(V, LLONG_MAX);
            dis[s] = 0;
            que.push({0, s});
            while (que.size()) {
                Q q = que.top();
                int v = q.second;
                que.pop();
                if (dis[v] < q.first) continue;
                for (int i = 0; i < g[v].size(); i++) {
                    auto edge = g[v][i];
                    int to = edge[0];
                    long long cap = edge[1], cost = edge[2];
                    if (cap > 0 and dis[to] > dis[v] + cost + h[v] - h[to]) {
                        dis[to] = dis[v] + cost + h[v] - h[to];
                        prevv[to] = v;
                        preve[to] = i;
                        que.push({dis[to], to});
                    }
                }
```

```
            }
            if (dis[t] == LLONG_MAX) return -1;
            for (int i = 0; i < V; i++) h[i] += dis[i];
            long long d = f;
            for (int i = t; i != s; i = prevv[i]) d = min(d, g[prevv[i]][preve[i]][1]);
            f -= d;
            res += d * h[t];
            for (int i = t; i != s; i = prevv[i]) {
                auto& edge = g[prevv[i]][preve[i]];
                edge[1] -= d;
                g[i][edge[3]][1] += d;
            }
        }
        return res;
    }
};
```

4 数学

4.1 modint

自動的に mod をとる構造体。mod が問題で固定かつ素数であるとき使用できる。

using mint = modint<1000000007>; 等のように定義して使用するのが推奨。

```
template <int64_t Modulus>
struct Modint {
    using mint = Modint;
    long long v;
    Modint() : v(0) {}
    Modint(long long x) {
        x %= Modulus;
        if (x < 0) x += Modulus;
        v = x;
    }
    const long long& val() const { return v; }
    // 代入演算子
    mint& operator+=(const mint rhs) {
        v += rhs.v;
        if (v >= Modulus) v -= Modulus;
        return *this;
    }
    mint& operator-=(const mint rhs) {
        if (v < rhs.v) v += Modulus;
        v -= rhs.v;
        return *this;
    }
    mint& operator*=(const mint rhs) {
        v = v * rhs.v % Modulus;
        return *this;
    }
    mint& operator/=(mint rhs) { return *this = *this * rhs.inv(); }
    // 累乗, 逆元
    mint pow(long long n) const {
        mint x = *this, res = 1;
        while (n) {
            if (n & 1) res *= x;
            x *= x;
            n >>= 1;
        }
        return res;
    }
    mint inv() const { return pow(Modulus - 2); }
    // 算術演算子
    mint operator+(const mint rhs) const { return mint(*this) += rhs; }
    mint operator-(const mint rhs) const { return mint(*this) -= rhs; }
    mint operator*(const mint rhs) const { return mint(*this) *= rhs; }
```

```
    mint operator/(const mint rhs) const { return mint(*this) /= rhs; }
    mint operator-() const { return mint() - *this; }  // 単項
};
```

入出力ストリームの実装

```
using mint = Modint<MOD>;

// 入出力ストリーム
istream& operator>>(istream& is, mint& x) {
    long long a;
    is >> a;
    x = a;
    return is;
}
ostream& operator<<(ostream& os, mint& x) {
    return os << x.val();
}
```

4.2 FFT

- encode(a): 整数型の配列 a を std::complex 型に変換。
- decode(a):std::complex 型の配列 a を 64bit 整数型に変換。配列の要素毎に実部を丸めて整数に変換している。
- FFT(a):std::complex 型で長さ n の配列 a をフーリエ変換する。整数型の配列は引数にとれないため、encode(a),decode(a) 等で適宜変換を行うこと。
- convolution(a,b): 長さ n の整数列 a, 長さ m の整数列 b の畳み込みを $O((n+m)\log(n+m))$ で計算する。畳み込み後の配列の要素がすべて double に収まる必要がある。

```
// 整数配列を複素数へ

vector<complex<double>> encode(vector<long long> &a){
    int N = a.size();
    vector<complex<double>> ret(N);
    for(int i = 0;i < N;i++){
        ret[i] = complex<double>(a[i],0);
    }
    return ret;
}

// 複素数配列を整数へ
vector<long long> decode(vector<complex<double>> &a){
    int N = a.size();
    vector<long long> ret(N);
    for(int i = 0;i < N;i++){
        ret[i] = (long long)round(a[i].real());
    }
    return ret;
}

