

Contents

1	準備	1
1.1	Caps Lock と Control の入れ替え	1
1.2	init.el	1
1.3	tpl.cpp	1
1.4	get input	1
1.5	alias	2
2	文字列	2
2.1	マッチング	2
2.1.1	複数文字列マッチング (Aho-Corasick 法)	2
2.2	Suffix Array	2
3	グラフ	3
3.1	強連結成分分解	3
3.1.1	関節点	3
3.1.2	橋	3
3.1.3	強連結成分分解	4
3.2	フロー	4
3.2.1	最大流	4
3.2.2	二部マッチング	4
3.2.3	最小費用流	5
3.3	木	5
3.3.1	木の直径	5
3.3.2	最小全域木	5
3.3.3	最小シュタイナー木	6
3.4	包除原理	6
3.4.1	彩色数	6
4	数学	6
4.1	整数	6
4.1.1	剰余	6
4.1.2	カタラン数	7
4.1.3	乱数 (xor shift)	7
4.2	多項式	7
4.2.1	FFT(complex)	7
4.2.2	FFT(modulo)	8
4.2.3	積 (FMT)	8
4.2.4	逆元 (FMT)	8
4.2.5	平方根 (FMT)	8
4.3	行列	8
4.3.1	線形方程式の解 (Givens 消去法)	9
5	幾何	9
5.1	凸包	11

6	データ構造	11
6.1	Union-Find 木	11
6.2	赤黒木	11
6.3	永続赤黒木	12
6.4	wavelet 行列	13

## 1 準備

### 1.1 Caps Lock と Control の入れ替え

```
1 xmodmap -e 'remove Lock = Caps_Lock';
2 xmodmap -e 'add Control = Caps_Lock';
3 xmodmap -e 'keysym Caps_Lock = Control_L';
```

### 1.2 init.el

linum は emacs24 のみ

```
1 (keyboard-translate ?\C-h ?\C-?)
2 (global-linum-mode t)
3 (setq linum-format "%4d ")
```

### 1.3 tpl.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define rep(i,n) repi(i,0,n)
5 #define repi(i,a,b) for(int i=int(a);i<int(b);++i)
6 #define repit(it,u) for(auto it=begin(u);it!=end(u);++it)
7 #define all(u) begin(u),end(u)
8 #define uniq(u) (u).erase(unique(all(u)),end(u))
9 #define ll long
10 #define long int64_t
11 #define mp make_pair
12 #define pb push_back
13 #define eb emplace_back
14
15 bool input()
16 {
17     return true;
18 }
19
20 void solve()
21 {
22 }
23
24
25 int main()
26 {
27     cin.tie(0);
28     ios_base::sync_with_stdio(false);
29
30     while (input()) solve();
31 }
```

### 1.4 get input

```
1 wget -r http://(url of sample input)
```

## 1.5 alias

```
1 alias g++='g++ -g -O2 -std=gnu++0x';
2 alias emacs='emacs -nw';
```

## 2 文字列

### 2.1 マッチング

#### 2.1.1 複数文字列マッチング (Aho-Corasick 法)

$O(N + M)$

```
1 const int C = 128;
2
3 struct pma_node {
4     pma_node *next[C]; // use next[0] as failure link
5     vector<int> match;
6     pma_node() { fill(next, next + C, (pma_node *) NULL); }
7     ~pma_node() { rep(i, C) if (next[i] != NULL) delete next[i]; }
8 };
9
10 pma_node *construct_pma(const vector<string>& pat) {
11     pma_node *const root = new pma_node();
12     root->next[0] = root;
13     // construct trie
14     rep(i, pat.size()) {
15         const string& s = pat[i];
16         pma_node *now = root;
17         for (const char c : s) {
18             if (now->next[int(c)] == NULL) now->next[int(c)] = new pma_node();
19             now = now->next[int(c)];
20         }
21         now->match.pb(i);
22     }
23     // make failure links by BFS
24     queue<pma_node *> q;
25     repi(i, 1, C) {
26         if (root->next[i] == NULL) root->next[i] = root;
27         else {
28             root->next[i]->next[0] = root;
29             q.push(root->next[i]);
30         }
31     }
32     while (not q.empty()) {
33         auto now = q.front();
34         q.pop();
35         repi(i, 1, C) if (now->next[i] != NULL) {
36             auto next = now->next[i];
37             while (next->next[i] == NULL) next = next->next[0];
38             now->next[i]->next[0] = next->next[i];
39             vector<int> tmp;
40             set_union(all(now->next[i]->match), all(next->next[i]->match), back_inserter(tmp));
41             now->next[i]->match = tmp;
42             q.push(now->next[i]);
43         }
44     }
45     return root;
46 }
47
48 void match(pma_node*& now, const string s, vector<int>& ret) {
```

```

49     for (const char c : s) {
50         while (now->next[int(c)] == NULL) now = now->next[0];
51         now = now->next[int(c)];
52         for (const int e : now->match) ret[e] = true;
53     }
54 }

```

## 2.2 Suffix Array

find\_string():  $O(|T| \log |S|)$

S 中に T が含まれないなら -1, 含まれるならその先頭.

LCS():  $O(|S| + |T|)$

最長共通部分文字列. (先頭, 長さ) を返す.

```

1  const int MAX_N = 1000000;
2  int n, k;
3  int rnk[MAX_N+1], tmp[MAX_N+1], sa[MAX_N+1], lcp[MAX_N+1];
4
5  bool compare_sa(int i, int j) {
6      if(rnk[i] != rnk[j]) return rnk[i] < rnk[j];
7      else {
8          int ri = i + k <= n ? rnk[i+k] : -1;
9          int rj = j + k <= n ? rnk[j+k] : -1;
10         return ri < rj;
11     }
12 }
13
14 void construct_sa(string S, int *sa) {
15     n = S.length();
16     for(int i = 0; i <= n; i++) {
17         sa[i] = i;
18         rnk[i] = i < n ? S[i] : -1;
19     }
20     for(k = 1; k <= n; k*=2) {
21         sort(sa, sa+n+1, compare_sa);
22         tmp[sa[0]] = 0;
23         for(int i = 1; i <= n; i++) {
24             tmp[sa[i]] = tmp[sa[i-1]] + (compare_sa(sa[i-1], sa[i]) ? 1 : 0);
25         }
26         for(int i = 0; i <= n; i++) {
27             rnk[i] = tmp[i];
28         }
29     }
30 }
31
32 void construct_lcp(string S, int *sa, int *lcp) {
33     int n = S.length();
34     for(int i = 0; i <= n; i++) rnk[sa[i]] = i;
35     int h = 0;
36     lcp[0] = 0;
37     for(int i = 0; i < n; i++) {
38         int j = sa[rnk[i] - 1];
39         if(h > 0) h--;
40         for(; j + h < n && i + h < n; h++) {
41             if(S[j+h] != S[i+h]) break;
42         }
43         lcp[rnk[i] - 1] = h;
44     }
45 }
46
47 //===== 使用例 =====//
48 // 文字列検索(蟻本 p338 改)  $O(|T| \log |S|)$ 
49 // S 中に T が含まれないなら -1, 含まれるならその先頭

```

```

50 int find_string(string S, int *sa, string T) {
51     int a = 0, b = S.length();
52     while(b - a > 1) {
53         int c = (a + b) / 2;
54         if(S.compare(sa[c], T.length(), T) < 0) a = c;
55         else b = c;
56     }
57     return (S.compare(sa[b], T.length(), T) == 0) ? sa[b] : -1;
58 }
59
60 // 最長共通部分文字列(蟻本 p341 改) construct_sa 以外は  $O(|S| + |T|)$ 
61 // (先頭, 長さ) を返す
62 pair<int, int> LCS(string S, string T) {
63     int sl = S.length();
64     S += '\0' + T;
65     construct_sa(S, sa);
66     construct_lcp(S, sa, lcp);
67     int len = 0, pos = -1;
68     for(int i = 0; i < S.length(); i++) {
69         if((sa[i] < sl) != (sa[i+1] < sl)) && (len < lcp[i])) {
70             len = lcp[i];
71             pos = sa[i];
72         }
73     }
74     return make_pair(pos, len);
75 }

```

## 3 グラフ

```

1  struct edge {
2      int to; long w;
3      edge(int to, long w) : to(to), w(w) {}
4  };
5  typedef vector<vector<edge>> graph;
6
7  graph rev(const graph& G) {
8      const int n = G.size();
9      graph ret(n);
10     rep(i, n) for (const auto& e : G[i]) {
11         ret[e.to].eb(i, e.w);
12     }
13     return ret;
14 }

```

### 3.1 強連結成分分解

#### 3.1.1 関節点

$O(E)$

ある関節点 u がグラフを k 個に分割するとき art には k-1 個の u が含まれる. 不要な場合は unique を忘れないこと.

