Contents

1 準備

1.1 init.el

linum は emacs24 のみ

```
;key
(keyboard-translate ?\C-h ?\C-?)
(global-set-key "\M-g" 'goto-line)

;tab
(setq-default indent-tabs-mode nil)
(setq-default tab-width 4)
(setq indent-line-function 'insert-tab)

;line number
(global-linum-mode t)
(setq linum-format "%4d ")
```

1.2 tpl.cpp

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

#define rep(i,n) repi(i,0,n)
#define repi(i,a,b) for(int i=int(a);i<int(b);++i)
#define repit(it,u) for(auto it=begin(u);it!=end(u);++it)
#define all(u) begin(u),end(u)
#define uniq(u) (u).erase(unique(all(u)),end(u))
#define ll long
#define long int64_t
#define mp make_pair
#define pb push_back
#define eb emplace_back

bool input()
{
    return true;
}

void solve()
{

}

int main()
{
    cin.tie(0);
    ios_base::sync_with_stdio(false);

    while (input()) solve();
}
```

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)

$O(N+M)$

```cpp
const int C = 128;

struct pma_node {
    pma_node *next[C]; // use next[0] as failure link
    vector<int> match;
    pma_node() { fill(next, next + C, (pma_node *) NULL); }
    ~pma_node() { rep(i, C) if (next[i] != NULL) delete next[i]; }
};

pma_node *construct_pma(const vector<string>& pat) {
    pma_node *const root = new pma_node();
    root->next[0] = root;
    // construct trie
    rep(i, pat.size()) {
        const string& s = pat[i];
        pma_node *now = root;
        for (const char c : s) {
            if (now->next[int(c)] == NULL) now->next[int(c)] = new pma_node();
            now = now->next[int(c)];
        }
        now->match.pb(i);
    }
    // make failure links by BFS
    queue<pma_node *> q;
    repi(i, 1, C) {
        if (root->next[i] == NULL) root->next[i] = root;
        else {
            root->next[i]->next[0] = root;
            q.push(root->next[i]);
        }
    }
    while (not q.empty()) {
        auto now = q.front();
        q.pop();
        repi(i, 1, C) if (now->next[i] != NULL) {
            auto next = now->next[0];
            while (next->next[i] == NULL) next = next->next[0];
            now->next[i]->next[0] = next->next[i];
            vector<int> tmp;
            set_union(all(now->next[i]->match), all(next->next[i]->match), back_inserter
                (tmp));
            now->next[i]->match = tmp;
            q.push(now->next[i]);
        }
    }
    return root;
}

void match(pma_node*& now, const string s, vector<int>& ret) {
    for (const char c : s) {
        while (now->next[int(c)] == NULL) now = now->next[0];
        now = now->next[int(c)];
        for (const int e : now->match) ret[e] = true;
    }
}
```

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.

```cpp
const int MAX_N = 1000000;
int n, k;
int rnk[MAX_N+1], tmp[MAX_N+1], sa[MAX_N+1], lcp[MAX_N+1];

bool compare_sa(int i, int j) {
  if(rnk[i] != rnk[j]) return rnk[i] < rnk[j];
  else {
    int ri = i + k <= n ? rnk[i+k] : -1;
    int rj = j + k <= n ? rnk[j+k] : -1;
    return ri < rj;
  }
}

void construct_sa(string S, int *sa) {
  n = S.length();
  for(int i = 0; i <= n; i++) {
    sa[i] = i;
    rnk[i] = i < n ? S[i] : -1;
  }
  for(k = 1; k <= n; k*=2) {
    sort(sa, sa+n+1, compare_sa);
    tmp[sa[0]] = 0;
    for(int i = 1; i <= n; i++) {
      tmp[sa[i]] = tmp[sa[i-1]] + (compare_sa(sa[i-1], sa[i]) ? 1 : 0);
    }
    for(int i = 0; i <= n; i++) {
      rnk[i] = tmp[i];
    }
  }
}

void construct_lcp(string S, int *sa, int *lcp) {
  int n = S.length();
  for(int i = 0; i <= n; i++) rnk[sa[i]] = i;
  int h = 0;
  lcp[0] = 0;
  for(int i = 0; i < n; i++) {
    int j = sa[rnk[i] - 1];
    if(h > 0) h--;
    for(; j + h < n && i + h < n; h++) {
      if(S[j+h] != S[i+h]) break;
    }
    lcp[rnk[i] - 1] = h;
  }
}

//============ 使用例 =============//
// 文字列検索(蟻本p338 改) 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
int find_string(string S, int *sa, string T) {
  int a = 0, b = S.length();
  while(b - a > 1) {
    int c = (a + b) / 2;
    if(S.compare(sa[c], T.length(), T) < 0) a = c;
    else b = c;
  }
  return (S.compare(sa[b], T.length(), T) == 0)?sa[b]:-1;
}

// 最長共通部分文字列(蟻本p341 改) construct_sa以外はO(|S+T|)
// (先頭, 長さ)を返す
pair<int, int> LCS(string S, string T) {
  int sl = S.length();
  S += '\0' + T;
  construct_sa(S, sa);
  construct_lcp(S, sa, lcp);
  int len = 0, pos = -1;
  for(int i = 0; i < S.length(); i++) {
    if(((sa[i] < sl) != (sa[i+1] < sl)) && (len < lcp[i])) {
      len = lcp[i];
      pos = sa[i];
    }
  }
  return make_pair(pos, len);
}
```

3 グラフ

```cpp
struct edge {
    int to; long w;
    edge(int to, long w) : to(to), w(w) {}
};
typedef vector<vector<edge> > graph;

graph rev(const graph& G) {
    const int n = G.size();
    graph ret(n);
    rep(i, n) for (const auto& e : G[i]) {
        ret[e.to].eb(i, e.w);
    }
    return ret;
}
```

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.

