

## Contents

<b>1</b>	<b>準備</b>	<b>1</b>			
1.1	Caps Lock を Control に変更	1			
1.2	init.el	1			
1.3	tpl.cpp	1			
1.4	get input	2			
1.5	alias	2			
<b>2</b>	<b>文字列</b>	<b>2</b>			
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			
2.3	回文長 (Manacher)	3			
<b>3</b>	<b>グラフ</b>	<b>3</b>			
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			
3.1.4	無向中国人郵便配達問題	4			
3.1.5	全点対間最短路 (Johnson)	4			
3.1.6	無向グラフの全域最小カット	5			
3.2	フロー	5			
3.2.1	最大流	5			
3.2.2	二部マッチング	5			
3.2.3	最小費用流	6			
3.2.4	Gomory-Hu 木	6			
3.3	木	7			
3.3.1	木の直径	7			
3.3.2	最小全域木	7			
3.3.3	最小全域有向木	7			
3.3.4	最小シュタイナー木	8			
3.3.5	木の同型性判定	8			
3.3.6	HL 分解	8			
3.4	彩色数	9			
3.4.1	包除原理	9			
3.4.2	極大独立集合	9			
3.5		10			
<b>4</b>	<b>数学</b>	<b>10</b>			
4.1	整数	10			
4.1.1	剰余	10			
4.1.2	剰余積	10			
4.1.3	離散対数問題	10			
4.1.4	カタラン数	10			
4.1.5	乱数 (xor shift)	11			
4.1.6	確率的素数判定 (Miller-Rabin 法)	11			
4.2	多項式	11			
4.2.1	FFT(complex)	11			
4.2.2	FFT(modulo)	11			
4.2.3	積 (FMT)	12			
4.2.4	逆元 (FMT)	12			
4.2.5	平方根 (FMT)	12			
4.3	行列	12			
4.3.1	線形方程式の解 (Givens 消去法)	13			
4.4	割り当て問題	13			
4.4.1	ハンガリアン法	13			
<b>5</b>	<b>幾何</b>	<b>13</b>			
5.1	点	13			
5.2	直線と線分	14			
5.3	円	14			
5.4	多角形	14			
5.4.1	凸包	15			
5.4.2	最近点对	15			
5.4.3	点-多角形包含判定	15			
5.4.4	凸多角形の共通部分	15			
5.4.5	凸多角形の直径	16			
5.4.6	ドロネー三角形分割 (逐次添加法)	16			
<b>6</b>	<b>データ構造</b>	<b>16</b>			
6.1	Union-Find 木	16			
6.2	Meldable Heap	17			
6.3	Binary-Indexed-Tree	17			
6.4	Segment Tree	17			
6.5	Sparse table	17			
6.6	赤黒木	17			
6.7	永続赤黒木	19			
6.8	wavelet 行列	20			
6.8.1	完備辞書	20			
6.8.2	wavelet 行列	20			
<b>7</b>	<b>その他</b>	<b>22</b>			
7.1	ビジュアライザ	22			

## 1 準備

### 1.1 Caps Lock を Control に変更

2 つ

#### 1. 変更

```
1 setxkbmap -option ctrl:nocaps;
```

元に戻す

```
1 setxkbmap -option;
```

#### 2. 上でダメな場合

```
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 -Wl,-stack_size,64000000';
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[0];
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
41                 (tmp));
42         now->next[i]->match = tmp;
43         q.push(now->next[i]);
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 // verify
2 // sa: http://www.spoj.com/problems/SARRAY/
3 // lcp: http://www.spoj.com/problems/SUBLEX/
4
5 int n, k;
6 vector<int> rnk, tmp, sa, lcp;
7
8 bool compare_sa(int i, int j) {
9     if(rnk[i] != rnk[j]) return rnk[i] < rnk[j];
10    else {
11        int ri = i + k <= n ? rnk[i+k] : -1;
12        int rj = j + k <= n ? rnk[j+k] : -1;
13        return ri < rj;
14    }
15 }
16
17 void construct_sa(const string &s) {
18     n = s.size();
19     rnk.assign(n+1, 0);
20     tmp.assign(n+1, 0);
21     sa.assign(n+1, 0);
22     lcp.assign(n+1, 0);
23     for(int i = 0; i <= n; i++) {
24         sa[i] = i;
25         rnk[i] = i < n ? s[i] : -1;
26     }
27     for(k = 1; k <= n; k*=2) {
28         sort(sa.begin(), sa.end(), compare_sa);
29         tmp[sa[0]] = 0;
30         for(int i = 1; i <= n; i++) {
31             tmp[sa[i]] = tmp[sa[i-1]] + (compare_sa(sa[i-1], sa[i]) ? 1 : 0);
32         }
33         for(int i = 0; i <= n; i++) {
34             rnk[i] = tmp[i];
35         }
36     }
37 }

```

```

36     }
37 }
38
39 void construct_lcp(const string &s) {
40     for(int i = 0; i <= n; i++) rnk[sa[i]] = i;
41     int h = 0;
42     lcp[0] = 0;
43     for(int i = 0; i < n; i++) {
44         int j = sa[rnk[i] - 1];
45         if(h > 0) h--;
46         for(; j + h < n && i + h < n; h++) {
47             if(s[j+h] != s[i+h]) break;
48         }
49         lcp[rnk[i] - 1] = h;
50     }
51 }

```

## 2.3 回文長 (Manacher)

$O(N)$

各文字を中心とした時の回文の最長の半径。

偶数長の回文はダミーを挟むことで求められている。

```

1 vector<int> manacher(const string &s) {
2     int n = s.size()*2;
3     vector<int> rad.assign(n, 0);
4     for (int i = 0, j = 0, k; i < n; i += k, j = max(j-k, 0)) {
5         while (i-j >= 0 && i+j+1 < n && s[(i-j)/2] == s[(i+j+1)/2]) ++j;
6         rad[i] = j;
7         for (k = 1; i-k >= 0 && rad[i]-k >= 0 && rad[i-k] != rad[i]-k; ++k)
8             rad[i+k] = min(rad[i-k], rad[i]-k);
9     }
10    return rad;
11 }

```

## 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 };
```

### 3.1.4 無向中国人郵便配達問題

$O(m \log n + o^2 2^o)$ ,  $-O2$  で  $o \leq 18$  程度が限界

```
1 long chinesePostman(const graph &g) {
2     long total = 0;
3     vector<int> odds;
4     rep(u, g.size()) {
5         for(auto &e: g[u]) total += e.w;
6         if (g[u].size() % 2) odds.push_back(u);
7     }
8     total /= 2;
9     int n = odds.size(), N = 1 << n;
10    int w[n][n]; // make odd vertices graph
11    rep(u,n) {
12        int s = odds[u]; // dijkstra's shortest path
13        vector<int> dist(g.size(), 1e9); dist[s] = 0;
14        vector<int> prev(g.size(), -2);
15        priority_queue<edge> Q;
16        Q.push( edge(-1, s, 0) );
17        while (!Q.empty()) {
18            edge e = Q.top(); Q.pop();
19            if (prev[e.to] != -2) continue;
20            prev[e.to] = e.src;
21            for(auto &f: g[e.to]) {
22                if (dist[f->to] > e.w+f->w) {
23                    dist[f->to] = e.w+f->w;
24                    Q.push(edge(f->src, f->to, e.w+f->w));
25                }
26            }
27        }
28        rep(v,n) w[u][v] = dist[odds[v]];
29    }
30    long best[N]; // DP for general matching
31    rep(S,N) best[S] = INF;
32    best[0] = 0;
33
34    for (int S = 0; S < N; ++S)
35        for (int i = 0; i < n; ++i)
36            if (!(S&(1<<i)))
37                for (int j = i+1; j < n; ++j)
38                    if (!(S&(1<<j)))
39                        best[S|(1<<i)|(1<<j)] = min(best[S|(1<<i)|(1<<j)], best[S]+w[i][j]);
40
41    return total + best[N-1];
42 }
```

### 3.1.5 全点对間最短路 (Johnson)

$O(\max(V, E) \log V, V^2)$