```

1  typedef vector<vector<int>> graph;
2
3  class articulation {
4      const int n;
5      graph G;
6      int cnt;
7      vector<int> num, low, art;

```

```

8     void dfs(int v) {
9         num[v] = low[v] = ++cnt;
10        for (int nv : G[v]) {
11            if (num[nv] == 0) {
12                dfs(nv);
13                low[v] = min(low[v], low[nv]);
14                if ((num[v] == 1 and num[nv] != 2) or
15                    (num[v] != 1 and low[nv] >= num[v])) {
16                    art[v] = true;
17                }
18            } else {
19                low[v] = min(low[v], num[nv]);
20            }
21        }
22    }
23    public:
24    articulation(const graph& G) : n(G.size()), G(G), cnt(0), num(n), low(n), art(n) {
25        rep(i, n) if (num[i] == 0) dfs(i);
26    }
27    vector<int> get() {
28        return art;
29    }
30 };

```

### 3.1.2 橋

$O(V + E)$

```

1     typedef vector<vector<int> > graph;
2
3     class bridge {
4     const int n;
5     graph G;
6     int cnt;
7     vector<int> num, low, in;
8     stack<int> stk;
9     vector<pair<int, int> > brid;
10    vector<vector<int> > comp;
11    void dfs(int v, int p) {
12        num[v] = low[v] = ++cnt;
13        stk.push(v), in[v] = true;
14        for (const int nv : G[v]) {
15            if (num[nv] == 0) {
16                dfs(nv, v);
17                low[v] = min(low[v], low[nv]);
18            } else if (nv != p and in[nv]) {
19                low[v] = min(low[v], num[nv]);
20            }
21        }
22        if (low[v] == num[v]) {
23            if (p != n) brid.eb(min(v, p), max(v, p));
24            comp.eb();
25            int w;
26            do {
27                w = stk.top();
28                stk.pop(), in[w] = false;
29                comp.back().pb(w);
30            } while (w != v);
31        }
32    }
33    public:
34    bridge(const graph& G) : n(G.size()), G(G), cnt(0), num(n), low(n), in(n) {
35        rep(i, n) if (num[i] == 0) dfs(i, n);
36    }
37    vector<pair<int, int> > get() {

```

```

38        return brid;
39    }
40    vector<vector<int> > components() {
41        return comp;
42    }
43 };

```

### 3.1.3 強連結成分分解

$O(V + E)$

```

1     typedef vector<vector<int> > graph;
2
3     class scc {
4     const int n;
5     graph G;
6     int cnt;
7     vector<int> num, low, in;
8     stack<int> stk;
9     vector<vector<int> > comp;
10    void dfs(int v) {
11        num[v] = low[v] = ++cnt;
12        stk.push(v), in[v] = true;
13        for (const int nv : G[v]) {
14            if (num[nv] == 0) {
15                dfs(nv);
16                low[v] = min(low[v], low[nv]);
17            } else if (in[nv]) {
18                low[v] = min(low[v], num[nv]);
19            }
20        }
21        if (low[v] == num[v]) {
22            comp.eb();
23            int w;
24            do {
25                w = stk.top();
26                stk.pop(), in[w] = false;
27                comp.back().pb(w);
28            } while (w != v);
29        }
30    }
31    public:
32    scc(const graph& G) : n(G.size()), G(G), cnt(0), num(n), low(n), in(n) {
33        rep(i, n) if (num[i] == 0) dfs(i);
34    }
35    vector<vector<int> > components() {
36        return comp;
37    }
38 };

```

## 3.2 フロー

### 3.2.1 最大流

$O(EV^2)$

```

1     const int inf = 1e9;
2     struct edge {
3         int to, cap, rev;
4         edge(int to, int cap, int rev) : to(to), cap(cap), rev(rev) {}
5     };
6     typedef vector<vector<edge> > graph;

```

```

7
8 void add_edge(graph& G, int from, int to, int cap) {
9     G[from].eb(to, cap, G[to].size());
10    G[to].eb(from, 0, G[from].size() - 1);
11 }
12
13 class max_flow {
14     const int n;
15     graph& G;
16     vector<int> level, iter;
17     void bfs(int s, int t) {
18         level.assign(n, -1);
19         queue<int> q;
20         level[s] = 0, q.push(s);
21         while (not q.empty()) {
22             const int v = q.front();
23             q.pop();
24             if (v == t) return;
25             for (const auto& e : G[v]) {
26                 if (e.cap > 0 and level[e.to] < 0) {
27                     level[e.to] = level[v] + 1;
28                     q.push(e.to);
29                 }
30             }
31         }
32     }
33     int dfs(int v, int t, int f) {
34         if (v == t) return f;
35         for (int& i = iter[v]; i < (int) G[v].size(); ++i) {
36             edge& e = G[v][i];
37             if (e.cap > 0 and level[v] < level[e.to]) {
38                 const int d = dfs(e.to, t, min(f, e.cap));
39                 if (d > 0) {
40                     e.cap -= d, G[e.to][e.rev].cap += d;
41                     return d;
42                 }
43             }
44         }
45         return 0;
46     }
47     public:
48     max_flow(graph& G) : n(G.size()), G(G) {}
49     int calc(int s, int t) {
50         int ret = 0, d;
51         while (bfs(s, t), level[t] >= 0) {
52             iter.assign(n, 0);
53             while ((d = dfs(s, t, inf)) > 0) ret += d;
54         }
55         return ret;
56     }
57 };

```

### 3.2.2 二部マッチング

$O(EV)$

```

1 int V;
2 vector<int> G[MAX_V];
3 int match[MAX_V];
4 bool used[MAX_V];
5
6 void add_edge(int u, int v){
7     G[u].push_back(v);
8     G[v].push_back(u);
9 }

```

```

10
11 bool dfs(int v){
12     used[v] = 1;
13     rep(i, G[v].size()){
14         int u = G[v][i], w = match[u];
15         if(w < 0 || !used[w] && dfs(w)){
16             match[v] = u;
17             match[u] = v;
18             return 1;
19         }
20     }
21     return 0;
22 }
23
24 int bi_matching(){
25     int res = 0;
26     memset(match, -1, sizeof(match));
27     rep(v, V) if(match[v] < 0){
28         memset(used, 0, sizeof(used));
29         if(dfs(v)) res++;
30     }
31     return res;
32 }

```

### 3.2.3 最小費用流

$O(VE \log V)$

```

1 const int inf = 1e9;
2 struct edge {
3     int to, cap, cost, rev;
4     edge(int to, int cap, int cost, int rev) : to(to), cap(cap), cost(cost), rev(rev) {}
5 };
6 typedef vector<vector<edge>> graph;
7
8 void add_edge(graph& G, int from, int to, int cap, int cost) {
9     G[from].eb(to, cap, cost, G[to].size());
10    G[to].eb(from, 0, -cost, G[from].size() - 1);
11 }
12
13 int min_cost_flow(graph& G, int s, int t, int f) {
14     const int n = G.size();
15     struct state {
16         int v, d;
17         state(int v, int d) : v(v), d(d) {}
18         bool operator <(const state& t) const { return d > t.d; }
19     };
20
21     int ret = 0;
22     vector<int> h(n, 0), dist, prev(n), prev_e(n);
23     while (f > 0) {
24         dist.assign(n, inf);
25         priority_queue<state> q;
26         dist[s] = 0, q.emplace(s, 0);
27         while (not q.empty()) {
28             const int v = q.top().v;
29             const int d = q.top().d;
30             q.pop();
31             if (dist[v] <= d) continue;
32             rep(i, G[v].size()) {
33                 const edge& e = G[v][i];
34                 if (e.cap > 0 and dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]) {
35                     dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
36                     prev[e.to] = v, prev_e[e.to] = i;
37                     q.emplace(e.to, dist[e.to]);
38                 }
39             }
40             f -= e.cap, e.cap = 0;
41             ret += dist[e.to] * e.cap;
42             h[e.to] = h[v] + e.cost;
43             prev_e[v] = i;
44             G[v][i].cap -= e.cap;
45             G[e.to][prev_e[e.to]].cap += e.cap;
46             prev_e[v] = -1;
47         }
48     }
49     return ret;
50 }

```