```cpp
typedef vector<vector<int> > graph;

class articulation {
    const int n;
    graph G;
    int cnt;
    vector<int> num, low, art;
    void dfs(int v) {
        num[v] = low[v] = ++cnt;
        for (int nv : G[v]) {
            if (num[nv] == 0) {
                dfs(nv);
                low[v] = min(low[v], low[nv]);
                if ((num[v] == 1 and num[nv] != 2) or
                    (num[v] != 1 and low[nv] >= num[v])) {
                    art[v] = true;
                }
            } else {
                low[v] = min(low[v], num[nv]);
            }
        }
    }
public:
    articulation(const graph& G) : n(G.size()), G(G), cnt(0), num(n), low(n), art(n) {
        rep(i, n) if (num[i] == 0) dfs(i);
    }
    vector<int> get() {
        return art;
    }
};
```

3.1.2 橋

$O(V + E)$

```cpp
typedef vector<vector<int> > graph;

class bridge {
    const int n;
    graph G;
    int cnt;
    vector<int> num, low, in;
    stack<int> stk;
    vector<pair<int, int> > brid;
    vector<vector<int> > comp;
    void dfs(int v, int p) {
        num[v] = low[v] = ++cnt;
        stk.push(v), in[v] = true;
        for (const int nv : G[v]) {
            if (num[nv] == 0) {
                dfs(nv, v);
                low[v] = min(low[v], low[nv]);
            } else if (nv != p and in[nv]) {
                low[v] = min(low[v], num[nv]);
            }
        }
        if (low[v] == num[v]) {
            if (p != n) brid.eb(min(v, p), max(v, p));
            comp.eb();
            int w;
            do {
                w = stk.top();
                stk.pop(), in[w] = false;
                comp.back().pb(w);
            } while (w != v);
        }
    }
public:
    bridge(const graph& G) : n(G.size()), G(G), cnt(0), num(n), low(n), in(n) {
        rep(i, n) if (num[i] == 0) dfs(i, n);
    }
    vector<pair<int, int> > get() {
        return brid;
    }
    vector<vector<int> > components() {
        return comp;
    }
};
```

3.1.3 強連結成分分解

$O(V + E)$

```cpp
typedef vector<vector<int> > graph;

class scc {
    const int n;
    graph G;
    int cnt;
    vector<int> num, low, in;
    stack<int> stk;
    vector<vector<int> > comp;
    void dfs(int v) {
        num[v] = low[v] = ++cnt;
        stk.push(v), in[v] = true;
        for (const int nv : G[v]) {
            if (num[nv] == 0) {
                dfs(nv);
                low[v] = min(low[v], low[nv]);
            } else if (in[nv]) {
                low[v] = min(low[v], num[nv]);
            }
        }
        if (low[v] == num[v]) {
            comp.eb();
            int w;
            do {
                w = stk.top();
                stk.pop(), in[w] = false;
                comp.back().pb(w);
            } while (w != v);
        }
    }
public:
    scc(const graph& G) : n(G.size()), G(G), cnt(0), num(n), low(n), in(n) {
        rep(i, n) if (num[i] == 0) dfs(i);
    }
    vector<vector<int> > components() {
        return comp;
    }
};
```

3.2 フロー

3.2.1 最大流

$O(EV^2)$

```
const int inf = 1e9;
struct edge {
    int to, cap, rev;
    edge(int to, int cap, int rev) : to(to), cap(cap), rev(rev) {}
};
typedef vector<vector<edge> > graph;

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

class max_flow {
    const int n;
    graph& G;
    vector<int> level, iter;
    void bfs(int s, int t) {
        level.assign(n, -1);
        queue<int> q;
        level[s] = 0, q.push(s);
        while (not q.empty()) {
            const int v = q.front();
            q.pop();
            if (v == t) return;
            for (const auto& e : G[v]) {
                if (e.cap > 0 and level[e.to] < 0) {
                    level[e.to] = level[v] + 1;
                    q.push(e.to);
                }
            }
        }
    }
    int dfs(int v, int t, int f) {
        if (v == t) return f;
        for (int& i = iter[v]; i < (int) G[v].size(); ++i) {
            edge& e = G[v][i];
            if (e.cap > 0 and level[v] < level[e.to]) {
                const int d = dfs(e.to, t, min(f, e.cap));
                if (d > 0) {
                    e.cap -= d, G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }
public:
    max_flow(graph& G) : n(G.size()), G(G) {}
    int calc(int s, int t) {
        int ret = 0, d;
        while (bfs(s, t), level[t] >= 0) {
            iter.assign(n, 0);
            while ((d = dfs(s, t, inf)) > 0) ret += d;
        }
        return ret;
    }
};
```

3.2.2 二部マッチング

$O(EV)$

```
int  V;
vector<int> G[MAX_V];
int match[MAX_V];
bool used[MAX_V];