// 非再帰
void FFT(vector<complex<double>> &a,int reverse = false) {
    int n = a.size();
    int h = 0;
    for (int i = 0; 1 << i < n; i++) h++;
    for (int i = 0; i < n; i++) {
        int j = 0;
        for (int k = 0; k < h; k++) j |= (i >> k & 1) << (h - 1 - k);
        if (i < j) swap(a[i], a[j]);
    }
    for (int b = 1; b < n; b *= 2) {
        for (int j = 0; j < b; j++) {
            double p2 = 2*acos(-1);
            if(reverse)p2 *= -1;
```

```
            complex<double> w = exp(complex<double>(0,p2*j/(double)(2*b)));
            for (int k = 0; k < n; k += b * 2) {
                complex<double> s = a[j+k];
                complex<double> t = a[j+k+b]*w;
                a[j+k] = s+t;
                a[j+k+b] = s-t;
            }
        }
    }
    if (reverse)for (int i = 0; i < n; i++) a[i] /= (double)n;
    return;
}

vector<long long> convolution(vector<long long> &a,vector<long long> &b){
    vector<complex<double>> A = encode(a),B = encode(b);
    int s = (int)a.size()+(int)b.size()-1;
    int t = 1;
    while(t < s)t *= 2;
    A.resize(t);B.resize(t);
    FFT(A,0);FFT(B,0);
    for(int i = 0;i < t;i++){
        A[i] *= B[i];
    }
    FFT(A,1);
    A.resize(s);
    return decode(A);
}
```

5 データ構造

5.1 セグメント木

モノイドを満たすデータ S に対し使用できるデータ構造。

長さ N の S の配列に対し、要素の 1 点更新、区間クエリを $O(\log N)$ で行える。モノイド S 同士の演算の計算量が $O(f(n))$ とき、すべての計算量が $O(f(n))$ 倍になる。

```
template <class S, S (*op)(S, S), S (*e)()>
struct SegmentTree {
  private:
    int _n, size, log;
    vector<S> dat;
    void update(int k) { dat[k] = op(dat[2 * k], dat[2 * k + 1]); }

  public:
    SegmentTree() : SegmentTree(0) {}
    SegmentTree(int n) : SegmentTree(vector<S>(n, e())) {}
    SegmentTree(const vector<S>& v) : _n(int(v.size())) {
        log = 0;
        while ((1 << log) < _n) log++;
        size = 1 << log;
        dat = vector<S>(2 * size, e());
        for (int i = 0; i < _n; i++) dat[size + i] = v[i];
        for (int i = size - 1; i >= 1; i--) {
            update(i);
        }
    }
    // a[p] = x
    void set(int p, S x) {
        p += size;
        dat[p] = x;
        for (int i = 1; i <= log; i++) update(p >> i);
    }
    // return a[p]
    S get(int p) const {
        return dat[p + size];
```

```
    }
    // return op(a[l], ..., a[r-1])
    S prod(int l, int r) const {
        S sml = e(), smr = e();
        l += size;
        r += size;
        while (l < r) {
            if (l & 1) sml = op(sml, dat[l++]);
            if (r & 1) smr = op(dat[--r], smr);
            l >>= 1;
            r >>= 1;
        }
        return op(sml, smr);
    }
    S all_prod() const { return dat[1]; }

    // SegmentTree上の二分探索（必要な場合）
    // return r, f(op(a[l], ..., a[r-1])) == true
    template <bool (*f)(S)>
    int max_right(int l) const {
        return max_right(l, [](S x) { return f(x); });
    }
    template <class F>
    int max_right(int l, F f) const {
        assert(f(e()));
        if (l == _n) return _n;
        l += size;
        S sm = e();
        do {
            while (l % 2 == 0) l >>= 1;
            if (!f(op(sm, dat[l]))) {
                while (l < size) {
                    l = (2 * l);
                    if (f(op(sm, dat[l]))) {
                        sm = op(sm, dat[l]);
                        l++;
                    }
                }
                return l - size;
            }
            sm = op(sm, dat[l]);
            l++;
        } while ((l & -l) != l);
        return _n;
    }
    // return l, f(op(a[l], ..., a[r-1])) == true
    template <bool (*f)(S)>
    int min_left(int r) const {
        return min_left(r, [](S x) { return f(x); });
    }
    template <class F>
    int min_left(int r, F f) const {
        assert(f(e()));
        if (r == 0) return 0;
        r += size;
        S sm = e();
        do {
            r--;
            while (r > 1 && (r % 2)) r >>= 1;
            if (!f(op(dat[r], sm))) {
                while (r < size) {
                    r = (2 * r + 1);
                    if (f(op(dat[r], sm))) {
                        sm = op(dat[r], sm);
                        r--;
                    }
                }
                return r + 1 - size;
```

