# 目次

1	テンプレート	2
2	グラフ	2
3	フロー	2
3.1	dinic	2
4	数学	3
4.1	modint	3
4.2	FFT	4
5	データ構造	5
5.1	セグメント木	5
5.2	遅延評価セグメント木	7

## 1 テンプレート

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

using ll = long long;
using P = pair<int, int>;
constexpr ll MOD = 10000000007;
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()</pre>
```

## 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。

#### 宣言とメンバ関数

- 1. Dinic flow(V): 構造体の宣言。V は頂点数。
- 2. flow.add\_edge(u, v, c):  $u \rightarrow v$  に容量 c の辺を追加する
- 3. flow.max\_flow(s, t):  $s \to t$  の最大流を返す

```
//Dinic法 D(V^2E)
   struct Dinic {
       int V;
                                                   // 頂点数
3
4
       vector<vector<long long>>> graph;
                                                   // グラフ
                                                   // 始点からの距離
5
       vector<int> dis;
                                                   // 次に処理する頂点のメモ
6
       vector<int> next;
       Dinic(int v) : V(v) { graph.resize(V); }
7
       void add_edge(int from, int to, long long capacity) {
8
           graph[from].push_back({to, capacity, (int)graph[to].size()});
9
           graph[to].push_back({from, 0, (int)graph[from].size() - 1});
10
11
12
       void bfs(int s) {
           dis.assign(V, -1);
13
           dis[s] = 0;
14
15
           deque<int> pos = {s};
16
           while (pos.size()) {
17
               int now = pos[0];
18
               pos.pop_front();
               for (auto& to : graph[now]) {
19
                    if (dis[to[0]] < 0 \text{ and } to[1] > 0) {
20
                        dis[to[0]] = dis[now] + 1;
21
                        pos.emplace_back(to[0]);
22
                    }
23
               }
24
           }
25
26
27
       long long dfs(int v, int t, long long f) {
28
           if (v == t) return f;
           for (int& i = next[v]; i < graph[v].size(); i++) {</pre>
29
               int to = graph[v][i][0];
30
               long long& cap = graph[v][i][1];
31
               int rev = graph[v][i][2];
32
               if (cap > 0 and dis[v] < dis[to]) {</pre>
33
34
                   long long d = dfs(to, t, min(f, cap));
                    if (d > 0) {
35
36
                        cap -= d;
```

```
37
                         graph[to][rev][1] += d;
38
                         return d;
39
                     }
                }
40
            }
41
            return 0;
42
43
       long long max_flow(int s, int t) {
44
45
            long long flow = 0;
            while (1) {
46
47
                bfs(s);
                if (dis[t] < 0) return flow;</pre>
48
49
                next.assign(V, 0);
50
                long long f;
                while ((f = dfs(s, t, LLONG_MAX)) > 0) flow += f;
51
            }
52
       }
53
  };
54
```

## 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>
 1
   struct Modint {
2
       using mint = Modint;
3
       long long v;
5
       Modint() : v(0) {}
6
       Modint(long long x) {
7
           x %= Modulus;
           if (x < 0) x += Modulus;
8
9
           v = x;
       }
10
       const long long& val() const { return v; }
11
       // 代入演算子
12
       mint& operator+=(const mint rhs) {
13
           v += rhs.v;
14
15
           if (v >= Modulus) v -= Modulus;
           return *this;
16
17
       }
18
       mint& operator-=(const mint rhs) {
19
           if (v < rhs.v) v += Modulus;</pre>
           v -= rhs.v;
20
           return *this;
21
       }
22
       mint& operator*=(const mint rhs) {
23
           v = v * rhs.v % Modulus;
24
25
           return *this;
26
27
       mint& operator/=(mint rhs) { return *this = *this * rhs.inv(); }
28
       // 累乗,逆元
29
       mint pow(long long n) const {
30
           mint x = *this, res = 1;
31
           while (n) {
32
               if (n & 1) res *= x;
33
               x *= x;
               n >>= 1;
34
           }
35
36
           return res;
37
       mint inv() const { return pow(Modulus - 2); }
38
       // 算術演算子
39
40
       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

45 };
```