```

38         }
39     }
40 }
41 if (dist[t] == inf) return -1;
42 rep(i, n) h[i] += dist[i];
43
44 int d = f;
45 for (int v = t; v != s; v = prev[v]) {
46     d = min(d, G[prev[v]][prev_e[v]].cap);
47 }
48 f -= d, ret += d * h[t];
49 for (int v = t; v != s; v = prev[v]) {
50     edge& e = G[prev[v]][prev_e[v]];
51     e.cap -= d, G[v][e.rev].cap += d;
52 }
53 }
54 return ret;
55 }

```

### 3.3 木

#### 3.3.1 木の直径

ある点 (どこでもよい) から一番遠い点  $a$  を求める. 点  $a$  から一番遠い点までの距離がその木の直径になる.

#### 3.3.2 最小全域木

```

1  #include "disjoint_set.cpp"
2  #include "graph.cpp"
3
4  struct mst_edge {
5      int u, v; long w;
6      mst_edge(int u, int v, long w) : u(u), v(v), w(w) {}
7      bool operator <(const mst_edge& t) const { return w < t.w; }
8      bool operator >(const mst_edge& t) const { return w > t.w; }
9  };
10
11 graph kruskal(const graph& G) {
12     const int n = G.size();
13     vector<mst_edge> E;
14     rep(i, n) for (const auto& e : G[i]) {
15         if (i < e.to) E.eb(i, e.to, e.w);
16     }
17     sort(all(E));
18
19     graph T(n);
20     disjoint_set uf(n);
21     for (const auto& e : E) {
22         if (not uf.same(e.u, e.v)) {
23             T[e.u].eb(e.v, e.w);
24             T[e.v].eb(e.u, e.w);
25             uf.merge(e.u, e.v);
26         }
27     }
28     return T;
29 }
30
31 graph prim(const vector<vector<long>> &A, int s = 0) {
32     const int n = A.size();
33     graph T(n);
34     vector<int> done(n);

```

```

35     priority_queue<mst_edge, vector<mst_edge>, greater<mst_edge>> > q;
36     q.emplace(-1, s, 0);
37     while (not q.empty()) {
38         const auto e = q.top();
39         q.pop();
40         if (done[e.v]) continue;
41         done[e.v] = 1;
42         if (e.u >= 0) {
43             T[e.u].eb(e.v, e.w);
44             T[e.v].eb(e.u, e.w);
45         }
46         rep(i, n) if (not done[i]) {
47             q.emplace(e.v, i, A[e.v][i]);
48         }
49     }
50     return T;
51 }

```

#### 3.3.3 最小シュタイナー木

$O(4^{|T|}V)$

$g$  は無向グラフの隣接行列.  $T$  は使いたい頂点の集合.

```

1  int minimum_steiner_tree(vi &T, vvi &g){
2      int n = g.size(), t = T.size();
3      if(t <= 1) return 0;
4      vvi d(g); // all-pair shortest
5      rep(k,n)rep(i,n)rep(j,n) //Warshall Floyd
6          d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
7
8      int opt[1 << t][n];
9      rep(S,1<<t) rep(x,n)
10         opt[S][x] = INF;
11
12     rep(p,t) rep(q,n) // trivial case
13         opt[1 << p][q] = d[T[p]][q];
14
15     rep(i,S,1<<t){ // DP step
16         if(!(S & (S-1))) continue;
17         rep(p,n) rep(E,S)
18             if((E | S) == S)
19                 opt[S][p] = min(opt[S][p], opt[E][p] + opt[S-E][p]);
20         rep(p,n) rep(q,n)
21             opt[S][p] = min(opt[S][p], opt[S][q] + d[p][q]);
22     }
23
24     int ans = INF;
25     rep(S,1<<t) rep(q,n)
26         ans = min(ans, opt[S][q] + opt[((1<<t)-1)-S][q]);
27     return ans;
28 }

```

### 3.4 包除原理

#### 3.4.1 彩色数

$O(2^V V)$

$N[i] := i$  と隣接する頂点の集合 ( $i$  も含む)

```

1  const int MAX_V=16;
2  const int mod = 10009;

```

```

3  int N[MAX_V], I[1<<MAX_V], V;
4  inline int mpow(int a, int k){ return k==0? 1: k%2? a*mpow(a,k-1)%mod: mpow(a*a%mod,k
    /2);}
5
6  bool can(int k){
7      int res = 0;
8      rep(S, 1<<V){
9          if(__builtin_popcountll(S)%2) res -= mpow(I[S], k);
10         else res += mpow(I[S],k);
11     }
12     return (res%mod+mod)%mod;
13 }
14
15 int color_number(){
16     memset(I, 0, sizeof(I));
17     I[0] = 1;
18     repi(S,1,1<<V){
19         int v = 0;
20         while(!(S&(1<<v))) v++;
21         I[S] = I[S-(1<<v)] + I[S&(~N[v])];
22     }
23     int lb = 0, ub = V, mid;
24     while(ub-lb>1){
25         mid = (lb+ub)/2;
26         if(can(mid)) ub = mid;
27         else lb = mid;
28     }
29     return ub;
30 }

```

## 4 数学

### 4.1 整数

#### 4.1.1 剰余

```

1  // (x, y) s.t. a x + b y = gcd(a, b)
2  long extgcd(long a, long b, long& x, long& y) {
3      long g = a; x = 1, y = 0;
4      if (b != 0) g = extgcd(b, a % b, y, x), y -= (a / b) * x;
5      return g;
6  }
7
8  // repi(i, 2, n) mod_inv[i] = mod_inv[m % i] * (m - m / i) % m
9  long mod_inv(long a, long m) {
10     long x, y;
11     if (extgcd(a, m, x, y) != 1) return 0;
12     return (x % m + m) % m;
13 }
14
15 // a mod p where n! = a p^e in O(log_p n)
16 long mod_fact(long n, long p, long& e) {
17     const int P = 1000010;
18     static long fact[P] = {1};
19     static bool done = false;
20     if (not done) {
21         repi(i, 1, P) fact[i] = fact[i - 1] * i % p;
22         done = true;
23     }
24     e = 0;
25     if (n == 0) return 1;
26     long ret = mod_fact(n / p, p, e);
27     e += n / p;

```