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

bool dfs(int v){
    used[v] = 1;
    rep(i,G[v].size()){
        int u = G[v][i], w = match[u];
        if(w < 0 || !used[w] && dfs(w)){
            match[v] = u;
            match[u] = v;
            return 1;
        }
    }
    return 0;
}

int bi_matching(){
    int res = 0;
    memset(match, -1, sizeof(match));
    rep(v,V) if(match[v] < 0){
        memset(used, 0, sizeof(used));
        if(dfs(v)) res++;
    }
    return res;
}
```

3.2.3 最小費用流

$O(FE \log V)$

```
const int inf = 1e9;
struct edge {
    int to, cap, cost, rev;
    edge(int to, int cap, int cost, int rev) : to(to), cap(cap), cost(cost), rev(rev) {}
};
typedef vector<vector<edge> > graph;

void add_edge(graph& G, int from, int to, int cap, int cost) {
    G[from].eb(to, cap, cost, G[to].size());
    G[to].eb(from, 0, -cost, G[from].size() - 1);
}

int min_cost_flow(graph& G, int s, int t, int f) {
    const int n = G.size();
    struct state {
        int v, d;
        state(int v, int d) : v(v), d(d) {}
        bool operator <(const state& t) const { return d > t.d; }
    };

    int ret = 0;
    vector<int> h(n, 0), dist, prev(n), prev_e(n);
    while (f > 0) {
        dist.assign(n, inf);
        priority_queue<state> q;
        dist[s] = 0, q.emplace(s, 0);
        while (not q.empty()) {
            const int v = q.top().v;
            const int d = q.top().d;
            q.pop();
            if (dist[v] <= d) continue;
            rep(i, G[v].size()) {
                const edge& e = G[v][i];
                if (e.cap > 0 and dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]) {
                    dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
                    prev[e.to] = v, prev_e[e.to] = i;
                    q.emplace(e.to, dist[e.to]);
                }
            }
        }
        if (dist[t] == inf) return -1;
        rep(i, n) h[i] += dist[i];

        int d = f;
        for (int v = t; v != s; v = prev[v]) {
            d = min(d, G[prev[v]][prev_e[v]].cap);
        }
        f -= d, ret += d * h[t];
        for (int v = t; v != s; v = prev[v]) {
            edge& e = G[prev[v]][prev_e[v]];
            e.cap -= d, G[v][e.rev].cap += d;
        }
    }
    return ret;
}
```

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

```
#include "disjoint_set.cpp"
#include "graph.cpp"

struct mst_edge {
    int u, v; long w;
    mst_edge(int u, int v, long w) : u(u), v(v), w(w) {}
    bool operator <(const mst_edge& t) const { return w < t.w; }
    bool operator >(const mst_edge& t) const { return w > t.w; }
};

graph kruskal(const graph& G) {
    const int n = G.size();
    vector<mst_edge> E;
    rep(i, n) for (const auto& e : G[i]) {
        if (i < e.to) E.eb(i, e.to, e.w);
    }
    sort(all(E));

    graph T(n);
    disjoint_set uf(n);
    for (const auto& e : E) {
        if (not uf.same(e.u, e.v)) {
            T[e.u].eb(e.v, e.w);
            T[e.v].eb(e.u, e.w);
            uf.merge(e.u, e.v);
        }
    }
    return T;
}

graph prim(const vector<vector<long> >& A, int s = 0) {
    const int n = A.size();
    graph T(n);
    vector<int> done(n);
    priority_queue<mst_edge, vector<mst_edge>, greater<mst_edge> > q;
    q.emplace(-1, s, 0);
    while (not q.empty()) {
        const auto e = q.top();
        q.pop();
        if (done[e.v]) continue;
        done[e.v] = 1;
        if (e.u >= 0) {
            T[e.u].eb(e.v, e.w);
            T[e.v].eb(e.u, e.w);
        }
        rep(i, n) if (not done[i]) {
            q.emplace(e.v, i, A[e.v][i]);
        }
    }
    return T;
}
```

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.

```
int minimum_steiner_tree(vi &T, vvi &g){
    int n = g.size(), t = T.size();
    if(t <= 1) return 0;
    vvi d(g);    // all-pair shortest
    rep(k,n)rep(i,n)rep(j,n)    //Warshall Floyd
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

    int opt[1 << t][n];
    rep(S,1<<t) rep(x,n)
        opt[S][x] = INF;

    rep(p,t) rep(q,n)   // trivial case
        opt[1 << p][q] = d[T[p]][q];

    repi(S,1,1<<t){   // DP step
        if(!(S & (S-1))) continue;
        rep(p,n) rep(E,S)
            if((E | S) == S)
                opt[S][p] = min(opt[S][p], opt[E][p] + opt[S-E][p]);
        rep(p,n) rep(q,n)
            opt[S][p] = min(opt[S][p], opt[S][q] + d[p][q]);
    }

    int ans = INF;
    rep(S,1<<t) rep(q,n)
        ans = min(ans, opt[S][q] + opt[((1<<t)-1)-S][q]);
    return ans;
}
```

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)

```
const int MAX_V=16;
const int mod = 10009;
int N[MAX_V], I[1<<MAX_V], V;
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);}

bool can(int k){
    int res = 0;
    rep(S, 1<<V){
        if(__builtin_popcountll(S)%2) res -= mpow(I[S], k);
        else res += mpow(I[S],k);
    }
    return (res%mod+mod)%mod;
}

int color_number(){
    memset(I, 0, sizeof(I));
    I[0] = 1;
    repi(S,1,1<<V){
        int v = 0;
        while(!(S&(1<<v))) v++;
        I[S] = I[S-(1<<v)] + I[S&(~N[v])];
    }
    int lb = 0, ub = V, mid;
    while(ub-lb>1){
        mid = (lb+ub)/2;
        if(can(mid)) ub = mid;
        else lb = mid;
    }
    return ub;
}
```

4 数学

4.1 整数

4.1.1 剰余