```
1 bool shortest_path(const graph &g, vector<vector<int>> &dist, vector<vector<int>> &prev) {
2     int n = g.size();
3     vector<int> h(n+1);
4     rep(k,n) rep(i,n) for(auto &e: g[i]) {
5         if (h[e.to] > h[e.from] + e->w) {
6             h[e.to] = h[e.from] + e->w;
7             if (k == n-1) return false; // negative cycle
8         }
9     }
10    dist.assign(n, vector<int>(n, 1e9));
11    prev.assign(n, vector<int>(n, -2));
12    rep(s, n) {
```

```
13        priority_queue<edge> q;
14        q.push(edge(s, s, 0));
15        while (!q.empty()) {
16            edge e = q.top(); q.pop();
17            if (prev[s][e.dst] != -2) continue;
18            prev[s][e.to] = e.from;
19            for(auto &f: g[e.to]) {
20                if (dist[s][f.to] > e.w + f->w) {
21                    dist[s][f.to] = e.w + f->w;
22                    q.push(edge(f->from, f.to, e.w + f->w));
23                }
24            }
25        }
26        rep(u, n) dist[s][u] += h[u] - h[s];
27    }
28 }
29
30 vector<int> build_path(const vector<vector<int>> &prev, int s, int t) {
31     vector<int> path;
32     for (int u = t; u >= 0; u = prev[s][u])
33         path.push_back(u);
34     reverse(begin(path), end(path));
35     return path;
36 }
```

### 3.1.6 無向グラフの全域最小カット

$O(V^3)$

```
1 int minimum_cut(const graph &g) {
2     int n = g.size();
3     vector< vector<int>> > h(n, vector<int>(n)); // make adj. matrix
4     rep(u,n) for(auto &e: g[u]) h[e.src][e.dst] += e.weight;
5     vector<int> V(n); rep(u, n) V[u] = u;
6
7     int cut = 1e9;
8     for(int m = n; m > 1; m--) {
9         vector<int> ws(m, 0);
10        int u, v;
11        int w;
12        rep(k, m) {
13            u = v; v = max_element(ws.begin(), ws.end())-ws.begin();
14            w = ws[v]; ws[v] = -1;
15            rep(i, m) if (ws[i] >= 0) ws[i] += h[V[v]][V[i]];
16        }
17        rep(i, m) {
18            h[V[i]][V[u]] += h[V[i]][V[v]];
19            h[V[u]][V[i]] += h[V[v]][V[i]];
20        }
21        V.erase(V.begin()+v);
22        cut = min(cut, w);
23    }
24    return cut;
25 }
```

## 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

$O(FE \log V)$

```

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     }
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 }

```

### 3.2.4 Gomory-Hu 木

$O(VMAXFLOW)$

```

1  #define RESIDUE(s,t) (capacity[s][t]-flow[s][t])
2  graph cutTree(const graph &g) {
3      int n = g.size();
4      Matrix capacity(n, Array(n)), flow(n, Array(n));
5      rep(u,n) for(auto &e: g[u]) capacity[e.from][e.to] += e.w;
6
7      vector<int> p(n), prev;
8      vector<int> w(n);
9      for (int s = 1; s < n; ++s) {
10         int t = p[s]; // max-flow(s, t)
11         rep(i,n) rep(j,n) flow[i][j] = 0;
12         int total = 0;
13         while (1) {
14             queue<int> Q; Q.push(s);
15             prev.assign(n, -1); prev[s] = s;
16             while (!Q.empty() && prev[t] < 0) {
17                 int u = Q.front(); Q.pop();
18                 for(auto &e: g[u]) if (prev[e.to] < 0 && RESIDUE(u, e.to) > 0) {
19                     prev[e.to] = u;
20                     Q.push(e.to);
21                 }
22             }
23             if (prev[t] < 0) goto esc;
24             int inc = 1e9;
25             for (int j = t; prev[j] != j; j = prev[j])
26                 inc = min(inc, RESIDUE(prev[j], j));
27             for (int j = t; prev[j] != j; j = prev[j])
28                 flow[prev[j]][j] += inc, flow[j][prev[j]] -= inc;
29             total += inc;
30         }
31     esc: w[s] = total; // make tree
32     rep(u, n) if (u != s && prev[u] != -1 && p[u] == t)
33         p[u] = s;
34     if (prev[p[t]] != -1)
35         p[s] = p[t], p[t] = s, w[s] = w[t], w[t] = total;

```

```

36     }
37     graph T(n); // (s, p[s]) is a tree edge of weight w[s]
38     rep(s, n) if (s != p[s]) {
39         T[ s ].push_back( Edge(s, p[s], w[s]) );
40         T[p[s]].push_back( Edge(p[s], s, w[s]) );
41     }
42     return T;
43 }
44
45 // Gomory-Hu tree を用いた最大流  $O(n)$ 
46 int max_flow(const graph &T, int u, int t, int p = -1, int w = 1e9) {
47     if (u == t) return w;
48     int d = 1e9;
49     for(auto &e: T[u]) if (e.to != p)
50         d = min(d, max_flow(T, e.to, t, u, min(w, e.w)));
51     return d;
52 }

```

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

```

```

36     q.pop();
37     if (done[e.v]) continue;
38     done[e.v] = 1;
39     if (e.u >= 0) {
40         T[e.u].eb(e.v, e.w);
41         T[e.v].eb(e.u, e.w);
42     }
43     rep(i, n) if (not done[i]) {
44         q.emplace(e.v, i, A[e.v][i]);
45     }
46 }
47 return T;
48 }

```

### 3.3.3 最小全域有向木

$O(VE)$

```

1 void visit(Graph &h, int v, int s, int r,
2     vector<int> &no, vector< vector<int> > &comp,
3     vector<int> &prev, vector< vector<int> > &next, vector<int> &mcost,
4     vector<int> &mark, int &cost, bool &found) {
5     const int n = h.size();
6     if (mark[v]) {
7         vector<int> temp = no;
8         found = true;
9         do {
10             cost += mcost[v];
11             v = prev[v];
12             if (v != s) {
13                 while (comp[v].size() > 0) {
14                     no[comp[v].back()] = s;
15                     comp[s].push_back(comp[v].back());
16                     comp[v].pop_back();
17                 }
18             }
19             while (v != s);
20             for(auto &j: comp[s]) if (j != r) for(auto &e: h[j])
21                 if (no[e.from] != s) e.w -= mcost[temp[j]];
22         }
23         mark[v] = true;
24         for(auto &i: next[v]) if (no[i] != no[v] && prev[no[i]] == v)
25             if (!mark[no[i]] || i == s)
26                 visit(h, i, s, r, no, comp, prev, next, mcost, mark, cost, found);
27     }
28     int minimum_spanning_arborescence(const graph &g, int r) {
29         const int n = g.size();
30         graph h(n);
31         rep(u,n) for(auto &e: g[u]) h[e.to].push_back(e);
32
33         vector<int> no(n);
34         vector< vector<int> > comp(n);
35         rep(u, n) comp[u].push_back(no[u] = u);
36
37         for (int cost = 0; ; ) {
38             vector<int> prev(n, -1);
39             vector<int> mcost(n, INF);
40
41             rep(j,n) if (j != r) for(auto &e: g[j])
42                 if (no[e.from] != no[j])
43                     if (e.w < mcost[no[j]])
44                         mcost[no[j]] = e.w, prev[no[j]] = no[e.from];
45
46             vector< vector<int> > next(n);
47             rep(u,n) if (prev[u] >= 0)

```

```

48         next[prev[u]].push_back(u);
49
50         bool stop = true;
51         vector<int> mark(n);
52         rep(u,n) if (u != r && !mark[u] && !comp[u].empty()) {
53             bool found = false;
54             visit(h, u, u, r, no, comp, prev, next, mcost, mark, cost, found);
55             if (found) stop = false;
56         }
57         if (stop) {
58             rep(u,n) if (prev[u] >= 0) cost += mcost[u];
59             return cost;
60         }
61     }
62 }