#### 入出力ストリームの実装

```
1 using mint = Modint<MOD>;
2
3
   // 入出カストリーム
  | istream& operator>>(istream& is, mint& x) {
5
       long long a;
6
       is >> a;
7
       x = a;
       return is;
8
   }
9
10
   ostream& operator<<(ostream& os, mint& x) {</pre>
       return os << x.val();</pre>
11
12
```

#### 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
2
   vector<complex<double>> encode(vector<long long> &a){
3
4
       int N = a.size();
       vector<complex<double>> ret(N);
5
       for(int i = 0;i < N;i++){</pre>
6
           ret[i] = complex<double>(a[i],0);
7
8
9
       return ret;
   }
10
11
   // 複素数配列を整数へ
12
13
   vector<long long> decode(vector<complex<double>> &a){
14
       int N = a.size();
15
       vector<long long> ret(N);
       for(int i = 0;i < N;i++){</pre>
16
           ret[i] = (long long)round(a[i].real());
17
       }
18
       return ret;
19
20 }
21
23
   void FFT(vector<complex<double>> &a,int reverse = false) {
24
       int n = a.size();
25
       int h = 0;
       for (int i = 0; 1 << i < n; i++) h++;
26
       for (int i = 0; i < n; i++) {
27
           int j = 0;
28
           for (int k = 0; k < h; k++) j \mid = (i >> k & 1) << (h - 1 - k);
29
           if (i < j) swap(a[i], a[j]);</pre>
30
31
32
       for (int b = 1; b < n; b *= 2) {
33
           for (int j = 0; j < b; j++) {
```

```
34
                double p2 = 2*acos(-1);
35
                if(reverse)p2 *= -1;
36
                complex<double> w = exp(complex<double>(0,p2*j/(double)(2*b)));
                for (int k = 0; k < n; k += b * 2) {
37
                    complex<double> s = a[j+k];
38
                    complex<double> t = a[j+k+b]*w;
39
                    a[j+k] = s+t;
40
                    a[j+k+b] = s-t;
41
42
                }
            }
43
44
45
       if (reverse)for (int i = 0; i < n; i++) a[i] /= (double)n;</pre>
46
       return;
   }
47
48
   vector<long long> convolution(vector<long long> &a,vector<long long> &b){
49
       vector<complex<double>> A = encode(a),B = encode(b);
50
       int s = (int)a.size()+(int)b.size()-1;
51
       int t = 1;
52
       while(t < s)t *= 2;
53
54
       A.resize(t); B.resize(t);
55
       FFT(A,0);FFT(B,0);
       for(int i = 0;i < t;i++){</pre>
56
57
            A[i] *= B[i];
58
       }
       FFT(A,1);
59
60
       A.resize(s);
       return decode(A);
61
   }
62
```

## 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 {
 3
     private:
 4
       int _n, size, log;
 5
       vector<S> dat;
       void update(int k) { dat[k] = op(dat[2 * k], dat[2 * k + 1]); }
 6
 7
     public:
 8
       SegmentTree() : SegmentTree(0) {}
 9
       SegmentTree(int n) : SegmentTree(vector<S>(n, e())) {}
10
       SegmentTree(const vector<S>& v) : _n(int(v.size())) {
11
12
            log = 0;
            while ((1 << log) < _n) log++;</pre>
13
            size = 1 << log;
14
15
            dat = vector < S > (2 * size, e());
            for (int i = 0; i < _n; i++) dat[size + i] = v[i];</pre>
16
17
            for (int i = size - 1; i >= 1; i--) {
                update(i);
18
            }
19
20
       // a[p] = x
21
22
       void set(int p, S x) {
            p += size;
23
            dat[p] = x;
^{24}
25
            for (int i = 1; i <= log; i++) update(p >> i);
26
27
       // return a[p]
```

```
28
       S get(int p) const {
29
           return dat[p + size];
30
       // return op(a[l], ..., a[r-1])
31
       S prod(int 1, int r) const {
32
           S sml = e(), smr = e();
33
           1 += size;
34
           r += size;
35
36
           while (1 < r) {</pre>
                if (1 & 1) sml = op(sml, dat[1++]);
37
38
                if (r & 1) smr = op(dat[--r], smr);
39
                1 >>= 1;
40
                r >>= 1;
           }
41
42
           return op(sml, smr);
43
       S all_prod() const { return dat[1]; }
44
45
       // SegmentTree上の二分探索 (必要な場合)
46
47
       // return r, f(op(a[l], ..., a[r-1])) == true
48
       template <bool (*f)(S)>
       int max_right(int 1) const {
49
           return max_right(1, [](S x) { return f(x); });
50
51
       }
52
       template <class F>
       int max_right(int 1, F f) const {
53
           assert(f(e()));
54
           if (1 == _n) return _n;
55
           1 += size;
56
           S sm = e();
57
           do {
58
                while (1 % 2 == 0) 1 >>= 1;
59
                if (!f(op(sm, dat[1]))) {
60
61
                    while (l < size) {</pre>
62
                        1 = (2 * 1);
63
                        if (f(op(sm, dat[1]))) {
64
                             sm = op(sm, dat[1]);
65
                             1++;
                        }
66
                    }
67
                    return 1 - size;
68
                }
69
70
                sm = op(sm, dat[1]);
71
                1++;
            } while ((1 & -1) != 1);
72
73
           return _n;
74
75
       // return l, f(op(a[l], ..., a[r-1])) == true
76
       template <bool (*f)(S)>
       int min_left(int r) const {
77
           return min_left(r, [](S x) { return f(x); });
78
79
       template <class F>
80
       int min_left(int r, F f) const {
81
82
            assert(f(e()));
           if (r == 0) return 0;
83
           r += size;
84
85
           S sm = e();
86
           do {
87
                while (r > 1 \&\& (r \% 2)) r >>= 1;
88
                if (!f(op(dat[r], sm))) {
89
                    while (r < size) {</pre>
90
                        r = (2 * r + 1);
91
                        if (f(op(dat[r], sm))) {
92
93
                             sm = op(dat[r], sm);
94
95
```