```
// (x, y) s.t. a x + b y = gcd(a, b)
long extgcd(long a, long b, long& x, long& y) {
    long g = a; x = 1, y = 0;
    if (b != 0) g = extgcd(b, a % b, y, x), y -= (a / b) * x;
    return g;
}

// repi(i, 2, n) mod_inv[i] = mod_inv[m % i] * (m - m / i) % m
long mod_inv(long a, long m) {
    long x, y;
    if (extgcd(a, m, x, y) != 1) return 0;
    return (x % m + m) % m;
}

// a mod p where n! = a p^e in O(log_p n)
long mod_fact(long n, long p, long& e) {
    const int P = 1000010;
    static long fact[P] = {1};
    static bool done = false;
    if (not done) {
        repi(i, 1, P) fact[i] = fact[i - 1] * i % p;
        done = true;
    }
    e = 0;
    if (n == 0) return 1;
    long ret = mod_fact(n / p, p, e);
    e += n / p;
    if (n / p % 2) return ret * (p - fact[n % p]) % p;
    return ret * fact[n % p] % p;
}

// nCk mod p
long mod_binom(long n, long k, long p) {
    if (k < 0 or n < k) return 0;
    long e1, e2, e3;
    long a1 = mod_fact(n, p, e1);
    long a2 = mod_fact(k, p, e2);
    long a3 = mod_fact(n - k, p, e3);
    if (e1 > e2 + e3) return 0;
    return a1 * mod_inv(a2 * a3 % p, p) % p;
}

// a^b mod m
```

```
long mod_pow(long a, long b, long m) {
    long ret = 1;
    do {
        if (b & 1) ret = ret * a % m;
        a = a * a % m;
    } while (b >>= 1);
    return ret;
}
```

4.1.2 カタラン数

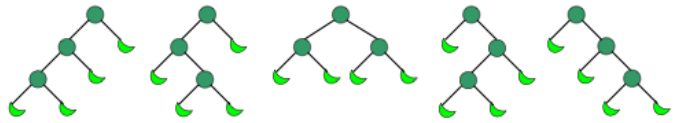
$n \leq 16$ 程度が限度. $n \geq 1$ について以下が成り立つ.

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

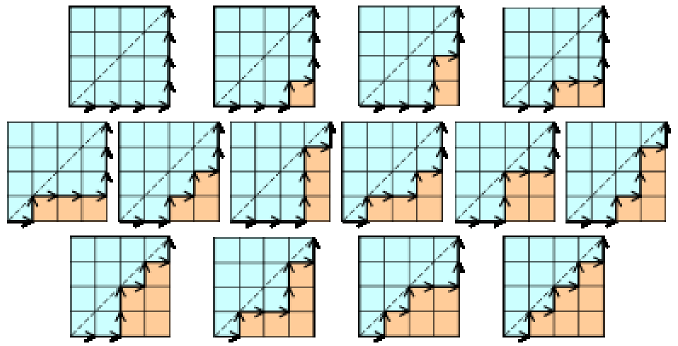
n が十分大きいとき, カタラン数は以下に近似できる.

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

() 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.
$C_3 = 5$



$C_4 = 14$



4.2 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.

```
typedef complex<double> cd;
vector<cd> fft(vector<cd> f, bool inv){
    int n, N = f.size();
    for(n=0;;n++) if(N == (1<<n)) break;
    rep(m,N){
        int m2 = 0;
        rep(i,n) if(m&(1<<i)) m2 |= (1<<(n-1-i));
        if(m < m2) swap(f[m], f[m2]);
    }

    for(int t=1;t<N;t*=2){
        double theta = acos(-1.0) / t;
        cd w(cos(theta), sin(theta));
        if(inv) w = cd(cos(theta), -sin(theta));
        for(int i=0;i<N;i+=2*t){
            cd power(1.0, 0.0);
            rep(j,t){
                cd tmp1 = f[i+j] + f[i+t+j] * power;
                cd tmp2 = f[i+j] - f[i+t+j] * power;
                f[i+j] = tmp1;
                f[i+t+j] = tmp2;
                power = power * w;
            }
        }
    }
    if(inv) rep(i,N) f[i] /= N;
    return f;
}
```

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.

```
#include "number_theory.cpp"

const int mod = 7*17*(1<<23)+1;
vector<int> fmt(vector<int> f, bool inv){
    int e, N = f.size();
    // assert((N&(N-1))==0 and "f.size() must be power of 2");
    for(e=0;;e++) if(N == (1<<e)) break;
    rep(m,N){
        int m2 = 0;
        rep(i,e) if(m&(1<<i)) m2 |= (1<<(e-1-i));
        if(m < m2) swap(f[m], f[m2]);
    }
    for(int t=1; t<N; t*=2){
        int r = pow_mod(3,(mod-1)/(t*2),mod);
        if(inv) r = mod_inverse(r,mod);
        for(int i=0; i<N; i+=2*t){
            int power = 1;
            rep(j,t){
                int x = f[i+j], y = 1LL*f[i+t+j]*power%mod;
                f[i+j] = (x+y)%mod;
                f[i+t+j] = (x-y+mod)%mod;
                power = 1LL*power*r%mod;
            }
        }
    }
    if(inv) for(int i=0,ni=mod_inv(N,mod);i<N;i++) f[i] = 1LL*f[i]*ni%mod;
    return f;
}
```

4.2.3　積 (FMT)

$O(N \log N)$
poly_mul() が必要.

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

4.2.4　逆元 (FMT)

$O(N \log N)$
extgcd(), *mod_inverse*(), *poly_mul*(), *fmt*() が必要.