```

### 3.3.4 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     repi(S,1,1<<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 }

```

### 3.3.5 木の同型性判定

順序付き  $O(n)$

順序なし  $O(n \log n)$

```

1 // ordered
2 struct node {
3     vector<node*> child;
4 };
5 bool otreeIsomorphism(node *n, node *m) {
6     if (n->child.size() != m->child.size()) return false;

```



```

7     rep(i, n->child.size())
8         if (!otreeIsomorphism(n->child[i], m->child[i])) return false;
9     return true;
10 }
11
12 // not ordered
13 struct node {
14     vector<node*> child;
15     vector<int> code;
16 };
17 void code(node *n) {
18     int size = 1;
19     vector< pair<vector<int>, int> > codes;
20     rep(i, n->child.size()) {
21         code(n->child[i]);
22         codes.push_back( make_pair(n->child[i]->code, i) );
23         size += codes[i].first[0];
24     }
25     sort(codes.rbegin(), codes.rend()); // !reverse
26     n->code.push_back(size);
27     for (int i = 0; i < n->child.size(); ++i) {
28         swap(n->child[i], n->child[ codes[i].second ]);
29         n->code.insert(n->code.end(),
30                     codes[i].first.begin(), codes[i].first.end());
31     }
32 }
33 bool utreeIsomorphism(node *n, node *m) {
34     code(n); code(m); return n->code == m->code;
35 }

```

### 3.3.6 HL 分解

```

1 namespace HLD {
2     const int N = 200010;
3     vector<vector<int>> chains, childs;
4     int V, dep[N], par[N], heavy[N], head[N], chain[N], id[N], size[N], q[N];
5
6     void calc_heavy() {
7         int root = -1;
8         childs.assign(V, vector<int>());
9         for(int v = 0; v < V; v++) {
10             size[v] = 0;
11             heavy[v] = -1;
12             if(par[v] < 0) root = v;
13             else childs[par[v]].push_back(v);
14         }
15         int l = 0, r = 0;
16         q[r++] = root;
17         while(l < r) {
18             int v = q[l++];
19             for(auto &w: childs[v]) {
20                 if(w == par[v]) continue;
21                 dep[w] = dep[v] + 1;
22                 q[r++] = w;
23             }
24         }
25         reverse(q, q+V);
26         for(int i = 1; i < V; i++) {
27             int v = q[i], &u = par[v];
28             size[u] += ++size[v];
29             if(heavy[u] == -1 or size[v] > size[heavy[u]]) heavy[u] = v;
30         }
31     }
32     void calc_chain() {

```

```

33     chains.clear();
34     int idx = 0;
35     for (int v = 0; v < V; v++) {
36         if(par[v] < 0 or heavy[par[v]] != v) {
37             chains.push_back(vector<int>());
38             for (int w = v; w != -1; w = heavy[w]) {
39                 chain[w] = idx;
40                 head[w] = v;
41                 id[w] = chains.back().size();
42                 chains.back().push_back(w);
43             }
44             idx++;
45         }
46     }
47 }
48 void make_par(const vector<vector<int>> &g, int root = 0) {
49     memset(par, -1, sizeof(par));
50     par[root] = 0;
51     int l = 0, r = 0;
52     q[r++] = root;
53     while(l < r) {
54         int v = q[l++];
55         for(const int &w: g[v]) if(par[w] < 0) q[r++] = w, par[w] = v;
56     }
57     par[root] = -1;
58 }
59 void build(const vector<vector<int>> &g, int root = 0) {
60     V = g.size();
61     make_par(g, root);
62     calc_heavy();
63     calc_chain();
64 }
65 int lca(int u, int v) {
66     while (chain[u] != chain[v]) {
67         if (dep[head[u]] > dep[head[v]]) swap(u, v);
68         v = par[head[v]];
69     }
70     return dep[u] < dep[v]? u: v;
71 }
72 }

```

## 3.4 彩色数

### 3.4.1 包除原理

$O(2^V V)$

$N[i] := i$  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 }

```

### 3.4.2 極大独立集合

```

1  typedef vector<vector<int>> graph;
2  class maximal_indsets {
3      const int n;
4      const graph& G;
5      vector<vector<int>> ret;
6      vector<int> cur, exists, deg, block;
7      void erase(int v) {
8          if (exists[v]) {
9              exists[v] = false;
10             for (int nv : G[v]) --deg[nv];
11         }
12     }
13     void restore(int v) {
14         exists[v] = true;
15         for (int nv : G[v]) ++deg[nv];
16     }
17     void select(int v) {
18         cur.push_back(v);
19         ++block[v], erase(v);
20         for (int nv : G[v]) ++block[nv], erase(nv);
21     }
22     void unselect(int v) {
23         cur.pop_back();
24         --block[v], restore(v);
25         for (int nv : G[v]) {
26             if (--block[nv] == 0) restore(nv);
27         }
28     }
29     void dfs() {
30         int mn = n, v = -1;
31         rep(u, n) if (exists[u]) {
32             if (deg[u] < mn) mn = deg[u], v = u;
33         }
34         if (v == -1) {
35             ret.push_back(cur);
36         } else {
37             select(v), dfs(), unselect(v);
38             for (int nv : G[v]) {
39                 if (exists[nv]) select(nv), dfs(), unselect(nv);
40             }
41         }
42     }
43 public:
44     maximal_indsets(const graph& G) : n(G.size()), G(G), exists(n, true), deg(n), block(
45         n) {
46         rep(v, n) deg[v] = G[v].size();
47     }
48 }

```

```

46     dfs();
47 }
48     const vector<vector<int>>& get() const { return ret; }
49 };

```

## 3.5

## 4 数学

### 4.1 整数

#### 4.1.1 剩余

```

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 }

```

```
51 }
```

#### 4.1.2 剰余積

```
1 inline long mod_mul(long a, long b, long m) {
2     long ret = a * b - m * long(roundl((long double)(a) * b / m));
3     return ret < 0 ? ret + m : ret;
4 }
5 long mod_powl(long a, long b, long m) {
6     long ret = 1;
7     do {
8         if (b & 1) ret = mod_mul(ret, a, m);
9         a = mod_mul(a, a, m);
10    } while (b >= 1);
11    return ret;
12 }
```

#### 4.1.3 離散対数問題

```
1 long discrete_log(long a, long m) {
2     if (a == 0) return -1;
3     long b = long(sqrt(m)) + 1, t = 1;
4     unordered_map<long, long> mem;
5     for (int i = 0; i < b; ++i) {
6         mem[t] = i;
7         t = t * a % m;
8         if (t == 1) return i + 1;
9     }
10    long u = t;
11    for (int i = b; i < m; i += b) {
12        if (mem.find(mod_inverse(u, m)) != mem.end()) {
13            return mem[mod_inverse(u, m)] + i;
14        }
15        u = u * t % m;
16    }
17    return -1;
18 }
```

#### 4.1.4 カタラン数

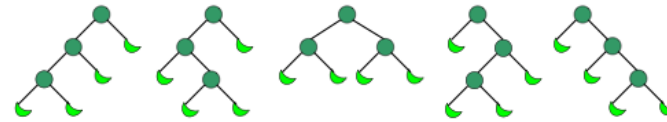
$n \leq 16$  程度が限度.  $n \geq 1$  について以下が成り立つ.