```

28     if (n / p % 2) return ret * (p - fact[n % p]) % p;
29     return ret * fact[n % p] % p;
30 }
31
32 // nCk mod p
33 long mod_binom(long n, long k, long p) {
34     if (k < 0 or n < k) return 0;
35     long e1, e2, e3;
36     long a1 = mod_fact(n, p, e1);
37     long a2 = mod_fact(k, p, e2);
38     long a3 = mod_fact(n - k, p, e3);
39     if (e1 > e2 + e3) return 0;
40     return a1 * mod_inv(a2 * a3 % p, p) % p;
41 }
42
43 // a^b mod m
44 long mod_pow(long a, long b, long m) {
45     long ret = 1;
46     do {
47         if (b & 1) ret = ret * a % m;
48         a = a * a % m;
49     } while (b >= 1);
50     return ret;
51 }

```

#### 4.1.2 カタラン数

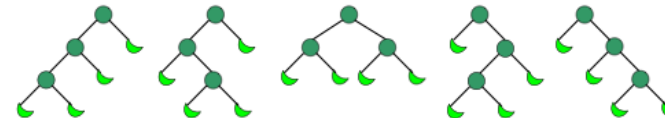
$n \leq 16$  程度が限度.  $n \geq 1$  について以下が成り立つ.

$$\begin{aligned}
 C_n &= \frac{1}{n+1} \binom{2n}{n} \\
 &= \binom{2n}{n} - \binom{2n}{n-1}
 \end{aligned}$$

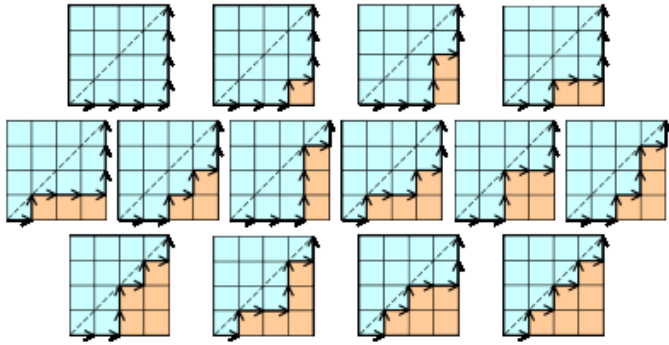
$n$  が十分大きいとき, カタラン数は以下に近似できる.

$$C_n = \frac{4^n}{n^{3/2} \sqrt{\pi}}$$

() を正しく並べる方法, 二分木, 格子状の経路の数え上げ, 平面グラフの交差などに使われる.  
 $C_3 = 5$



$C_4 = 14$



#### 4.1.3 乱数 (xor shift)

周期は  $2^{128} - 1$

```
1 unsigned xorshift() {
2     static unsigned x = 123456789;
3     static unsigned y = 362436069;
4     static unsigned z = 521288629;
5     static unsigned w = 88675123;
6     unsigned t;
7     t = x ^ cb^86 (x << 11);
8     x = y; y = z; z = w;
9     return w = (w ^ cb^86 (w >> 19)) ^ cb^86 (t ^ cb^86 (t >> 8));
10 }
```

## 4.2 多項式

FFT は基本定数重めなので TLE に注意する。

#### 4.2.1 FFT(complex)

$O(N \log N)$

複素数を用いた FFT. 変換する vector のサイズは 2 の冪乗にすること。

```
1 typedef complex<double> cd;
2 vector<cd> fft(vector<cd> f, bool inv){
3     int n, N = f.size();
4     for(n=0;;n++) if(N == (1<<n)) break;
5     rep(m,N){
6         int m2 = 0;
7         rep(i,n) if(m&(1<<i)) m2 |= (1<<(n-1-i));
8         if(m < m2) swap(f[m], f[m2]);
9     }
10
11     for(int t=1;t<N;t*=2){
12         double theta = acos(-1.0) / t;
13         cd w(cos(theta), sin(theta));
14         if(inv) w = cd(cos(theta), -sin(theta));
15         for(int i=0;i<N;i+=2*t){
16             cd power(1.0, 0.0);
17             rep(j,t){
18                 cd tmp1 = f[i+j] + f[i+t+j] * power;
19                 cd tmp2 = f[i+j] - f[i+t+j] * power;
```

```
20                 f[i+j] = tmp1;
21                 f[i+t+j] = tmp2;
22                 power = power * w;
23             }
24         }
25     }
26     if(inv) rep(i,N) f[i] /= N;
27     return f;
28 }
```

#### 4.2.2 FFT(modulo)

$O(N \log N)$

剰余環を用いた FFT(FMT). 変換する vector のサイズは 2 の冪乗にすること. mod は  $a * 2^e + 1$  の形。

```
1 #include "number_theory.cpp"
2
3 const int mod = 7*17*(1<<23)+1;
4 vector<int> fmt(vector<int> f, bool inv){
5     int e, N = f.size();
6     // assert((N&(N-1))==0 and "f.size() must be power of 2");
7     for(e=0;;e++) if(N == (1<<e)) break;
8     rep(m,N){
9         int m2 = 0;
10        rep(i,e) if(m&(1<<i)) m2 |= (1<<(e-1-i));
11        if(m < m2) swap(f[m], f[m2]);
12    }
13    for(int t=1; t<N; t*=2){
14        int r = pow_mod(3,(mod-1)/(t*2),mod);
15        if(inv) r = mod_inverse(r,mod);
16        for(int i=0; i<N; i+=2*t){
17            int power = 1;
18            rep(j,t){
19                int x = f[i+j], y = 1LL*f[i+t+j]*power%mod;
20                f[i+j] = (x+y)%mod;
21                f[i+t+j] = (x-y+mod)%mod;
22                power = 1LL*power*r%mod;
23            }
24        }
25    }
26    if(inv) for(int i=0,ni=mod_inv(N,mod);i<N;i++) f[i] = 1LL*f[i]*ni%mod;
27    return f;
28 }
```

#### 4.2.3 積 (FMT)

$O(N \log N)$

`poly_mul()` が必要。

```
1 vector<int> poly_mul(vector<int> f, vector<int> g){
2     int N = max(f.size(),g.size())*2;
3     f.resize(N); g.resize(N);
4     f = fmt(f,0); g = fmt(g,0);
5     rep(i,N) f[i] = 1LL*f[i]*g[i]%mod;
6     f = fmt(f,1);
7     return f;
8 }
```



#### 4.2.4 逆元 (FMT)

$O(N \log N)$

*extgcd(), mod\_inverse(), poly\_mul(), fmt() が必要。*

```
1 vector<int> poly_inv(const vector<int> &f){
2     int N = f.size();
3     vector<int> r(1, mod_inv(f[0], mod));
4     for(int k = 2; k <= N; k <= 1){
5         vector<int> nr = poly_mul(poly_mul(r, r), vector<int>(f.begin(), f.begin()+k));
6         nr.resize(k);
7         rep(i, k/2) {
8             nr[i] = (2*r[i]-nr[i]+mod)%mod;
9             nr[i+k/2] = (mod-nr[i+k/2])%mod;
10        }
11        r = nr;
12    }
13    return r;
14 }
```

#### 4.2.5 平方根 (FMT)

$O(N \log N)$

*extgcd(), mod\_inverse(), poly\_inv(), poly\_mul(), fmt() が必要。*

```
1 const int inv2 = (mod+1)/2;
2 vector<int> poly_sqrt(const vector<int> &f) {
3     int N = f.size();
4     vector<int> s(1, 1); // s[0] = sqrt(f[0])
5     for(int k = 2; k <= N; k <= 1) {
6         s.resize(k);
7         vector<int> ns = poly_mul(poly_inv(s), vector<int>(f.begin(), f.begin()+k));
8         ns.resize(k);
9         rep(i, k) s[i] = 1LL*(s[i]+ns[i])*inv2%mod;
10    }
11    return s;
12 }
```

#### 4.3 行列

```
1 typedef double number;
2 typedef vector<number> vec;
3 typedef vector<vec> mat;
4
5 vec mul(const mat& A, const vec& x) {
6     const int n = A.size();
7     vec b(n);
8     rep(i, n) rep(j, A[0].size()) {
9         b[i] = A[i][j] * x[j];
10    }
11    return b;
12 }
13
14 mat mul(const mat& A, const mat& B) {
15     const int n = A.size();
16     const int o = A[0].size();
17     const int m = B[0].size();
18     mat C(n, vec(m));
19     rep(i, n) rep(k, o) rep(j, m) {
20         C[i][j] += A[i][k] * B[k][j];
21    }
```

```
21 }
22     return C;
23 }
24
25 mat pow(mat A, long m) {
26     const int n = A.size();
27     mat B(n, vec(n));
28     rep(i, n) B[i][i] = 1;
29     do {
30         if (m & 1) B = mul(B, A);
31         A = mul(A, A);
32     } while (m >= 1);
33     return B;
34 }
35
36 const number eps = 1e-4;
37
38 // determinant; O(n^3)
39 number det(mat A) {
40     int n = A.size();
41     number D = 1;
42     rep(i, n){
43         int pivot = i;
44         rep(j, i+1, n)
45             if (abs(A[j][i]) > abs(A[pivot][i])) pivot = j;
46         swap(A[pivot], A[i]);
47         D *= A[i][i] * (i != pivot ? -1 : 1);
48         if (abs(A[i][i]) < eps) break;
49         rep(j, i+1, n)
50             for(int k=n-1; k>=i; --k)
51                 A[j][k] -= A[i][k] * A[j][i] / A[i][i];
52     }
53     return D;
54 }
55
56 // rank; O(n^3)
57 int rank(mat A) {
58     int n = A.size(), m = A[0].size(), r = 0;
59     for(int i = 0; i < m and r < n; i++){
60         int pivot = r;
61         rep(j, r+1, n)
62             if (abs(A[j][i]) > abs(A[pivot][i])) pivot = j;
63         swap(A[pivot], A[r]);
64         if (abs(A[r][i]) < eps) continue;
65         for(int k=m-1; k>=i; --k)
66             A[r][k] /= A[r][i];
67         rep(j, r+1, n) rep(k, i, m)
68             A[j][k] -= A[r][k] * A[j][i];
69         ++r;
70     }
71     return r;
72 }
```

#### 4.3.1 線形方程式の解 (Givens 消去法)

$O(N^3)$