```
vector<int> poly_inv(const vector<int> &f){
    int N = f.size();
    vector<int> r(1,mod_inv(f[0],mod));
    for(int k = 2; k <= N; k <<= 1){
        vector<int> nr = poly_mul(poly_mul(r,r), vector<int>(f.begin(),f.begin()+k));
        nr.resize(k);
        rep(i,k/2) {
            nr[i] = (2*r[i]-nr[i]+mod)%mod;
            nr[i+k/2] = (mod-nr[i+k/2])%mod;
        }
        r = nr;
    }
    return r;
}
```

4.2.5　平方根 (FMT)

$O(NlogN)$
extgcd(), *mod_inverse*(), *poly_inv*(), *poly_mul*(), *fmt*() が必要.

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

4.3　行列

C++11 だと array という名前では衝突するので arr にしている.

```
typedef double number;
typedef vector<number> vec;
typedef vector<vec> mat;

vec mul(const mat& A, const vec& x) {
    const int n = A.size();
    vec b(n);
    rep(i, n) rep(j, A[0].size()) {
        b[i] = A[i][j] * x[j];
    }
    return b;
}

mat mul(const mat& A, const mat& B) {
    const int n = A.size();
    const int o = A[0].size();
    const int m = B[0].size();
    mat C(n, vec(m));
    rep(i, n) rep(k, o) rep(j, m) {
        C[i][j] += A[i][k] * B[k][j];
    }
    return C;
}

mat pow(mat A, long m) {
    const int n = A.size();
    mat B(n, vec(n));
    rep(i, n) B[i][i] = 1;
    do {
        if (m & 1) B = mul(B, A);
        A = mul(A, A);
    } while (m >>= 1);
    return B;
}

const number eps = 1e-4;

// determinant; O(n^3)
number det(mat A) {
    int n = A.size();
    number D = 1;
    rep(i,n){
        int pivot = i;
        repi(j,i+1,n)
            if (abs(A[j][i]) > abs(A[pivot][i])) pivot = j;
        swap(A[pivot], A[i]);
        D *= A[i][i] * (i != pivot ? -1 : 1);
        if (abs(A[i][i]) < eps) break;
        repi(j,i+1,n)
            for(int k=n-1;k>=i;--k)
                A[j][k] -= A[i][k] * A[j][i] / A[i][i];
    }
    return D;
}

// rank; O(n^3)
int rank(mat A) {
    int n = A.size(), m = A[0].size(), r = 0;
    for(int i = 0; i < m and r < n; i++){
        int pivot = r;
        repi(j,r+1,n)
            if (abs(A[j][i]) > abs(A[pivot][i])) pivot = j;
        swap(A[pivot], A[r]);
        if (abs(A[r][i]) < eps) continue;
        for(int k=m-1;k>=i;--k)
            A[r][k] /= A[r][i];
        repi(j,r+1,n) repi(k,i,m)
            A[j][k] -= A[r][k] * A[j][i];
        ++r;
    }
    return r;
}
```

4.3.1 線形方程式の解 (Givens 消去法)

$O(N^3)$

```
// Givens elimination; O(n^3)

typedef double number;
typedef vector<vector<number> > matrix;

inline double my_hypot(double x, double y) { return sqrt(x * x + y * y); }
inline void givens_rotate(number& x, number& y, number c, number s) {
    number u = c * x + s * y, v = -s * x + c * y;
    x = u, y = v;
}
vector<number> givens(matrix A, vector<number> b) {
    const int n = b.size();
    rep(i, n) repi(j, i + 1, n) {
        const number r = my_hypot(A[i][i], A[j][i]);
        const number c = A[i][i] / r, s = A[j][i] / r;
        givens_rotate(b[i], b[j], c, s);
        repi(k, i + 1, n) givens_rotate(A[i][k], A[j][k], c, s);
    }
    for (int i = n - 1; i >= 0; --i) {
        repi(j, i + 1, n) b[i] -= A[i][j] * b[j];
        b[i] /= A[i][i];
    }
    return b;
}
```

5 幾何

```
// constants and eps-considered operators

const double eps = 1e-8; // choose carefully!
const double pi = acos(-1.0);

inline bool lt(double a, double b) { return a < b - eps; }
inline bool gt(double a, double b) { return lt(b, a); }
inline bool le(double a, double b) { return !lt(b, a); }
inline bool ge(double a, double b) { return !lt(a, b); }
inline bool ne(double a, double b) { return lt(a, b) or lt(b, a); }
inline bool eq(double a, double b) { return !ne(a, b); }

// points and lines

typedef complex<double> point;

inline double dot  (point a, point b) { return real(conj(a) * b); }
inline double cross(point a, point b) { return imag(conj(a) * b); }

struct line {
    point a, b;
    line(point a, point b) : a(a), b(b) {}
};