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n-1}$$

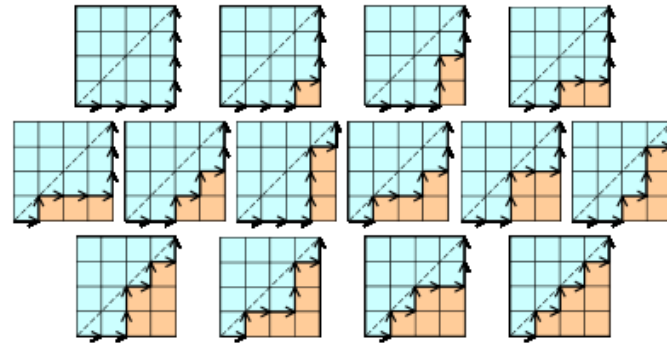
$n$  が十分大きいとき, カタラン数は以下に近似できる.

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

() 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.  
 $C_3 = 5$



$C_4 = 14$



#### 4.1.5 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 }
```

#### 4.1.6 確率的素数判定 (Miller-Rabin 法)

$O(k \log^3 n)$

合成数を素数と判定する確率は最大で  $4^{-k}$

```
1 bool suspect(long a, int s, long d, long n) {
2     long x = mod_pow(a, d, n); // use mod_powl instead for large n
3     if (x == 1) return true;
4     for (int r = 0; r < s; ++r) {
5         if (x == n - 1) return true;
6         x = x * x % n; // use mod_mul instead for large n
7     }
8     return false;
9 }
10 // {2,7,61,-1} is for n < 4759123141 (= 2^32)
11 // {2,3,5,7,11,13,17,19,23,-1} is for n < 10^16 (at least)
12 bool is_prime(long n) {
13     if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
14     int test[] = {2,3,5,7,11,13,17,19,23,-1};
15     long d = n - 1, s = 0;
```

```

16 while (d % 2 == 0) ++s, d /= 2;
17 for (int i = 0; test[i] < n && test[i] != -1; ++i)
18     if (!suspect(test[i], s, d, n)) return false;
19 return true;
20 }

```

## 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     }
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         repi(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         repi(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         repi(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         repi(j,r+1,n) repi(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) repi(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         repi(k, i, n) givens_rotate(A[i][k], A[j][k], c, s);
18     }
19     for (int i = n - 1; i >= 0; --i) {
20         repi(j, i + 1, n) b[i] -= A[i][j] * b[j];
21         b[i] /= A[i][i];
22     }
23     return b;
24 }

```

## 4.4 割り当て問題

### 4.4.1 ハンガリアン法

$O(N^2)$

```

1  int hungarian(const vector<vector<int>>> &a) {
2      int n = a.size(), p, q;

```

```

3   vector<int> fx(n, inf), fy(n, 0), x(n, -1), y(n, -1);
4   rep(i,n) rep(j,n) fx[i] = max(fx[i], a[i][j]);
5
6   for (int i = 0; i < n; ) {
7       vector<int> t(n, -1), s(n+1, i);
8       for (p = q = 0; p <= q && x[i] < 0; ++p)
9           for (int k = s[p], j = 0; j < n && x[i] < 0; ++j)
10              if (fx[k] + fy[j] == a[k][j] && t[j] < 0) {
11                  s[++q] = y[j], t[j] = k;
12                  if (s[q] < 0)
13                      for (p = j; p >= 0; j = p)
14                          y[j] = k = t[j], p = x[k], x[k] = j;
15              }
16              if (x[i] < 0) {
17                  int d = inf;
18                  rep(k,q+1) rep(j,n) if (t[j] < 0) d = min(d, fx[s[k]] + fy[j] - a[s[k]][j]);
19                  rep(j,n) fy[j] += (t[j] < 0 ? 0 : d);
20                  rep(k,q+1) fx[s[k]] -= d;
21              } else i++;
22          }
23          int ret = 0;
24          rep(i,n) ret += a[i][x[i]];
25          return ret;
26      }

```

## 5 幾何

```

1 // constants and eps-considered operators
2 const double eps = 1e-8; // choose carefully!
3 const double pi = acos(-1.0);
4
5 inline bool lt(double a, double b) { return a < b - eps; }
6 inline bool gt(double a, double b) { return lt(b, a); }
7 inline bool le(double a, double b) { return !lt(b, a); }
8 inline bool ge(double a, double b) { return !lt(a, b); }
9 inline bool ne(double a, double b) { return lt(a, b) or lt(b, a); }
10 inline bool eq(double a, double b) { return !ne(a, b); }

```

### 5.1 点

```

1 typedef complex<double> point;
2 inline double dot (point a, point b) { return real(conj(a) * b); }
3 inline double cross(point a, point b) { return imag(conj(a) * b); }
4 /*
5  * Here is what ccw(a, b, c) returns:
6  *
7  *      1
8  *  -----
9  *  2 |a  0  b| -2
10 *  -----
11 *      -1
12 *
13 * Note: we can implement intersectPS(p, s) as !ccw(s.a, s.b, p).
14 */
15 int ccw(point a, point b, point c) {
16     b -= a, c -= a;
17     if (cross(b, c) > eps) return +1;
18     if (cross(b, c) < eps) return -1;
19     if (dot(b, c) < eps) return +2; // c -- a -- b
20     if (lt(norm(b), norm(c))) return -2; // a -- b -- c

```

```

21     return 0;
22 }

```

### 5.2 直線と線分

```

1 struct line {
2     point a, b;
3     line(point a, point b) : a(a), b(b) {}
4 };
5
6 bool intersectLS(const line& l, const line& s) {
7     return ccw(l.a, l.b, s.a) * ccw(l.a, l.b, s.b) <= 0;
8 }
9 bool intersectSS(const line& s, const line& t) {
10    return intersectLS(s, t) and intersectLS(t, s);
11 }
12 bool intersectLL(const line& l, const line& m) {
13     return ne(cross(l.b - l.a, m.b - m.a), 0.0) // not parallel
14         or eq(cross(l.b - l.a, m.a - l.a), 0.0); // overlap
15 }
16 point crosspointLL(const line& l, const line& m) {
17     double A = cross(l.b - l.a, m.b - m.a);
18     double B = cross(l.b - l.a, m.a - l.a);
19     if (eq(A, 0.0) and eq(B, 0.0)) return m.a; // overlap
20     assert(ne(A, 0.0)); // not parallel
21     return m.a - B / A * (m.b - m.a);
22 }
23 point proj(const line& l, point p) {
24     double t = dot(l.b - l.a, p - l.a) / norm(l.b - l.a);
25     return l.a + t * (l.b - l.a);
26 }
27 point reflection(const line& l, point p) { return 2.0 * proj(l, p) - p; }
28
29 double distanceLP(const line& l, point p) { return abs(proj(l, p) - p); }
30 double distanceLL(const line& l, const line& m) {
31     return intersectLL(l, m) ? 0.0 : distanceLP(l, m.a);
32 }
33 double distanceLS(const line& l, const line& s) {
34     return intersectLS(l, s) ? 0.0 : min(distanceLP(l, s.a), distanceLP(l, s.b));
35 }
36 double distancePS(point p, const line& s) {
37     point h = proj(s, p);
38     return ccw(s.a, s.b, h) ? min(abs(s.a - p), abs(s.b - p)) : abs(h - p);
39 }
40 double distanceSS(const line& s, const line& t) {
41     if (intersectSS(s, t)) return 0.0;
42     return min(min(distancePS(s.a, t), distancePS(s.b, t)),
43               min(distancePS(t.a, s), distancePS(t.b, s)));
44 }

```

### 5.3 円

```

1 struct circle {
2     point o; double r;
3     circle(point o, double r) : o(o), r(r) {}
4 };
5
6 bool intersectCL(const circle& c, const line& l) {
7     return le(norm(proj(l, c.o) - c.o), c.r * c.r);
8 }