```
1 // Givens elimination; O(n^3)
2
3 typedef double number;
4 typedef vector<vector<number>> matrix;
5
6 inline double my_hypot(double x, double y) { return sqrt(x * x + y * y); }
7 inline void givens_rotate(number& x, number& y, number c, number s) {
8     number u = c * x + s * y, v = -s * x + c * y;
```

```

9     x = u, y = v;
10 }
11 vector<number> givens(matrix A, vector<number> b) {
12     const int n = b.size();
13     rep(i, n) rep(j, i + 1, n) {
14         const number r = my_hypot(A[i][i], A[j][i]);
15         const number c = A[i][i] / r, s = A[j][i] / r;
16         givens_rotate(b[i], b[j], c, s);
17         rep(k, i, n) givens_rotate(A[i][k], A[j][k], c, s);
18     }
19     for (int i = n - 1; i >= 0; --i) {
20         rep(j, i + 1, n) b[i] -= A[i][j] * b[j];
21         b[i] /= A[i][i];
22     }
23     return b;
24 }

```

## 5 幾何

```

1 // constants and eps-considered operators
2
3 const double eps = 1e-8; // choose carefully!
4 const double pi = acos(-1.0);
5
6 inline bool lt(double a, double b) { return a < b - eps; }
7 inline bool gt(double a, double b) { return lt(b, a); }
8 inline bool le(double a, double b) { return !lt(b, a); }
9 inline bool ge(double a, double b) { return !lt(a, b); }
10 inline bool ne(double a, double b) { return lt(a, b) or lt(b, a); }
11 inline bool eq(double a, double b) { return !ne(a, b); }
12
13 // points and lines
14
15 typedef complex<double> point;
16
17 inline double dot (point a, point b) { return real(conj(a) * b); }
18 inline double cross(point a, point b) { return imag(conj(a) * b); }
19
20 struct line {
21     point a, b;
22     line(point a, point b) : a(a), b(b) {}
23 };
24
25 /*
26  * Here is what ccw(a, b, c) returns:
27  *
28  *      1
29  * -----
30  *  2 |a  0  b| -2
31  * -----
32  *      -1
33  *
34  * Note: we can implement intersectPS(p, s) as !ccw(s.a, s.b, p).
35  */
36 int ccw(point a, point b, point c) {
37     b -= a, c -= a;
38     if (cross(b, c) > eps) return +1;
39     if (cross(b, c) < eps) return -1;
40     if (dot(b, c) < eps) return +2; // c -- a -- b
41     if (lt(norm(b), norm(c))) return -2; // a -- b -- c
42     return 0;
43 }
44 bool intersectLS(const line& l, const line& s) {

```

```

45     return ccw(l.a, l.b, s.a) * ccw(l.a, l.b, s.b) <= 0;
46 }
47 bool intersectSS(const line& s, const line& t) {
48     return intersectLS(s, t) and intersectLS(t, s);
49 }
50 bool intersectLL(const line& l, const line& m) {
51     return ne(cross(l.b - l.a, m.b - m.a), 0.0) // not parallel
52         or eq(cross(l.b - l.a, m.a - l.a), 0.0); // overlap
53 }
54 point crosspointLL(const line& l, const line& m) {
55     double A = cross(l.b - l.a, m.b - m.a);
56     double B = cross(l.b - l.a, m.a - l.a);
57     if (eq(A, 0.0) and eq(B, 0.0)) return m.a; // overlap
58     assert(ne(A, 0.0)); // not parallel
59     return m.a - B / A * (m.b - m.a);
60 }
61 point proj(const line& l, point p) {
62     double t = dot(l.b - l.a, p - l.a) / norm(l.b - l.a);
63     return l.a + t * (l.b - l.a);
64 }
65 point reflection(const line& l, point p) { return 2.0 * proj(l, p) - p; }
66
67 // distances (for shortest path)
68
69 double distanceLP(const line& l, point p) { return abs(proj(l, p) - p); }
70 double distanceLL(const line& l, const line& m) {
71     return intersectLL(l, m) ? 0.0 : distanceLP(l, m.a);
72 }
73 double distanceLS(const line& l, const line& s) {
74     return intersectLS(l, s) ? 0.0 : min(distanceLP(l, s.a), distanceLP(l, s.b));
75 }
76 double distancePS(point p, const line& s) {
77     point h = proj(s, p);
78     return ccw(s.a, s.b, h) ? min(abs(s.a - p), abs(s.b - p)) : abs(h - p);
79 }
80 double distanceSS(const line& s, const line& t) {
81     if (intersectSS(s, t)) return 0.0;
82     return min(min(distancePS(s.a, t), distancePS(s.b, t)),
83                min(distancePS(t.a, s), distancePS(t.b, s)));
84 }
85
86 // circles
87
88 struct circle {
89     point o; double r;
90     circle(point o, double r) : o(o), r(r) {}
91 };
92
93 bool intersectCL(const circle& c, const line& l) {
94     return le(norm(proj(l, c.o) - c.o), c.r * c.r);
95 }
96 int intersectCS(const circle& c, const line& s) {
97     if (not intersectCL(c, s)) return 0;
98     double a = abs(s.a - c.o);
99     double b = abs(s.b - c.o);
100     if (lt(a, c.r) and lt(b, c.r)) return 0;
101     if (lt(a, c.r) or lt(b, c.r)) return 1;
102     return ccw(s.a, s.b, proj(s, c.o)) ? 0 : 2;
103 }
104 bool intersectCC(const circle& c, const circle& d) {
105     double dist = abs(d.o - c.o);
106     return le(abs(c.r - d.r), dist) and le(dist, c.r + d.r);
107 }
108 line crosspointCL(const circle& c, const line& l) {
109     point h = proj(l, c.o);
110     double a = sqrt(c.r * c.r - norm(h - c.o));
111     point d = a * (l.b - l.a) / abs(l.b - l.a);

```

```

112     return line(h - d, h + d);
113 }
114 line crosspointCC(const circle& c, const circle& d) {
115     double dist = abs(d.o - c.o), th = arg(d.o - c.o);
116     double ph = acos((c.r * c.r + dist * dist - d.r * d.r) / (2.0 * c.r * dist));
117     return line(c.o + polar(c.r, th - ph), c.o + polar(c.r, th + ph));
118 }
119
120 line tangent(const circle& c, double th) {
121     point h = c.o + polar(c.r, th);
122     point d = polar(c.r, th) * point(0, 1);
123     return line(h - d, h + d);
124 }
125 vector<line> common_tangents(const circle& c, const circle& d) {
126     vector<line> ret;
127     double dist = abs(d.o - c.o), th = arg(d.o - c.o);
128     if (abs(c.r - d.r) < dist) { // outer
129         double ph = acos((c.r - d.r) / dist);
130         ret.pb(tangent(c, th - ph));
131         ret.pb(tangent(c, th + ph));
132     }
133     if (abs(c.r + d.r) < dist) { // inner
134         double ph = acos((c.r + d.r) / dist);
135         ret.pb(tangent(c, th - ph));
136         ret.pb(tangent(c, th + ph));
137     }
138     return ret;
139 }
140 pair<circle, circle> tangent_circles(const line& l, const line& m, double r) {
141     double th = arg(m.b - m.a) - arg(l.b - l.a);
142     double ph = (arg(m.b - m.a) + arg(l.b - l.a)) / 2.0;
143     point p = crosspointLL(l, m);
144     point d = polar(r / sin(th / 2.0), ph);
145     return mp(circle(p - d, r), circle(p + d, r));
146 }
147 line bisector(point a, point b);
148 circle circum_circle(point a, point b, point c) {
149     point o = crosspointLL(bisector(a, b), bisector(a, c));
150     return circle(o, abs(a - o));
151 }
152
153 // polygons
154
155 typedef vector<point> polygon;
156
157 double area(const polygon& g) {
158     double ret = 0.0;
159     int j = g.size() - 1;
160     rep(i, g.size()) {
161         ret += cross(g[j], g[i]), j = i;
162     }
163     return ret / 2.0;
164 }
165 point centroid(const polygon& g) {
166     if (g.size() == 1) return g[0];
167     if (g.size() == 2) return (g[0] + g[1]) / 2.0;
168     point ret = 0.0;
169     int j = g.size() - 1;
170     rep(i, g.size()) {
171         ret += cross(g[j], g[i]) * (g[j] + g[i]), j = i;
172     }
173     return ret / area(g) / 6.0;
174 }
175 line bisector(point a, point b) {
176     point m = (a + b) / 2.0;
177     return line(m, m + (b - a) * point(0, 1));
178 }

```