/*
 *  Here is what ccw(a, b, c) returns:
 *
 *          1
 *  ------------------
 *    2 |a  0  b| -2
 *  ------------------
 *         -1
 *
 *  Note: we can implement intersectPS(p, s) as !ccw(s.a, s.b, p).
 */
int ccw(point a, point b, point c) {
    b -= a, c -= a;
    if (cross(b, c) > eps)    return +1;
    if (cross(b, c) < eps)    return -1;
    if (dot(b, c) < eps)      return +2; // c -- a -- b
    if (lt(norm(b), norm(c))) return -2; // a -- b -- c
    return 0;
}
bool intersectLS(const line& l, const line& s) {
    return ccw(l.a, l.b, s.a) * ccw(l.a, l.b, s.b) <= 0;
}
bool intersectSS(const line& s, const line& t) {
    return intersectLS(s, t) and intersectLS(t, s);
}
bool intersectLL(const line& l, const line& m) {
    return ne(cross(l.b - l.a, m.b - m.a), 0.0)  // not parallel
        or eq(cross(l.b - l.a, m.a - l.a), 0.0); // overlap
}
point crosspointLL(const line& l, const line& m) {
    double A = cross(l.b - l.a, m.b - m.a);
    double B = cross(l.b - l.a, m.a - l.a);
    if (eq(A, 0.0) and eq(B, 0.0)) return m.a; // overlap
    assert(ne(A, 0.0));                         // not parallel
    return m.a - B / A * (m.b - m.a);
}
point proj(const line& l, point p) {
    double t = dot(l.b - l.a, p - l.a) / norm(l.b - l.a);
    return l.a + t * (l.b - l.a);
}
point reflection(const line& l, point p) { return 2.0 * proj(l, p) - p; }

// distances (for shortest path)

double distanceLP(const line& l, point p) { return abs(proj(l, p) - p); }
double distanceLL(const line& l, const line& m) {
    return intersectLL(l, m) ? 0.0 : distanceLP(l, m.a);
}
double distanceLS(const line& l, const line& s) {
    return intersectLS(l, s) ? 0.0 : min(distanceLP(l, s.a), distanceLP(l, s.b));
}
double distancePS(point p, const line& s) {
    point h = proj(s, p);
    return ccw(s.a, s.b, h) ? min(abs(s.a - p), abs(s.b - p)) : abs(h - p);
}
double distanceSS(const line& s, const line& t) {
    if (intersectSS(s, t)) return 0.0;
    return min(min(distancePS(s.a, t), distancePS(s.b, t)),
               min(distancePS(t.a, s), distancePS(t.b, s)));
}

// circles
```

```cpp

struct circle {
    point o; double r;
    circle(point o, double r) : o(o), r(r) {}
};

bool intersectCL(const circle& c, const line& l) {
    return le(norm(proj(l, c.o) - c.o), c.r * c.r);
}
int intersectCS(const circle& c, const line& s) {
    if (not intersectCL(c, s)) return 0;
    double a = abs(s.a - c.o);
    double b = abs(s.b - c.o);
    if (lt(a, c.r) and lt(b, c.r)) return 0;
    if (lt(a, c.r) or lt(b, c.r)) return 1;
    return ccw(s.a, s.b, proj(s, c.o)) ? 0 : 2;
}
bool intersectCC(const circle& c, const circle& d) {
    double dist = abs(d.o - c.o);
    return le(abs(c.r - d.r), dist) and le(dist, c.r + d.r);
}
line crosspointCL(const circle& c, const line& l) {
    point h = proj(l, c.o);
    double a = sqrt(c.r * c.r - norm(h - c.o));
    point d = a * (l.b - l.a) / abs(l.b - l.a);
    return line(h - d, h + d);
}
line crosspointCC(const circle& c, const circle& d) {
    double dist = abs(d.o - c.o), th = arg(d.o - c.o);
    double ph = acos((c.r * c.r + dist * dist - d.r * d.r) / (2.0 * c.r * dist));
    return line(c.o + polar(c.r, th - ph), c.o + polar(c.r, th + ph));
}

line tangent(const circle& c, double th) {
    point h = c.o + polar(c.r, th);
    point d = polar(c.r, th) * point(0, 1);
    return line(h - d, h + d);
}
vector<line> common_tangents(const circle& c, const circle& d) {
    vector<line> ret;
    double dist = abs(d.o - c.o), th = arg(d.o - c.o);
    if (abs(c.r - d.r) < dist) { // outer
        double ph = acos((c.r - d.r) / dist);
        ret.pb(tangent(c, th - ph));
        ret.pb(tangent(c, th + ph));
    }
    if (abs(c.r + d.r) < dist) { // inner
        double ph = acos((c.r + d.r) / dist);
        ret.pb(tangent(c, th - ph));
        ret.pb(tangent(c, th + ph));
    }
    return ret;
}
pair<circle, circle> tangent_circles(const line& l, const line& m, double r) {
    double th = arg(m.b - m.a) - arg(l.b - l.a);
    double ph = (arg(m.b - m.a) + arg(l.b - l.a)) / 2.0;
    point p = crosspointLL(l, m);
    point d = polar(r / sin(th / 2.0), ph);
    return mp(circle(p - d, r), circle(p + d, r));
}
line bisector(point a, point b);
circle circum_circle(point a, point b, point c) {
    point o = crosspointLL(bisector(a, b), bisector(a, c));
    return circle(o, abs(a - o));
}