```

```

9 int intersectCS(const circle& c, const line& s) {
10     if (not intersectCL(c, s)) return 0;
11     double a = abs(s.a - c.o);
12     double b = abs(s.b - c.o);
13     if (lt(a, c.r) and lt(b, c.r)) return 0;
14     if (lt(a, c.r) or lt(b, c.r)) return 1;
15     return ccw(s.a, s.b, proj(s, c.o)) ? 0 : 2;
16 }
17 bool intersectCC(const circle& c, const circle& d) {
18     double dist = abs(d.o - c.o);
19     return le(abs(c.r - d.r), dist) and le(dist, c.r + d.r);
20 }
21 line crosspointCL(const circle& c, const line& l) {
22     point h = proj(l, c.o);
23     double a = sqrt(c.r * c.r - norm(h - c.o));
24     point d = a * (l.b - l.a) / abs(l.b - l.a);
25     return line(h - d, h + d);
26 }
27 line crosspointCC(const circle& c, const circle& d) {
28     double dist = abs(d.o - c.o), th = arg(d.o - c.o);
29     double ph = acos((c.r * c.r + dist * dist - d.r * d.r) / (2.0 * c.r * dist));
30     return line(c.o + polar(c.r, th - ph), c.o + polar(c.r, th + ph));
31 }
32
33 line tangent(const circle& c, double th) {
34     point h = c.o + polar(c.r, th);
35     point d = polar(c.r, th) * point(0, 1);
36     return line(h - d, h + d);
37 }
38 vector<line> common_tangents(const circle& c, const circle& d) {
39     vector<line> ret;
40     double dist = abs(d.o - c.o), th = arg(d.o - c.o);
41     if (abs(c.r - d.r) < dist) { // outer
42         double ph = acos((c.r - d.r) / dist);
43         ret.pb(tangent(c, th - ph));
44         ret.pb(tangent(c, th + ph));
45     }
46     if (abs(c.r + d.r) < dist) { // inner
47         double ph = acos((c.r + d.r) / dist);
48         ret.pb(tangent(c, th - ph));
49         ret.pb(tangent(c, th + ph));
50     }
51     return ret;
52 }
53 pair<circle, circle> tangent_circles(const line& l, const line& m, double r) {
54     double th = arg(m.b - m.a) - arg(l.b - l.a);
55     double ph = (arg(m.b - m.a) + arg(l.b - l.a)) / 2.0;
56     point p = crosspointLL(l, m);
57     point d = polar(r / sin(th / 2.0), ph);
58     return mp(circle(p - d, r), circle(p + d, r));
59 }
60 line bisector(point a, point b);
61 circle circum_circle(point a, point b, point c) {
62     point o = crosspointLL(bisector(a, b), bisector(a, c));
63     return circle(o, abs(a - o));
64 }

```

## 5.4 多角形

```

1 typedef vector<point> polygon;
2
3 double area(const polygon& g) {
4     double ret = 0.0;
5     int j = g.size() - 1;

```

```

6     rep(i, g.size()) {
7         ret += cross(g[j], g[i]), j = i;
8     }
9     return ret / 2.0;
10 }
11 point centroid(const polygon& g) {
12     if (g.size() == 1) return g[0];
13     if (g.size() == 2) return (g[0] + g[1]) / 2.0;
14     point ret = 0.0;
15     int j = g.size() - 1;
16     rep(i, g.size()) {
17         ret += cross(g[j], g[i]) * (g[j] + g[i]), j = i;
18     }
19     return ret / area(g) / 6.0;
20 }
21 line bisector(point a, point b) {
22     point m = (a + b) / 2.0;
23     return line(m, m + (b - a) * point(0, 1));
24 }
25 polygon convex_cut(const polygon& g, const line& l) {
26     polygon ret;
27     int j = g.size() - 1;
28     rep(i, g.size()) {
29         if (ccw(l.a, l.b, g[j]) != -1) ret.pb(g[j]);
30         if (intersectLS(l, line(g[j], g[i]))) ret.pb(crosspointLL(l, line(g[j], g[i])));
31         j = i;
32     }
33     return ret;
34 }
35 polygon voronoi_cell(polygon g, const vector<point>& v, int k) {
36     rep(i, v.size()) if (i != k) {
37         g = convex_cut(g, bisector(v[i], v[k]));
38     }
39     return g;
40 }

```

### 5.4.1 凸包

```

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

```

### 5.4.2 最近点对

だいたい  $O(n \log n)$ , 最悪縦 1 列に並んでる場合  $O(n^2)$

```

1 pair<point,point> closest_pair(vector<point> p) {
2     int n = p.size(), s = 0, t = 1, m = 2, S[n];
3     S[0] = 0, S[1] = 1;
4     sort(all(p)); // "p < q" <=> "p.x < q.x"
5     double d = norm(p[s]-p[t]);
6     for (int i = 2; i < n; S[m++] = i++) rep(j, m) {
7         if (norm(p[S[j]]-p[i])<d) d = norm(p[s = S[j]]-p[t = i]);
8         if (real(p[S[j]]) < real(p[i]) - d) S[j--] = S[--m];
9     }
10    return make_pair(p[s], p[t]);
11 }

```

### 5.4.3 点-多角形包含判定

$O(n)$

```

1 enum { OUT, ON, IN };
2 int contains(const polygon& P, const point& p) {
3     bool in = false;
4     for (int i = 0; i < (int)P.size(); ++i) {
5         point a = P[i] - p, b = P[(i+1)%P.size()] - p;
6         if (imag(a) > imag(b)) swap(a, b);
7         if (imag(a) <= 0 && 0 < imag(b) && cross(a, b) < 0) in = !in;
8         if (cross(a, b) == 0 && dot(a, b) <= 0) return ON;
9     }
10    return in ? IN : OUT;
11 }

```

### 5.4.4 凸多角形の共通部分

$O(n+m)$

```

1 bool intersect_lpt(const point& a, const point& b,
2                   const point& c, const point& d, point &r) {
3     number D = cross(b - a, d - c);
4     if (eq(D,0)) return false;
5     number t = cross(c - a, d - c) / D;
6     number s = -cross(a - c, b - a) / D;
7     r = a + t * (b - a);
8     return ge(t, 0) && le(t, 1) && ge(s, 0) && le(s, 1);
9 }
10 polygon convex_intersect(const polygon &P, const polygon &Q) {
11     const int n = P.size(), m = Q.size();
12     int a = 0, b = 0, aa = 0, ba = 0;
13     enum { Pin, Qin, Unknown } in = Unknown;
14     polygon R;
15     do {
16         int a1 = (a+n-1) % n, b1 = (b+m-1) % m;
17         number C = cross(P[a] - P[a1], Q[b] - Q[b1]);
18         number A = cross(P[a1] - Q[b], P[a] - Q[b]);
19         number B = cross(Q[b1] - P[a], Q[b] - P[a]);
20         point r;
21         if (intersect_lpt(P[a1], P[a], Q[b1], Q[b], r)) {
22             if (in == Unknown) aa = ba = 0;
23             R.push_back(r);
24             in = B > 0 ? Pin : A > 0 ? Qin : in;
25         }
26         if (C == 0 && B == 0 && A == 0) {
27             if (in == Pin) { b = (b + 1) % m; ++ba; }
28             else { a = (a + 1) % n; ++aa; }
29         } else if (C >= 0) {

```

```

30         if (A > 0) { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
31         else { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
32     } else {
33         if (B > 0) { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
34         else { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
35     }
36 } while ( (aa < n || ba < m) && aa < 2*n && ba < 2*m );
37 if (in == Unknown) {
38     if (convex_contains(Q, P[0])) return P;
39     if (convex_contains(P, Q[0])) return Q;
40 }
41 return R;
42 }

```

### 5.4.5 凸多角形の直径

$O(n)$

```

1 inline double diff(const vector<point> &P, const int &i) { return (P[(i+1)%P.size()] - P[i]); }
2 number convex_diameter(const polygon &pt) {
3     const int n = pt.size();
4     int is = 0, js = 0;
5     for (int i = 1; i < n; ++i) {
6         if (imag(pt[i]) > imag(pt[is])) is = i;
7         if (imag(pt[i]) < imag(pt[js])) js = i;
8     }
9     number maxd = norm(pt[is]-pt[js]);
10
11     int i, maxi, j, maxj;
12     i = maxi = is;
13     j = maxj = js;
14     do {
15         if (cross(diff(pt,i), diff(pt,j)) >= 0) j = (j+1) % n;
16         else i = (i+1) % n;
17         if (norm(pt[i]-pt[j]) > maxd) {
18             maxd = norm(pt[i]-pt[j]);
19             maxi = i; maxj = j;
20         }
21     } while (i != is || j != js);
22     return maxd; /* farthest pair is (maxi, maxj). */
23 }