```

179 polygon convex_cut(const polygon& g, const line& l) {
180     polygon ret;
181     int j = g.size() - 1;
182     rep(i, g.size()) {
183         if (ccw(l.a, l.b, g[j]) != -1) ret.pb(g[j]);
184         if (intersectLS(l, line(g[j], g[i]))) ret.pb(crosspointLL(l, line(g[j], g[i])));
185         j = i;
186     }
187     return ret;
188 }
189 polygon voronoi_cell(polygon g, const vector<point>& v, int k) {
190     rep(i, v.size()) if (i != k) {
191         g = convex_cut(g, bisector(v[i], v[k]));
192     }
193     return g;
194 }

```

## 5.1 凸包

```

1 #include "geometry.cpp"
2
3 namespace std {
4     bool operator <(const point& a, const point& b) {
5         return ne(real(a), real(b)) ? lt(real(a), real(b)) : lt(imag(a), imag(b));
6     }
7 }
8
9 polygon convex_hull(vector<point> v) {
10     const int n = v.size();
11     sort(all(v));
12     polygon ret(2 * n);
13     int k = 0;
14     for (int i = 0; i < n; ret[k++] = v[i++]) {
15         while (k >= 2 and ccw(ret[k - 2], ret[k - 1], v[i]) <= 0) --k;
16     }
17     for (int i = n - 2, t = k + 1; i >= 0; ret[k++] = v[i--]) {
18         while (k >= t and ccw(ret[k - 2], ret[k - 1], v[i]) <= 0) --k;
19     }
20     ret.resize(k - 1);
21     return ret;
22 }

```

## 6 データ構造

### 6.1 Union-Find 木

```

1 #include "macro.cpp"
2
3 class disjoint_set {
4     vector<int> p;
5     int root(int i) { return p[i] >= 0 ? p[i] = root(p[i]) : i; }
6 public:
7     disjoint_set(int n) : p(n, -1) {}
8     bool same(int i, int j) { return root(i) == root(j); }
9     int size(int i) { return -p[root(i)]; }
10    void merge(int i, int j) {
11        i = root(i), j = root(j);
12        if (i == j) return;
13        if (p[i] > p[j]) swap(i, j);

```

```

14     p[i] += p[j], p[j] = i;
15 }
16 };

```

## 6.2 赤黒木

```

1  template<class T> class rbtree {
2  public:
3      enum COL { BLACK, RED,};
4      struct node {
5          T val;
6          int color;
7          int rnk, size;
8          node *left, *right;
9
10         node(){}
11         node(T v) : val(v), color(BLACK), rnk(0), size(1) {
12             left = right = NULL;
13         }
14         node(node *l, node *r, int c) : color(c) {
15             left = l;
16             right = r;
17             update();
18         }
19         void update() {
20             rnk = max((left? left->rnk+(left->color==BLACK): 0),
21                       (right? right->rnk+(right->color==BLACK): 0));
22             size = (left? left->size: 0)+(right? right->size: 0)+(!left and !right);
23         }
24     };
25
26     node *root;
27
28     rbtree() { root = NULL;}
29     rbtree(T val) { root = new_node(val);}
30
31     node *new_node(T v) { return new node(v);}
32     node *new_node(node *l, node *r, int c) { return new node(l,r,c);}
33
34     node *right_rotate(node *v) {
35         node *w = v->left;
36         v->left = w->right;
37         w->right = v;
38         v->left->update();
39         v->update();
40         w->right->update();
41         v->color = RED;
42         w->color = BLACK;
43         return w;
44     }
45
46     node *left_rotate(node *v) {
47         node *w = v->right;
48         v->right = w->left;
49         w->left = v;
50         v->right->update();
51         v->update();
52         w->left->update();
53         v->color = RED;
54         w->color = BLACK;
55         return w;
56     }
57
58     node *merge_sub(node *u, node *v) {

```

```

59         if(u->rnk < v->rnk) {
60             node *w = merge_sub(u,v->left);
61             v->left = w;
62             v->update();
63             if(v->color == BLACK and w->color == RED and w->left->color == RED) {
64                 if(v->right->color == BLACK) return right_rotate(v);
65                 else {
66                     v->color = RED;
67                     v->right->color = BLACK;
68                     w->color = BLACK;
69                     return v;
70                 }
71             }
72             else return v;
73         }
74         else if(u->rnk > v->rnk) {
75             node *w = merge_sub(u->right,v);
76             u->right = w;
77             u->update();
78             if(u->color == BLACK and w->color == RED and w->right->color == RED) {
79                 if(u->left->color == BLACK) return left_rotate(u);
80                 else {
81                     u->color = RED;
82                     u->left->color = BLACK;
83                     w->color = BLACK;
84                     return u;
85                 }
86             }
87             else return u;
88         }
89         else return new_node(u,v,RED);
90     }
91
92     node *merge(node *u, node *v) {
93         if(!u) return v;
94         if(!v) return u;
95         u = merge_sub(u,v);
96         u->color = BLACK;
97         return u;
98     }
99
100     pair<node*,node*> split(node *v, int k) {
101         if(!k) return pair<node*,node*>(NULL,v);
102         if(k == v->size) return pair<node*,node*>(v,NULL);
103         if(k < v->left->size) {
104             auto p = split(v->left,k);
105             return pair<node*,node*>(p.first,merge(p.second,v->right));
106         }
107         else if(k > v->left->size) {
108             auto p = split(v->right,k-v->left->size);
109             return pair<node*,node*>(merge(v->left,p.first),p.second);
110         }
111         else return pair<node*,node*>(v->left,v->right);
112     }
113
114     // insert val at k
115     node *insert(T val, int k) { return insert(new_node(val),k);}
116     // insert tree v at k
117     node *insert(node *v, int k) {
118         auto p = split(root,k);
119         return root = merge(merge(p.first,v),p.second);
120     }
121
122     // delete at k
123     node *erase(int k) {
124         auto p = split(root,k+1);
125         return root = merge(split(p.first,k).first, p.second);

```

```

126     }
127
128     node *build(const vector<T> &vs) {
129         if(!vs.size()) return NULL;
130         if((int)vs.size() == 1) return new_node(vs[0]);
131         int m = vs.size()/2;
132         return merge(build(vector<T>(begin(vs),begin(vs)+m)),
133                     build(vector<T>(begin(vs)+m,end(vs))));
134     }
135
136     int size() { return root->size;}
137
138     void get(vector<T> &vs) { get(root,vs);}
139     void get(node *v, vector<T> &vs) {
140         if(!v->left and !v->right) vs.push_back(v->val);
141         else {
142             if(v->left) get(v->left,vs);
143             if(v->right) get(v->right,vs);
144         }
145     }
146
147     node *push_back(T val) {
148         node *v = new_node(val);
149         return root = merge(root,v);
150     }
151 };

```

### 6.3 永続赤黒木