// polygons

typedef vector<point> polygon;

double area(const polygon& g) {
    double ret = 0.0;
    int j = g.size() - 1;
    rep(i, g.size()) {
        ret += cross(g[j], g[i]), j = i;
    }
    return ret / 2.0;
}
point centroid(const polygon& g) {
    if (g.size() == 1) return g[0];
    if (g.size() == 2) return (g[0] + g[1]) / 2.0;
    point ret = 0.0;
    int j = g.size() - 1;
    rep(i, g.size()) {
        ret += cross(g[j], g[i]) * (g[j] + g[i]), j = i;
    }
    return ret / area(g) / 6.0;
}
line bisector(point a, point b) {
    point m = (a + b) / 2.0;
    return line(m, m + (b - a) * point(0, 1));
}
polygon convex_cut(const polygon& g, const line& l) {
    polygon ret;
    int j = g.size() - 1;
    rep(i, g.size()) {
        if (ccw(l.a, l.b, g[j]) != -1) ret.pb(g[j]);
        if (intersectLS(l, line(g[j], g[i]))) ret.pb(crosspointLL(l, line(g[j], g[i])));
        j = i;
    }
    return ret;
}
polygon voronoi_cell(polygon g, const vector<point>& v, int k) {
    rep(i, v.size()) if (i != k) {
        g = convex_cut(g, bisector(v[i], v[k]));
    }
    return g;
}
```

```cpp
#include "geometry.cpp"

namespace std {
    bool operator <(const point& a, const point& b) {
        return ne(real(a), real(b)) ? lt(real(a), real(b)) : lt(imag(a), imag(b));
    }
}

polygon convex_hull(vector<point> v) {
    const int n = v.size();
    sort(all(v));
    polygon ret(2 * n);
    int k = 0;
    for (int i = 0; i < n; ret[k++] = v[i++]) {
        while (k >= 2 and ccw(ret[k - 2], ret[k - 1], v[i]) <= 0) --k;
    }
    for (int i = n - 2, t = k + 1; i >= 0; ret[k++] = v[i--]) {
        while (k >= t and ccw(ret[k - 2], ret[k - 1], v[i]) <= 0) --k;
    }
    ret.resize(k - 1);
    return ret;
}
```

6 データ構造

6.1 Union-Find 木

```cpp
#include "macro.cpp"

class disjoint_set {
    vector<int> p;
    int root(int i) { return p[i] >= 0 ? p[i] = root(p[i]) : i; }
public:
    disjoint_set(int n) : p(n, -1) {}
    bool same(int i, int j) { return root(i) == root(j); }
    int size(int i) { return -p[root(i)]; }
    void merge(int i, int j) {
        i = root(i), j = root(j);
        if (i == j) return;
        if (p[i] > p[j]) swap(i, j);
        p[i] += p[j], p[j] = i;
    }
};
```

6.2 赤黒木

```cpp
template<class T> class rbtree {
public:
    enum COL { BLACK, RED,};
    struct node {
        T val;
        int color;
        int rnk, size;
        node *left, *right;

        node(){}
        node(T v) : val(v), color(BLACK), rnk(0), size(1) {
            left = right = NULL;
        }
        node(node *l, node *r, int c) : color(c) {
            left = l;
            right = r;
            update();
        }
        void update() {
            rnk = max((left? left->rnk+(left->color==BLACK): 0),
                      (right? right->rnk+(right->color==BLACK): 0));
            size = (left? left->size: 0)+(right? right->size: 0)+(!left and !right);
        }
    };

    node *root;

    rbtree() { root = NULL;}
    rbtree(T val) { root = new_node(val);}

    node *new_node(T v) { return new node(v);}
    node *new_node(node *l, node *r, int c) { return new node(l,r,c);}

    node *right_rotate(node *v) {
        node *w = v->left;
        v->left = w->right;
        w->right = v;
        v->left->update();
        v->update();
        w->right->update();
        v->color = RED;
        w->color = BLACK;
        return w;
    }

    node *left_rotate(node *v) {
        node *w = v->right;
        v->right = w->left;
        w->left = v;
        v->right->update();
        v->update();
        w->left->update();
        v->color = RED;
        w->color = BLACK;
        return w;
    }

    node *merge_sub(node *u, node *v) {
        if(u->rnk < v->rnk) {
            node *w = merge_sub(u,v->left);
            v->left = w;
            v->update();
            if(v->color == BLACK and w->color == RED and w->left->color == RED) {
                if(v->right->color == BLACK)  return right_rotate(v);
                else {
                    v->color = RED;
                    v->right->color = BLACK;
                    w->color = BLACK;
                    return v;
                }
            }
            else return v;
        }
        else if(u->rnk > v->rnk) {
            node *w = merge_sub(u->right,v);
            u->right = w;
            u->update();
            if(u->color == BLACK and w->color == RED and w->right->color == RED) {
                if(u->left->color == BLACK) return left_rotate(u);
                else {
                    u->color = RED;
                    u->left->color = BLACK;
                    w->color = BLACK;
                    return u;
                }
            }
            else return u;
        }
        else return new_node(u,v,RED);
    }

    node *merge(node *u, node *v) {
        if(!u) return v;
        if(!v) return u;
        u = merge_sub(u,v);
        u->color = BLACK;
        return u;
    }

    pair<node*,node*> split(node *v, int k) {
        if(!k) return pair<node*,node*>(NULL,v);
        if(k == v->size) return pair<node*,node*>(v,NULL);
        if(k < v->left->size) {
            auto p = split(v->left,k);
            return pair<node*,node*>(p.first,merge(p.second,v->right));
        }
        else if(k > v->left->size) {
            auto p = split(v->right,k-v->left->size);
            return pair<node*,node*>(merge(v->left,p.first),p.second);
        }
        else return pair<node*,node*>(v->left,v->right);
    }