```
            }
            sm = op(dat[r], sm);
        } while ((r & -r) != r);
        return 0;
    }
};
```

使用例

Range Minimum Query (RMQ)

```
int op(int a, int b) { return min(a, b); }
int e() { return INT32_MAX; }

int n;
SegmentTree<int, op, e> seg(n);
```

5.2 遅延評価セグメント木

モノイド S と、S に対する作用素 $f : S \to S$ に対し利用できるデータ構造。
長さ N の S の配列に対し、

- 区間 $[l, r)$ の要素に一括で f を作用 $(a_i \leftarrow f(a_i), l \leq i < r)$
- 区間 $[l, r)$ の要素の総積の取得

を $O(\log N)$ で行うことができる。

```
template <class S,
          S (*op)(S, S),
          S (*e)(),
          class F,
          S (*mapping)(F, S),
          F (*composition)(F, F),
          F (*id)()>
struct LazySegmentTree {
  private:
    int _n, size, log;
    vector<S> dat;
    vector<F> lz;
    void update(int k) { dat[k] = op(dat[2 * k], dat[2 * k + 1]); }
    void all_apply(int k, F f) {
        dat[k] = mapping(f, dat[k]);
        if (k < size) lz[k] = composition(f, lz[k]);
    }
    void push(int k) {
        all_apply(2 * k, lz[k]);
        all_apply(2 * k + 1, lz[k]);
        lz[k] = id();
    }
    int lower_bits(int x, int k) { return x & ((1 << k) - 1); }

  public:
    LazySegmentTree() : LazySegmentTree(0) {}
    LazySegmentTree(int n) : LazySegmentTree(vector<S>(n, e())) {}
    LazySegmentTree(const vector<S>& v) : _n(int(v.size())) {
        log = 0;
        while ((1 << log) < _n) log++;
        size = 1 << log;
        dat = vector<S>(2 * size, e());
        lz = vector<F>(size, id());
        for (int i = 0; i < _n; i++) dat[size + i] = v[i];
        for (int i = size - 1; i >= 1; i--) update(i);
    }
    // a[p] = x
    void set(int p, S x) {
```

```
        p += size;
        for (int i = log; i >= 1; i--) push(p >> i);
        dat[p] = x;
        for (int i = 1; i <= log; i++) update(p >> i);
    }
    // return a[p]
    S get(int p) {
        p += size;
        for (int i = log; i >= 1; i--) push(p >> i);
        return dat[p];
    }
    // return op(a[l], ..., a[r-1])
    S prod(int l, int r) {
        if (l == r) return e();
        l += size;
        r += size;
        for (int i = log; i >= 1; i--) {
            if (lower_bits(l, i) > 0) push(l >> i);
            if (lower_bits(r, i) > 0) push((r - 1) >> i);
        }
        S sml = e(), smr = e();
        while (l < r) {
            if (l & 1) sml = op(sml, dat[l++]);
            if (r & 1) smr = op(dat[--r], smr);
            l >>= 1;
            r >>= 1;
        }
        return op(sml, smr);
    }
    S all_prod() { return dat[1]; }
    // a[p] = f(a[p])
    void apply(int p, F f) {
        p += size;
        for (int i = log; i >= 1; i--) push(p >> i);
        dat[p] = mapping(f, dat[p]);
        for (int i = 1; i <= log; i++) update(p >> i);
    }
    // i = l...r-1 について a[i] = f(a[i])
    void apply(int l, int r, F f) {
        if (l == r) return;
        l += size;
        r += size;
        for (int i = log; i >= 1; i--) {
            if (lower_bits(l, i) > 0) push(l >> i);
            if (lower_bits(r, i) > 0) push((r - 1) >> i);
        }
        int l2 = l, r2 = r;
        while (l < r) {
            if (l & 1) all_apply(l++, f);
            if (r & 1) all_apply(--r, f);
            l >>= 1;
            r >>= 1;
        }
        l = l2;
        r = r2;
        for (int i = 1; i <= log; i++) {
            if (lower_bits(l, i) > 0) update(l >> i);
            if (lower_bits(r, i) > 0) update((r - 1) >> i);
        }
    }
    // SegmentTree上の二分探索 (必要な場合)
    // return r, f(op(a[l], ..., a[r-1])) == true
    template <bool (*g)(S)>
    int max_right(int l) {
        return max_right(l, [](S x) { return g(x); });
    }
    template <class G>
    int max_right(int l, G g) {
```