```

1 //const int MAX = 15000000, BOUND = 14000000;
2 template<class T> class prbtree {
3 public:
4     enum COL { BLACK, RED,};
5     struct node {
6         T val;
7         int color;
8         int rnk, size;
9         node *left, *right;
10
11         node(){}
12         node(T v) : val(v), color(BLACK), rnk(0), size(1) {
13             left = right = NULL;
14         }
15         node(node *l, node *r, int c) : color(c) {
16             left = l;
17             right = r;
18             rnk = max((l? l->rnk+(l->color==BLACK): 0),
19                     (r? r->rnk+(r->color==BLACK): 0));
20             size = !l and !r? 1: !l? r->size: !r? r->size: l->size+r->size;
21         }
22     };
23
24     node *root;
25     //     node nodes[MAX];
26     //     int called;
27
28     prbtree() {
29         root = NULL;
30         // called = 0;
31     }
32
33     prbtree(T val) {
34         root = new_node(val);
35         // called = 0;

```

```

36     }
37
38     // node *new_node(T v) { return &(nodes[called++] = node(v));}
39     // node *new_node(node *l, node *r, int c) { return &(nodes[called++] = node(l,r,c
40     ));}
41     node *new_node(T v) { return new node(v);}
42     node *new_node(node *l, node *r, int c) { return new node(l,r,c);}
43
44     node *merge_sub(node *u, node *v) {
45         if(u->rnk < v->rnk) {
46             node *w = merge_sub(u,v->left);
47             if(v->color == BLACK and w->color == RED and w->left->color == RED){
48                 if(v->right->color == BLACK) return new_node(w->left,new_node(w->right,
49                     v->right,RED),BLACK);
50                 else return new_node(new_node(w->left,w->right,BLACK),new_node(v->right
51                     ->left,v->right->right,BLACK),RED);
52             }
53             else return new_node(w,v->right,v->color);
54         }
55         else if(u->rnk > v->rnk) {
56             node *w = merge_sub(u->right,v);
57             if(u->color == BLACK and w->color == RED and w->right->color == RED){
58                 if(u->left->color == BLACK) return new_node(new_node(u->left,w->left,
59                     RED),w->right,BLACK);
60                 else return new_node(new_node(u->left->left,u->left->right,BLACK),
61                     new_node(w->left,w->right,BLACK),RED);
62             }
63             else return new_node(u->left,w,u->color);
64         }
65         else return new_node(u,v,RED);
66     }
67
68     node *merge(node *u, node *v) {
69         if(!u) return v;
70         if(!v) return u;
71         u = merge_sub(u,v);
72         if(u->color == RED) return new_node(u->left,u->right,BLACK);
73         return u;
74     }
75
76     pair<node*,node*> split(node *v, int k) {
77         if(!k) return pair<node*,node*>(NULL,v);
78         if(k == v->size) return pair<node*,node*>(v,NULL);
79         if(k < v->left->size) {
80             auto p = split(v->left,k);
81             return pair<node*,node*>(p.first,merge(p.second,v->right));
82         }
83         else if(k > v->left->size) {
84             auto p = split(v->right,k-v->left->size);
85             return pair<node*,node*>(merge(v->left,p.first),p.second);
86         }
87         else return pair<node*,node*>(v->left,v->right);
88     }
89
90     node *build(const vector<T> &vs) {
91         if(!vs.size()) return NULL;
92         if((int)vs.size() == 1) return new_node(vs[0]);
93         int m = vs.size()/2;
94         return merge(build(vector<T>(begin(vs),begin(vs)+m)), build(vector<T>(begin(vs)+
95             m,end(vs))));
96     }
97
98     int size() { return root->size;}
99
100     void get(vector<T> &vs) { get(root,vs);}
101     void get(node *v, vector<T> &vs) {
102         if(!v->left and !v->right) vs.push_back(v->val);

```

```

97     else {
98         if(v->left) get(v->left,vs);
99         if(v->right) get(v->right,vs);
100     }
101 }
102
103 node *push_back(T val) {
104     node *v = new_node(val);
105     return root = merge(root,v);
106 }
107
108 // insert leaf at k
109 node *insert(int k, T val) {
110     return insert(new_node(val), k);
111 }
112
113 // insert tree v at k
114 node *insert(node *v, int k) {
115     auto p = split(root,k);
116     return root = merge(merge(p.first,v),p.second);
117 }
118
119 // copy [l,r)
120 node *copy(int l, int r) {
121     return split(split(root, l).second, r-l).first;
122 }
123 // copy and insert [l,r) at k
124 node *copy_paste(int l, int r, int k) {
125     return insert(copy(l,r),k);
126 }
127 };

```

## 6.4 wavelet 行列

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  class fidict
5  {
6      typedef unsigned long long ull;
7      vector<ull> bs;
8      vector<int> sum[2];
9      int N, M;
10     int popcount(int r) const { return sum[1][r/64]+__builtin_popcountll(bs[r/64]&((1ULL
11         <<r%64)-1ULL));}
12     int popcount(int l, int r) const { return popcount(r)-popcount(l);}
13
14     int _select(ull x, int i) const {
15         ull a, b, c, d; int t, s;
16         a = (x & 0x5555555555555555ULL) + ((x >> 1) & 0x5555555555555555ULL);
17         b = (a & 0x3333333333333333ULL) + ((a >> 2) & 0x3333333333333333ULL);
18         c = (b & 0x0f0f0f0f0f0f0f0fULL) + ((b >> 4) & 0x0f0f0f0f0f0f0f0fULL);
19         d = (c & 0x00ff00ff00ff00ffULL) + ((c >> 8) & 0x00ff00ff00ff00ffULL);
20         t = (d & 0xffff) + ((d >> 16) & 0xffff);
21         s = 0;
22         s += ((t - i) & 256) >> 3; i -= t & ((t - i) >> 8);
23         t = (d >> s) & 0x1f;
24         s += ((t - i) & 256) >> 4; i -= t & ((t - i) >> 8);
25         t = (c >> s) & 0xf;
26         s += ((t - i) & 256) >> 5; i -= t & ((t - i) >> 8);
27         t = (b >> s) & 0x7;
28         s += ((t - i) & 256) >> 6; i -= t & ((t - i) >> 8);
29         t = (a >> s) & 0x3;
30         s += ((t - i) & 256) >> 7; i -= t & ((t - i) >> 8);

```

```

30     t = (x >> s) & 0x1;
31     s += ((t - i) & 256) >> 8;
32     return s;
33 }
34 public:
35 fidict(){
36     fidict(const vector<bool> &a) {
37         N = a.size(); M = (N+63)/64;
38         bs.assign(M,0);
39         sum[0].assign(M+1,0);
40         sum[1].assign(M+1,0);
41         for(int i = 0; i < N; ++i) {
42             ull k = ull(a[i])<<(i%64);
43             bs[i/64] |= k;
44             sum[k>0][i/64+1]++;
45         }
46         for (int i = 0; i < M; ++i) {
47             sum[0][i+1] += sum[0][i];
48             sum[1][i+1] += sum[1][i];
49         }
50     }
51
52     // number of 1 in [0,r), O(1)
53     int rank(bool val, int r) const { return val? popcount(r): r-popcount(r);}
54     int rank(bool val, int l, int r) const { return rank(val,r)-rank(val,l);}
55
56     // index of i th val; 0-indexed, O(log N)
57     int select(bool val, int i) {
58         if(i >= sum[val].back() or i < 0) return -1;
59         int j = lower_bound(begin(sum[val]),end(sum[val]),++i)-begin(sum[val])-1;
60         i -= sum[val][j];
61         return _select(val?bs[j]:~bs[j],i)+j*64;
62     }
63     int select(bool val, int i, int l) { return select(val,i+rank(val,l));}
64     bool operator[](const int &i) { return bs[i/64]&(1ULL<<(i%64));}
65 };
66
67 // T is a kind of integer
68 template <class T> class wavelet
69 {
70     typedef unsigned long long ull;
71     int N, D; // length, depth
72     T M; // max value
73     vector<T> seq;
74     vector<int> zeros;
75     vector<fidict> B;
76
77     void build(vector<T> f) {
78         vector<T> l, r;
79         for (int d = 0; d < D; d++) {
80             vector<bool> b;
81             for(auto &e: f) {
82                 bool k = (e>>(D-d-1))&1;
83                 if(k) r.push_back(e);
84                 else l.push_back(e);
85                 b.push_back(k);
86             }
87             B.push_back(fidict(b));
88             zeros.push_back(l.size());
89             swap(l,f);
90             f.insert(end(f),begin(r),end(r));
91             l.clear(); r.clear();
92         }
93     }
94     // structure topk_node is for topk
95     struct topk_node {
96         T val;