    // insert val at k
    node *insert(T val, int k) { return insert(new_node(val),k);}
    // insert tree v at k
    node *insert(node *v, int k) {
        auto p = split(root,k);
        return root = merge(merge(p.first,v),p.second);
    }

    // delete at k
    node *erase(int k) {
        auto p = split(root,k+1);
        return root = merge(split(p.first,k).first, p.second);
    }

    node *build(const vector<T> &vs) {
        if(!vs.size()) return NULL;
        if((int)vs.size() == 1) return new_node(vs[0]);
        int m = vs.size()/2;
        return merge(build(vector<T>(begin(vs),begin(vs)+m)),
                     build(vector<T>(begin(vs)+m,end(vs))));
    }

    int size() { return root->size;}

    void get(vector<T> &vs) { get(root,vs);}
    void get(node *v, vector<T> &vs) {
        if(!v->left and !v->right) vs.push_back(v->val);
        else {
            if(v->left) get(v->left,vs);
            if(v->right) get(v->right,vs);
        }
    }

    node *push_back(T val) {
        node *v = new_node(val);
        return root = merge(root,v);
    }
};
```

6.3 永続赤黒木

```
//const int MAX = 15000000, BOUND = 14000000;
template<class T> class prbtree {
public:
    enum COL { BLACK, RED,};
    struct node {
        T val;
        int color;
        int rnk, size;
        node *left, *right;

        node(){}
        node(T v) : val(v), color(BLACK), rnk(0), size(1) {
            left = right = NULL;
        }
        node(node *l, node *r, int c) : color(c) {
            left = l;
            right = r;
            rnk = max((l? l->rnk+(l->color==BLACK): 0),
                      (r? r->rnk+(r->color==BLACK): 0));
            size = !l and !r? 1: !l? r->size: !r? r->size: l->size+r->size;
        }
    };

    node *root;
//        node nodes[MAX];
//        int called;

    prbtree() {
        root = NULL;
        // called = 0;
    }

    prbtree(T val) {
        root = new_node(val);
        // called = 0;
    }

    // node *new_node(T v) { return &(nodes[called++] = node(v));}
    // node *new_node(node *l, node *r, int c) { return &(nodes[called++] = node(l,r,c
         ));}
    node *new_node(T v) { return new node(v);}
    node *new_node(node *l, node *r, int c) { return new node(l,r,c);}

    node *merge_sub(node *u, node *v) {
        if(u->rnk < v->rnk) {
            node *w = merge_sub(u,v->left);
            if(v->color == BLACK and w->color == RED and w->left->color == RED){
                if(v->right->color == BLACK)  return new_node(w->left,new_node(w->right,
                     v->right,RED),BLACK);
                else return new_node(new_node(w->left,w->right,BLACK),new_node(v->right
                     ->left,v->right->right,BLACK),RED);
            }
            else return new_node(w,v->right,v->color);
        }
        else if(u->rnk > v->rnk) {
            node *w = merge_sub(u->right,v);
            if(u->color == BLACK and w->color == RED and w->right->color == RED){
                if(u->left->color == BLACK)  return new_node(new_node(u->left,w->left,
                     RED),w->right,BLACK);
                else return new_node(new_node(u->left->left,u->left->right,BLACK),
                     new_node(w->left,w->right,BLACK),RED);
            }
            else return new_node(u->left,w,u->color);
        }
        else return new_node(u,v,RED);
    }

    node *merge(node *u, node *v) {
        if(!u) return v;
        if(!v) return u;
        u = merge_sub(u,v);
        if(u->color == RED) return new_node(u->left,u->right,BLACK);
        return u;
    }

    pair<node*,node*> split(node *v, int k) {
        if(!k) return pair<node*,node*>(NULL,v);
        if(k == v->size) return pair<node*,node*>(v,NULL);
        if(k < v->left->size) {
            auto p = split(v->left,k);
            return pair<node*,node*>(p.first,merge(p.second,v->right));
        }
        else if(k > v->left->size) {
            auto p = split(v->right,k-v->left->size);
            return pair<node*,node*>(merge(v->left,p.first),p.second);
        }
        else return pair<node*,node*>(v->left,v->right);
    }

    node *build(const vector<T> &vs) {
        if(!vs.size()) return NULL;
        if((int)vs.size() == 1) return new_node(vs[0]);
        int m = vs.size()/2;
        return merge(build(vector<T>(begin(vs),begin(vs)+m)), build(vector<T>(begin(vs)+
             m,end(vs))));
    }

    int size() { return root->size;}

    void get(vector<T> &vs) { get(root,vs);}
    void get(node *v, vector<T> &vs) {
        if(!v->left and !v->right) vs.push_back(v->val);
        else {
            if(v->left) get(v->left,vs);
            if(v->right) get(v->right,vs);
        }
    }

    node *push_back(T val) {
        node *v = new_node(val);
        return root = merge(root,v);
    }

    // insert leaf at k
    node *insert(int k, T val) {
        return insert(new_node(val), k);
    }

    // insert tree v at k
    node *insert(node *v, int k) {
        auto p = split(root,k);
        return root = merge(merge(p.first,v),p.second);
    }

    // copy [l,r)
    node *copy(int l, int r) {
        return split(split(root, l).second, r-l).first;
    }
    // copy and insert [l,r) at k
    node *copy_paste(int l, int r, int k) {
        return insert(copy(l,r),k);
    }
};
```