```

### 5.4.6 ドロネー三角形分割 (逐次添加法)

$O(n^2)$

```

1 bool incircle(point a, point b, point c, point p) {
2     a -= p; b -= p; c -= p;
3     return norm(a) * cross(b, c)
4         + norm(b) * cross(c, a)
5         + norm(c) * cross(a, b) >= 0;
6     // < : inside, = cocircular, > outside
7 }
8 #define SET_TRIANGLE(i, j, r) \
9     E[i].insert(j); em[i][j] = r; \
10    E[j].insert(r); em[j][r] = i; \
11    E[r].insert(i); em[r][i] = j; \
12    S.push(pair<int,int>(i, j));
13 #define REMOVE_EDGE(i, j) \
14     E[i].erase(j); em[i][j] = -1; \
15     E[j].erase(i); em[j][i] = -1;

```



```

16 #define DECOMPOSE_ON(i,j,k,r) { \
17     int m = em[j][i]; REMOVE_EDGE(j,i); \
18     SET_TRIANGLE(i,m,r); SET_TRIANGLE(m,j,r); \
19     SET_TRIANGLE(j,k,r); SET_TRIANGLE(k,i,r); }
20 #define DECOMPOSE_IN(i,j,k,r) { \
21     SET_TRIANGLE(i,j,r); SET_TRIANGLE(j,k,r); \
22     SET_TRIANGLE(k,i,r); }
23 #define FLIP_EDGE(i,j) { \
24     int k = em[j][i]; REMOVE_EDGE(i,j); \
25     SET_TRIANGLE(i,k,r); SET_TRIANGLE(k,j,r); }
26 #define IS_LEGAL(i, j) \
27     (em[i][j] < 0 || em[j][i] < 0 || \
28     !incircle(P[i],P[j],P[em[i][j]],P[em[j][i]]))
29 double Delaunay(vector<point> P) {
30     const int n = P.size();
31     P.push_back( point(-inf,-inf) );
32     P.push_back( point(+inf,-inf) );
33     P.push_back( point( 0 ,+inf) );
34     int em[n+3][n+3]; memset(em, -1, sizeof(em));
35     set<int> E[n+3];
36     stack< pair<int,int> > S;
37     SET_TRIANGLE(n+0, n+1, n+2);
38     for (int r = 0; r < n; ++r) {
39         int i = n, j = n+1, k;
40         while (1) {
41             k = em[i][j];
42             if (ccw(P[i], P[em[i][j]], P[r]) == +1) j = k;
43             else if (ccw(P[j], P[em[i][j]], P[r]) == -1) i = k;
44             else break;
45         }
46         if (ccw(P[i], P[j], P[r]) != +1) { DECOMPOSE_ON(i,j,k,r); }
47         else if (ccw(P[j], P[k], P[r]) != +1) { DECOMPOSE_ON(j,k,i,r); }
48         else if (ccw(P[k], P[i], P[r]) != +1) { DECOMPOSE_ON(k,i,j,r); }
49         else { DECOMPOSE_IN(i,j,k,r); }
50         while (!S.empty()) {
51             int u = S.top().first, v = S.top().second; S.pop();
52             if (!IS_LEGAL(u, v)) FLIP_EDGE(u, v);
53         }
54     }
55     double minarg = 1e5;
56     for (int a = 0; a < n; ++a) {
57         for(auto &b: E[a]) {
58             int c = em[a][b];
59             if (b < n && c < n) {
60                 point p = P[a] - P[b], q = P[c] - P[b];
61                 minarg = min(minarg, acos(dot(p,q)/abs(p)/abs(q)));
62             }
63         }
64     }
65     return minarg;
66 }

```

## 6 データ構造

### 6.1 Union-Find 木

```

1 class disjoint_set {
2     vector<int> p;
3 public:
4     disjoint_set(int n) : p(n, -1) {}
5     int root(int i) { return p[i] >= 0 ? p[i] = root(p[i]) : i; }
6     bool same(int i, int j) { return root(i) == root(j); }
7     int size(int i) { return -p[root(i)]; }

```

```

8     void merge(int i, int j) {
9         i = root(i), j = root(j);
10        if (i == j) return;
11        if (p[i] > p[j]) swap(i, j);
12        p[i] += p[j], p[j] = i;
13    }
14 };

```

### 6.2 Meldable Heap

```

1 template <class T>
2 class meldable_heap {
3     struct node {
4         node *l = NULL, *r = NULL;
5         T val;
6         node(const T& val) : val(val) {}
7         ~node() { delete l, delete r; }
8     };
9     node *meld(node *a, node *b) {
10        if (!a) return b;
11        if (!b) return a;
12        if (a->val > b->val) swap(a, b);
13        a->r = meld(a->r, b);
14        swap(a->l, a->r);
15        return a;
16    }
17    node *root = NULL;
18    meldable_heap(node *root) : root(root) {}
19 public:
20    meldable_heap() {}
21    bool empty() const { return !root; }
22    const T& top() const { return root->val; }
23    void meld(const meldable_heap<T>&& t) { root = meld(root, t.root); }
24    void push(const T& val) { root = meld(root, new node(val)); }
25    void pop() {
26        node *t = root;
27        root = meld(t->l, t->r);
28        t->l = t->r = NULL;
29        delete t;
30    }
31 };

```

### 6.3 Binary-Indexed-Tree

0-indexed

```

1 template<class T> struct bit {
2     int n;
3     vector<T> dat;
4
5     bit(int n) : n(n) { dat.assign(n,0); }
6     // sum [0,i)
7     T sum(int i){
8         int ret = 0;
9         for(--i; i>=0; i=(i&(i+1))-1) ret += dat[i];
10        return ret;
11    }
12    // sum [i,j)
13    T sum(int i, int j){ return sum(j) - sum(i); }
14    // add x to i
15    void add(int i, T x){ for(; i < n; i|=i+1) dat[i] += x; }

```

```
16 };
```

## 6.4 Segment Tree

区間 add と RMQ ができる。

```
1  template<class T> struct segtree {
2      int N;
3      vector<T> dat, sum;
4      segtree(int n) {
5          N = 1;
6          while(N < n) N <= 1;
7          dat.assign(2*N-1,0);
8          sum.assign(2*N-1,0);
9      }
10     void add(int a, int b, T x) { add(a,b,x,0,0,N);}
11     T add(int a, int b, T x, int k, int l, int r) {
12         if(b <= l or r <= a) return dat[k];
13         if(a <= l and r <= b) {
14             sum[k] += x;
15             return dat[k] += x;
16         }
17         int m = (l+r)/2;
18         return dat[k] = min(add(a,b,x,2*k+1,l,m),add(a,b,x,2*k+2,m,r))+sum[k];
19     }
20     T minimum(int a, int b) { return minimum(a,b,0,0,N);}
21     T minimum(int a, int b, int k, int l, int r) {
22         if(b <= l or r <= a) return 1e9;
23         if(a <= l and r <= b) return dat[k];
24         int m = (l+r)/2;
25         return min(minimum(a,b,2*k+1,l,m),minimum(a,b,2*k+2,m,r))+sum[k];
26     }
27 };
```

## 6.5 Sparse table

```
1  const int N = 200010;
2  const int K = 18;
3  int st[K][N];
4  void construct(int *a, int n) {
5      copy_n(a, n, st[0]);
6      repi(k, 1, K) {
7          for (int i = 0; i+(1<<k) <= n; ++i) {
8              st[k][i] = min(st[k-1][i], st[k-1][i+(1<<(k-1))]);
9          }
10     }
11 }
12 int query(int a, int b) {
13     int k = 31-__builtin_clz(b-a);
14     return min(st[k][a], st[k][b-(1<<k)]);
15 }
```