```

```

97     int l, r, d;
98     topk_node(T val, int l, int r, int d)
99         : val(val), l(l), r(r), d(d) {}
100     bool operator<(const topk_node &v) const { return r-l < v.r-v.l; }
101 };
102 // rec for range_maxk
103 void rmk_rec(int l, int r, int d, int &k, T val, vector<T> &vs) {
104     if(l==r) return;
105     if(d == D) {
106         while(l++ < r and k > 0) vs.push_back(val), k--;
107         return;
108     }
109     int lc = B[d].rank(1,l), rc = B[d].rank(1,r);
110     if(vs.size()) {
111         rmk_rec(lc+zeros[d],rc+zeros[d],d+1,k,val|(1ULL<<(D-d-1)),vs);
112         rmk_rec(l-lc,r-rc,d+1,k,val,vs);
113     }
114     else {
115         if(rc-lc > 0) rmk_rec(lc+zeros[d],rc+zeros[d],d+1,k,val|(1ULL<<(D-d-1)),vs);
116         if(vs.size() and k > 0) rmk_rec(l-lc,r-rc,d+1,k,val,vs);
117     }
118 }
119 // rec for range_freq
120 int rf_rec(int l, int r, int d, T val, T lb, T ub) {
121     if(l==r) return 0;
122     if(d == D) return (lb<=val and val<ub? r-l: 0);
123     T nv = val|(1LL<<(D-d-1)), nnv = nv|(((1LL<<(D-d-1))-1));
124     if(ub <= val or nnv < lb) return 0;
125     if(lb <= val and nnv < ub) return r-l;
126     int lc = B[d].rank(1,l), rc = B[d].rank(1,r);
127     return rf_rec(l-lc,r-rc,d+1,val,lb,ub)+rf_rec(lc+zeros[d],rc+zeros[d],d+1,nv,lb,
128         ub);
129 }
130 // rec for range_list
131 void rl_rec(int l, int r, int d, T val, T lb, T ub, vector<pair<T,int>> &vs) {
132     if(l==r) return;
133     if(d == D) {
134         if(val < lb or ub <= val) return;
135         if(r-l) vs.push_back(make_pair(val,r-l));
136         return;
137     }
138     T nv = val|(1LL<<(D-d-1)), nnv = nv|(((1LL<<(D-d-1))-1));
139     if(nnv < lb or ub <= val) return;
140     int lc = B[d].rank(1,l), rc = B[d].rank(1,r);
141     rl_rec(l-lc,r-rc,d+1,val,lb,ub,vs);
142     rl_rec(lc+zeros[d],rc+zeros[d],d+1,nv,lb,ub,vs);
143 }
144 // rec for range_exist
145 bool re_rec(int l, int r, int d, T val, T lb, T ub) {
146     if(l==r) return 0;
147     if(d == D) return (lb<=val and val<ub? r-l: 0);
148     T nv = val|(1LL<<(D-d-1)), nnv = nv|(((1LL<<(D-d-1))-1));
149     if(nnv < lb or ub <= val) return 0;
150     if(lb <= val and nnv < ub) return 1;
151     int lc = B[d].rank(1,l), rc = B[d].rank(1,r);
152     return re_rec(l-lc,r-rc,d+1,val,lb,ub) || re_rec(lc+zeros[d],rc+zeros[d],d+1,nv,
153         lb,ub);
154 }
155 public:
156     wavelet(const vector<T> &f) {
157         N = f.size();
158         M = *max_element(begin(f),end(f));
159         D = 64-__builtin_clzll(M);
160         seq = f;
161         build(f);

```

```

162     }
163
164     // number of val, O(D)
165     int rank(T val, int l, int r) {
166         for (int d = 0; d < D; d++) {
167             bool b = (val>>(D-d-1))&1;
168             l = B[d].rank(b,l)+b*zeros[d];
169             r = B[d].rank(b,r)+b*zeros[d];
170         }
171         return r-l;
172     }
173     int rank(T val, int r) { return rank(val,0,r); }
174
175     // index of val, O(D log D)
176     int select(T val, int i) {
177         int ls[64], rs[64], l = 0, r = N;
178         for (int d = 0; d < D; d++) {
179             ls[d] = l; rs[d] = r;
180             bool b = (val>>(D-d-1))&1;
181             l = B[d].rank(b,l)+b*zeros[d];
182             r = B[d].rank(b,r)+b*zeros[d];
183         }
184         for (int d = D-1; d >= 0; d--) {
185             bool b = (val>>(D-d-1))&1;
186             i = B[d].select(b,i,ls[d]);
187             if(i >= rs[d] or i < 0) return -1;
188             i -= ls[d];
189         }
190         return i;
191     }
192     int select(T val, int i, int l) { return select(val,i+rank(val,l)); }
193
194     T access(int i) { return seq[i]; }
195     T operator[](int i) { return seq[i]; }
196
197     // ith large val in [l,r), O(D)
198     T quantile(int i, int l, int r) {
199         T ret = 0;
200         for (int d = 0; d < D; d++) {
201             int lc = B[d].rank(1,l), rc = B[d].rank(1,r);
202             if(rc-lc >= i) {
203                 l = lc+zeros[d];
204                 r = rc+zeros[d];
205                 ret |= 1ULL<<(D-d-1);
206             }
207             else {
208                 i -= rc-lc;
209                 l -= lc;
210                 r -= rc;
211             }
212         }
213         return ret;
214     }
215     T maximum(int l, int r) { return quantile(0,l,r); }
216     T minimum(int l, int r) { return quantile(r-l-1,l,r); }
217
218     // freq top k in [l,r), O(D^3)?
219     vector<T> topk(int l, int r, int k) {
220         priority_queue<topk_node> q; // (freq,((l,r),d))
221         vector<T> ret;
222         q.push(topk_node(0,l,r,0));
223         while(!q.empty()) {
224             topk_node v = q.top(); q.pop();
225             if(v.d == D) {
226                 ret.push_back(v.val);
227                 if(--k) break;
228             }

```

```

229         int lc = B[v.d].rank(1,lc), rc = B[v.d].rank(1,rc);
230         q.push(topk_node(v.val|(1ULL<<v.d), lc+zeros[v.d], rc+zeros[v.d], v.d+1));
231         q.push(topk_node(v.val, 1-lc, r-rc, v.d+1));
232     }
233     return ret;
234 }
235
236 // k most large vals
237 vector<T> range_maxk(int l, int r, int k) {
238     vector<T> ret;
239     rmk_rec(l,r,0,k,0,ret);
240     return ret;
241 }
242
243 // number of [lb,ub) elements in [l,r), O(DK) K = freq
244 int range_freq(int l, int r, T lb, T ub) { return rf_rec(l,r,0,0,lb,ub);}
245
246 // list of elements and freq in [lb,ub) in [l,r) O(DK) size of list(<= r-l)
247 vector<pair<T,int>> range_list(int l, int r, T lb, T ub) {
248     vector<pair<T,int>> ret;
249     rl_rec(l,r,0,0,lb,ub,ret);
250     return ret;
251 }
252
253 // list of elements in rectangle [(l,lb),(r,ub)), O(DK log D) K = size of list(<= r-l)
254 // selectを使わなくてもできる?
255 vector<pair<int,T>> range_rect(int l, int r, T lb, T ub) {
256     vector<pair<int,T>> ret;
257     vector<pair<T,int>> vs = range_list(l,r,lb,ub);
258     for(auto &p: vs)
259         for (int i = 0; i < p.second; i++)
260             ret.push_back(make_pair(select(p.first,i,l,r),p.first));
261     return ret;
262 }
263
264 bool range_exist(int l, int r, T lb, T ub) {
265     return re_rec(l,r,0,0,lb,ub);
266 }
267 };

```