#### 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,
 2
              S (*op)(S, S),
 3
              S (*e)(),
 4
              class F,
 5
              S (*mapping)(F, S),
              F (*composition)(F, F),
 6
              F (*id)()>
 7
   struct LazySegmentTree {
 8
 9
     private:
10
       int _n, size, log;
       vector<S> dat;
11
12
       vector<F> lz;
       void update(int k) { dat[k] = op(dat[2 * k], dat[2 * k + 1]); }
13
       void all_apply(int k, F f) {
14
15
            dat[k] = mapping(f, dat[k]);
16
            if (k < size) lz[k] = composition(f, lz[k]);</pre>
17
18
       void push(int k) {
            all_apply(2 * k, lz[k]);
19
            all_apply(2 * k + 1, lz[k]);
20
21
            lz[k] = id();
22
       int lower_bits(int x, int k) { return x & ((1 << k) - 1); }</pre>
23
24
25
     public:
26
       LazySegmentTree() : LazySegmentTree(0) {}
27
       LazySegmentTree(int n) : LazySegmentTree(vector<S>(n, e())) {}
28
       LazySegmentTree(const vector<S>& v) : _n(int(v.size())) {
29
            log = 0;
            while ((1 << log) < _n) log++;</pre>
30
            size = 1 << log;
31
            dat = vector<S>(2 * size, e());
32
            lz = vector<F>(size, id());
33
34
            for (int i = 0; i < _n; i++) dat[size + i] = v[i];</pre>
            for (int i = size - 1; i >= 1; i--) update(i);
35
36
```