## 6.6 赤黒木

```
1  template<class T> class rbtree {
2      enum COL { BLACK, RED,};
3      struct node {
4          T val, lazy, min_val;
```

```
5      int color, rnk, size;
6      node *left, *right;
7      // if !left then this node is leaf
8      node(){}
9      node(T v) : val(v), min_val(v), color(BLACK), rnk(0), size(1) {
10         lazy = 0;
11         left = right = NULL;
12     }
13     node(node *l, node *r, int c) : color(c) {
14         lazy = 0;
15         left = l;
16         right = r;
17         update();
18     }
19     void update() {
20         eval();
21         if(left) {
22             rnk = max(left->rnk+(left->color==BLACK),
23                     right->rnk+(right->color==BLACK));
24             size = left->size+right->size;
25             left->eval(); right->eval();
26             min_val = min(left->min_val, right->min_val);
27         }
28     }
29     void eval() {
30         min_val += lazy;
31         if(!left) val += lazy;
32         else {
33             left->lazy += lazy;
34             right->lazy += lazy;
35         }
36         lazy = 0;
37     }
38 };
39
40 node *new_node(T v) { return new node(v);}
41 node *new_node(node *l, node *r, int c) { return new node(l,r,c);}
42 node *rotate(node *v, int d) {
43     node *w = d? v->right: v->left;
44     if(d) {
45         v->right = w->left;
46         w->left = v;
47         v->right->update();
48     }
49     else {
50         v->left = w->right;
51         w->right = v;
52         v->left->update();
53     }
54     v->update(); w->update();
55     v->color = RED;
56     w->color = BLACK;
57     return w;
58 }
59 node *merge_sub(node *u, node *v) {
60     u->eval(); v->eval();
61     if(u->rnk < v->rnk) {
62         node *w = merge_sub(u,v->left);
63         v->left = w;
64         v->update();
65         if(v->color == BLACK and w->color == RED and w->left->color == RED) {
66             if(v->right->color == BLACK) return rotate(v,0);
67             else {
68                 v->color = RED;
69                 v->left->color = v->right->color = BLACK;
70                 return v;
71             }
72         }
73     }
```

```

72     }
73     else return v;
74 }
75 else if(u->rnk > v->rnk) {
76     node *w = merge_sub(u->right,v);
77     u->right = w;
78     u->update();
79     if(u->color == BLACK and w->color == RED and w->right->color == RED) {
80         if(u->left->color == BLACK) return rotate(u,l);
81         else {
82             u->color = RED;
83             u->left->color = u->right->color = BLACK;
84             return u;
85         }
86     }
87     else return u;
88 }
89 else return new_node(u,v,RED);
90 }
91 node *insert(node *v, int k) {
92     auto p = split(root,k);
93     return root = merge(merge(p.first,v),p.second);
94 }
95 void add(node *v, int res, T val) {
96     if(res < 1) return;
97     v->eval();
98     if(v->size == res) {
99         v->lazy += val;
100         return;
101     }
102     add(v->left, min(v->left->size, res), val);
103     add(v->right, res-v->left->size, val);
104     v->update();
105 }
106 T get(node *v, int k) {
107     v->eval();
108     if(!v->left) return v->val;
109     if(v->left->size > k) return get(v->left, k);
110     return get(v->right, k-v->left->size);
111 }
112 T minimum(node *v, int l, int r) {
113     if(r-l < 1) return inf;
114     v->eval();
115     if(v->size == r-l) return v->min_val;
116     return min(minimum(v->left, l, min(r, v->left->size)),
117               minimum(v->right, l-min(l, v->left->size), r-v->left->size));
118 }
119 T inf;
120 public:
121
122     node *root;
123     rbtree() {
124         inf = (((1LL<<((sizeof(T)*8-2))-1)<<1)+1;
125         root = NULL;
126     }
127     void clear() { delete root; root = NULL;}
128     node *build(const vector<T> &vs) {
129         if(!vs.size()) return root = NULL;
130         if((int)vs.size() == 1) return root = new_node(vs[0]);
131         int m = vs.size()/2;
132         return root = merge(build(vector<T>(begin(vs),begin(vs)+m)),
133                             build(vector<T>(begin(vs)+m,end(vs))));
134     }
135     int size() { return root? root->size: 0;}
136     node *push_back(T val) { return root = merge(root,new_node(val));}
137     node *push_front(T val) { return root = merge(new_node(val),root);}
138     node *merge(node *u, node *v) {

```

```

139         if(!u) return v;
140         if(!v) return u;
141         u = merge_sub(u,v);
142         u->color = BLACK;
143         return u;
144     }
145     pair<node*,node*> split(node *v, int k) {
146         if(!k) return pair<node*,node*>(NULL,v);
147         if(k == v->size) return pair<node*,node*>(v,NULL);
148         v->eval();
149         if(k < v->left->size) {
150             auto p = split(v->left,k);
151             return pair<node*,node*>(p.first,merge(p.second,v->right));
152         }
153         else if(k > v->left->size) {
154             auto p = split(v->right,k-v->left->size);
155             return pair<node*,node*>(merge(v->left,p.first),p.second);
156         }
157         else return pair<node*,node*>(v->left,v->right);
158     }
159
160     node *insert(int k, T val) { return insert(new_node(val),k);}
161     node *erase(int k) {
162         auto p = split(root,k+1);
163         return root = merge(split(p.first,k).first, p.second);
164     }
165     void add(int l, int r, T val) { add(root, r, val); add(root, l, -val);}
166     T get(int k) { return get(root, k);}
167     T minimum(int l, int r) { return minimum(root, l, r);}
168     T operator[](const int &i) { return get(i);}
169 };

```

## 6.7 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 }

```

```

92 int size() { return root->size;}
93
94 void get(vector<T> &vs) { get(root,vs);}
95 void get(node *v, vector<T> &vs) {
96     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
124 // copy and insert [l,r) at k
125 node *copy_paste(int l, int r, int k) {
126     return insert(copy(l,r),k);
127 }

```

## 6.8 wavelet 行列

N := 列の長さ

M := 最大値

### 6.8.1 完備辞書

function	計算量
count	$O(1)$
select	$O(\log N)$

```

1 template<int N> class FID {
2     static const int bucket = 512, block = 16;
3     static char popcount[];
4     int n, B[N/bucket+10];
5     unsigned short bs[N/block+10], b[N/block+10];
6
7 public:
8     FID(){}
9     FID(int n, bool s[]) : n(n) {

```

```

10     if(!popcount[l]) for (int i = 0; i < (1<<block); i++) popcount[i] =
11         __builtin_popcount(i);
12
13     bs[0] = B[0] = b[0] = 0;
14     for (int i = 0; i < n; i++) {
15         if(i%block == 0) {
16             bs[i/block+1] = 0;
17             if(i%bucket == 0) {
18                 B[i/bucket+1] = B[i/bucket];
19                 b[i/block+1] = b[i/block] = 0;
20             }
21             else b[i/block+1] = b[i/block];
22         }
23         bs[i/block] |= short(s[i])<<(i%block);
24         b[i/block+1] += s[i];
25         B[i/bucket+1] += s[i];
26     }
27     if(n%bucket == 0) b[n/block] = 0;
28
29     // number of val in [0,r), O(1)
30     int count(bool val, int r) { return val? B[r/bucket]+b[r/block]+popcount[bs[r/block
31         ]&((1<<(r%block))-1)]: r-count(1,r); }
32     // number of val in [l,r), O(1)
33     int count(bool val, int l, int r) { return count(val,r)-count(val,l); }
34     // position of ith in val, 0-indexed, O(log n)
35     int select(bool val, int i) {
36         if(i < 0 or count(val,n) <= i) return -1;
37         i++;
38         int lb = 0, ub = n, md;
39         while(ub-lb>1) {
40             md = (lb+ub)>>1;
41             if(count(val,md) >= i) ub = md;
42             else lb = md;
43         }
44         return ub-1;
45     }
46     int select(bool val, int i, int l) { return select(val,i+count(val,l)); }
47     bool operator[](int i) { return bs[i/block]>>(i%block)&1; }
48 };
49 template<int N> char FID<N>::popcount[1<<FID<N>::block];

```

## 6.8.2 wavelet 行列

function	計算量	FID::count	FID::select
count	$O(\log M)$	o	
select	$O(\log N \log M)$	o	o
get	$O(\log M)$	o	
maximum	$O(\log M)$ or $O(k \log M)$	o	
kth_number	$O(\log M)$	o	
freq	$O(\log M)$	o	
freq_list	$O(k \log M)$	o	
get_rect	$O(k \log N \log M)$	o	o