```cpp
        assert(g(e()));
        if (l == _n) return _n;
        l += size;
        for (int i = log; i >= 1; i--) push(l >> i);
        S sm = e();
        do {
            while (l % 2 == 0) l >>= 1;
            if (!g(op(sm, dat[l]))) {
                while (l < size) {
                    push(l);
                    l = (2 * l);
                    if (g(op(sm, dat[l]))) {
                        sm = op(sm, dat[l]);
                        l++;
                    }
                }
                return l - size;
            }
            sm = op(sm, dat[l]);
            l++;
        } while ((l & -l) != l);
        return _n;
    }
    // return l, f(op(a[l], ..., a[r-1])) == true
    template <bool (*g)(S)>
    int min_left(int r) {
        return min_left(r, [](S x) { return g(x); });
    }
    template <class G>
    int min_left(int r, G g) {
        assert(g(e()));
        if (r == 0) return 0;
        r += size;
        for (int i = log; i >= 1; i--) push((r - 1) >> i);
        S sm = e();
        do {
            r--;
            while (r > 1 && (r % 2)) r >>= 1;
            if (!g(op(dat[r], sm))) {
                while (r < size) {
                    push(r);
                    r = (2 * r + 1);
                    if (g(op(dat[r], sm))) {
                        sm = op(dat[r], sm);
                        r--;
                    }
                }
                return r + 1 - size;
            }
            sm = op(dat[r], sm);
        } while ((r & -r) != r);
        return 0;
    }
};
```

5.2.1 使用例

Range Update & Range Minimum Query

```cpp
constexpr int INF = INT32_MAX;
constexpr int ID = INT32_MAX;

int op(int a, int b) { return min(a, b); }
int e() { return INF; }
int mapping(int f, int a) { return (f == ID ? a : f); }
int composition(int f, int g) { return (f == ID ? g : f); }
int id() { return ID; }

```

```
int n;
LazySegmentTree<int, op, e, int, mapping, composition, id> seg(n);
```

Range Add & Range Sum Query

```
using S = pair<ll, ll>;

S op(S a, S b) { return S(a.first + b.first, a.second + b.second); }
S e() { return P(0, 0); }
S mapping(ll f, S x) { return S(x.first + f * x.second, x.second); }
ll composition(ll f, ll g) { return f + g; }
ll id() { return 0; }

int n;
vector<S> a(n, S(0, 1));
LazySegmentTree<S, op, e, ll, mapping, composition, id> seg(a);
```

Range Add & Range Minimum Query

```
int op(int a, int b) { return min(a, b); }
int e() { return INT32_MAX; }
int mapping(int f, int x) { return x + f; }
int composition(int f, int g) { return f + g; }
int id() { return 0; }

vector<int> a(n, 0);
LazySegmentTree<int, op, e, int, mapping, composition, id> seg(a);
```

Range Update & Range Sum Query

```
using S = pair<ll, ll>;
constexpr int ID = INT32_MAX;

S op(S a, S b) { return S(a.first + b.first, a.second + b.second); }
S e() { return S(0, 0); }
S mapping(int f, S x) { return (f == ID ? x : S(f * x.second, x.second)); }
int composition(int f, int g) { return (f == ID ? g : f); }
int id() { return ID; }

int n;
vector<S> a(n, S(0, 1));
LazySegmentTree<S, op, e, int, mapping, composition, id> seg(a);
```