```
37
        // a[p] = x
38
        void set(int p, S x) {
39
            p += size;
            for (int i = log; i >= 1; i--) push(p >> i);
40
            dat[p] = x;
41
            for (int i = 1; i <= log; i++) update(p >> i);
42
43
        // return a[p]
44
        S get(int p) {
45
46
            p += size;
47
            for (int i = log; i >= 1; i--) push(p >> i);
            return dat[p];
48
49
        // return op(a[l], ..., a[r-1])
50
51
        S prod(int 1, int r) {
            if (1 == r) return e();
52
            1 += size;
53
            r += size;
54
            for (int i = log; i >= 1; i--) {
55
                if (lower_bits(1, i) > 0) push(1 >> i);
56
57
                if (lower_bits(r, i) > 0) push((r - 1) >> i);
58
            S sml = e(), smr = e();
59
60
            while (1 < r) {
61
                if (1 & 1) sml = op(sml, dat[1++]);
62
                if (r & 1) smr = op(dat[--r], smr);
63
                1 >>= 1;
64
                r >>= 1;
            }
65
66
            return op(sml, smr);
67
68
        S all_prod() { return dat[1]; }
69
        // a[p] = f(a[p])
        void apply(int p, F f) {
70
71
            p += size;
72
            for (int i = log; i >= 1; i--) push(p >> i);
73
            dat[p] = mapping(f, dat[p]);
74
            for (int i = 1; i <= log; i++) update(p >> i);
75
        // i = 1...r-1 について a[i] = f(a[i])
76
        void apply(int 1, int r, F f) {
77
78
            if (1 == r) return;
79
            1 += size;
            r += size;
80
            for (int i = log; i >= 1; i--) {
81
82
                if (lower_bits(1, i) > 0) push(1 >> i);
83
                if (lower_bits(r, i) > 0) push((r - 1) >> i);
84
            }
            int 12 = 1, r2 = r;
85
            while (1 < r) {
86
                if (1 & 1) all_apply(1++, f);
87
                if (r & 1) all_apply(--r, f);
88
                1 >>= 1;
89
                r >>= 1;
90
            }
91
            1 = 12;
92
93
            r = r2;
94
            for (int i = 1; i <= log; i++) {</pre>
95
                if (lower_bits(1, i) > 0) update(1 >> i);
96
                if (lower_bits(r, i) > 0) update((r - 1) >> i);
            }
97
98
        // SegmentTree上の二分探索 (必要な場合)
99
        // return r, f(op(a[l], ..., a[r-1])) == true
100
        template <bool (*g)(S)>
101
102
        int max_right(int 1) {
            return max_right(1, [](S x) { return g(x); });
103
104
```

```
105
        template <class G>
106
        int max_right(int 1, G g) {
107
             assert(g(e()));
            if (1 == _n) return _n;
108
            1 += size;
109
            for (int i = log; i >= 1; i--) push(1 >> i);
110
            S sm = e();
111
            do {
112
                 while (1 % 2 == 0) 1 >>= 1;
113
                 if (!g(op(sm, dat[1]))) {
114
115
                      while (l < size) {</pre>
116
                          push(1);
117
                          1 = (2 * 1);
                          if (g(op(sm, dat[1]))) {
118
119
                              sm = op(sm, dat[1]);
120
                              1++;
                          }
121
                     }
122
                     return 1 - size;
123
124
125
                 sm = op(sm, dat[1]);
126
                 1++;
127
             } while ((1 & -1) != 1);
128
             return _n;
129
        }
        // return l, f(op(a[l], ..., a[r-1])) == true
130
        template <bool (*g)(S)>
131
        int min_left(int r) {
132
            return min_left(r, [](S x) { return g(x); });
133
        }
134
        template <class G>
135
        int min_left(int r, G g) {
136
137
             assert(g(e()));
             if (r == 0) return 0;
138
            r += size;
139
            for (int i = log; i >= 1; i--) push((r - 1) >> i);
140
141
            S sm = e();
142
            do {
143
                 while (r > 1 \&\& (r \% 2)) r >>= 1;
144
                 if (!g(op(dat[r], sm))) {
145
                     while (r < size) {</pre>
146
147
                          push(r);
                          r = (2 * r + 1);
148
                          if (g(op(dat[r], sm))) {
149
150
                              sm = op(dat[r], sm);
151
152
                          }
                      }
153
154
                     return r + 1 - size;
155
                 sm = op(dat[r], sm);
156
             } while ((r & -r) != r);
157
            return 0;
158
159
160
    };
```

#### 5.2.1 使用例

Range Update & Range Minimum Query

```
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); }
```

```
8 int id() { return ID; }
9
10 int n;
11 LazySegmentTree<int, op, e, int, mapping, composition, id> seg(n);
```

Range Add & Range Sum Query

```
using S = pair<11, 11>;

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<11, 11>;
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);
```