```

1 template<class T, int N, int D> class wavelet {
2     int n, zs[D];
3     FID<N> dat[D];
4

```

```

5 void max_dfs(int d, int l, int r, int &k, T val, vector<T> &vs) {
6     if(l >= r or !k) return;
7     if(d == D) {
8         while(l++ < r and k > 0) vs.push_back(val), k--;
9         return;
10    }
11    int lc = dat[d].count(1,l), rc = dat[d].count(1,r);
12    // if min, change this order
13    max_dfs(d+1, lc+zs[d], rc+zs[d], k, 1ULL<<(D-d-1)|val,vs);
14    max_dfs(d+1, l-lc, r-rc, k, val, vs);
15 }
16
17 T max_dfs(int d, int l, int r, T val, T a, T b) {
18     if(r-l <= 0 or val >= b) return -1;
19     if(d == D) return val>=a? val: -1;
20     int lc = dat[d].count(1,l), rc = dat[d].count(1,r);
21     T ret = max_dfs(d+1, lc+zs[d], rc+zs[d], 1ULL<<(D-d-1)|val, a, b);
22     if(!ret) return ret;
23     return max_dfs(d+1, l-lc, r-rc, val, a, b);
24 }
25
26 int freq_dfs(int d, int l, int r, T val, T a, T b) {
27     if(l == r) return 0;
28     if(d == D) return (a <= val and val < b)? r-l: 0;
29     T nv = 1ULL<<(D-d-1)|val, nnv = ((1ULL<<(D-d-1))-1)|nv;
30     if(nnv < a or b <= val) return 0;
31     if(a <= val and nnv < b) return r-l;
32     int lc = dat[d].count(1,l), rc = dat[d].count(1,r);
33     return freq_dfs(d+1,l-lc,r-rc,val,a,b)+
34         freq_dfs(d+1,lc+zs[d],rc+zs[d],nv,a,b);
35 }
36
37 void list_dfs(int d, int l, int r, T val, T a, T b, vector<pair<T,int>> &vs) {
38     if(val >= b or r-l <= 0) return;
39     if(d == D) {
40         if(a <= val) vs.push_back(make_pair(val,r-l));
41         return;
42     }
43     T nv = val|(1LL<<(D-d-1)), nnv = nv|(((1LL<<(D-d-1))-1));
44     if(nnv < a) return;
45     int lc = dat[d].count(1,l), rc = dat[d].count(1,r);
46     list_dfs(d+1,l-lc,r-rc,val,a,b,vs);
47     list_dfs(d+1,lc+zs[d],rc+zs[d],nv,a,b,vs);
48 }
49 public:
50     wavelet(int n, T seq[]) : n(n) {
51         T f[N], l[N], r[N];
52         bool b[N];
53         memcpy(f, seq, sizeof(T)*n);
54         for (int d = 0; d < D; d++) {
55             int lh = 0, rh = 0;
56             for (int i = 0; i < n; i++) {
57                 bool k = (f[i]>>(D-d-1))&1;
58                 if(k) r[rh++] = f[i];
59                 else l[lh++] = f[i];
60                 b[i] = k;
61             }
62             dat[d] = FID<N>(n,b);
63             zs[d] = lh;
64             swap(l,f);
65             memcpy(f+lh, r, rh*sizeof(T));
66         }
67     }
68
69     T get(int i) {
70         T ret = 0;
71         bool b;

```

```

72     for (int d = 0; d < D; d++) {
73         ret <= 1;
74         b = dat[d][i];
75         ret |= b;
76         i = dat[d].count(b,i)+b*zs[d];
77     }
78     return ret;
79 }
80 T operator[](int i) { return get(i); }
81
82 int count(T val, int l, int r) {
83     for (int d = 0; d < D; d++) {
84         bool b = (val>>(D-d-1))&1;
85         l = dat[d].count(b,l)+b*zs[d];
86         r = dat[d].count(b,r)+b*zs[d];
87     }
88     return r-l;
89 }
90 int count(T val, int r) { return count(val,0,r); }
91
92 int select(T val, int k) {
93     int ls[D], rs[D], l = 0, r = n;
94     for (int d = 0; d < D; d++) {
95         ls[d] = l; rs[d] = r;
96         bool b = val>>(D-d-1)&1;
97         l = dat[d].count(b,l)+b*zs[d];
98         r = dat[d].count(b,r)+b*zs[d];
99     }
100    for (int d = D-1; d >= 0; d--) {
101        bool b = val>>(D-d-1)&1;
102        k = dat[d].select(b,k,ls[d]);
103        if(k >= rs[d] or k < 0) return -1;
104        k -= ls[d];
105    }
106    return k;
107 }
108 int select(T val, int k, int l) { return select(val,k+count(val,l)); }
109
110 vector<T> maximum(int l, int r, int k) {
111     if (r-l < k) k = r-l;
112     if(k < 0) return {};
113     vector<T> ret;
114     max_dfs(0,l,r,k,0,ret);
115     return ret;
116 }
117
118 T maximum(int l, int r, T a, T b) { return max_dfs(0,l,r,0,a,b); }
119
120 // k is 0-indexed
121 T kth_number(int l, int r, int k) {
122     if(r-l <= k or k < 0) return -1;
123     T ret = 0;
124     for (int d = 0; d < D; d++) {
125         int lc = dat[d].count(1,l), rc = dat[d].count(1,r);
126         if(rc-lc > k) {
127             l = lc+zs[d];
128             r = rc+zs[d];
129             ret |= 1ULL<<(D-d-1);
130         }
131         else {
132             k -= rc-lc;
133             l -= lc;
134             r -= rc;
135         }
136     }
137     return ret;
138 }

```

```

139
140 vector<pair<T,int>> freq_list(int l, int r, T a, T b) {
141     vector<pair<T,int>> ret;
142     list_dfs(0,l,r,0,a,b,ret);
143     return ret;
144 }
145
146 vector<pair<int,T>> get_rect(int l, int r, T a, T b) {
147     vector<pair<T,int>> res = freq_list(l,r,a,b);
148     vector<pair<int,T>> ret;
149     for(auto &e: res)
150         for (int i = 0; i < e.second; i++)
151             ret.push_back(make_pair(select(e.first,i,l), e.first));
152     return ret;
153 }
154 // number of elements in [l,r) in [a,b), 0(D)
155 int freq(int l, int r, T a, T b) { return freq_dfs(0,l,r,0,a,b); }
156 };

```

## 7 その他

### 7.1 ビジュアライザ

```

1 <script>
2 function line(x,y,a,b){c.b();c.moveTo(x,y);c.lineTo(a,b);c.s();}
3 function circle(x,y,r){c.b();c.arc(x,y,r,0,7,0);c.s();}
4 window.onload=function(){d=document;d.i=d.getElementById;
5 c=d.i('c').getContext('2d');c.b=c.beginPath;c.s=c.stroke;
6 d.i('s').src='data.js?';};
7 </script>
8 <body><canvas id="c" width="500" height="500" style="border:1px solid #000;"></canvas>
9 <script id="s"></script></body>

```