Contents

1 Basic

1.1 Default [f4b5b2]

```
#include <bits/stdc++.h>
using namespace std;
#define pb emplace_back
#define iter(x) (x).begin(), (x).end()
#define size(x) (int)(x).size()
#define INF 0x3f3f3f3f
#define tmin(a, b) (a) = min((a), (b))
#define tmax(a, b) (a) = max((a), (b))
typedef long long ll;
typedef pair<int, int> pii;

void db() { cerr << endl; }
template <class T, class... U>
void db(T a, U... b) { cerr << a << " ", db(b...); }

signed main() {
    cin.tie(0)->sync_with_stdio(0);
}
```

```
setxkbmap -option caps:swapescape #caps to esc
cpp file.cpp -dD -P -fpreprocessed |
    tr -d '[:space:]' | md5sum | cut -c-6 #hash command
```

```
#!/usr/bin/bash
for ((i=0;;i++))
do
python gen.py > case.in
./A < case.in > aout
./B < case.in > bout

if ! (cmp -s aout bout);
then
cat case.in
fi

done
```

2 Geometry

2.1 Template [b71649]

```
#define eps 1e-9
struct P
{
    double x, y;
    P() { x = y = 0; }
    P(double x, double y) : x(x), y(y) {}
    friend bool operator<(const P &a, const P &b)
        { return a.x == b.x ? a.y < b.y : a.x < b.x; }
    friend bool operator==(const P &a, const
        P &b) { return a.x == b.x && a.y == b.y; }
    friend bool operator!=(const P &a, const
        P &b) { return a.x != b.x || a.y != b.y; }
    P operator+(const
        P &b) const { return P(x + b.x, y + b.y); }
    void operator+=(const P &b) { x += b.x, y += b.y; }
    P operator-(const
        P &b) const { return P(x - b.x, y - b.y); }
    void operator-=(const P &b) { x -= b.x, y -= b.y; }
    P operator
        *(double b) const { return P(x * b, y * b); }
    void operator*=(double b) { x *= b, y *= b; }
    P operator
        /(double b) const { return P(x / b, y / b); }
    void operator/=(double b) { x /= b, y /= b; }
    double operator*(
        const P &b) const { return x * b.x + y * b.y; }
    double operator^(
        const P &b) const { return x * b.y - y * b.x; }
    double lth() const { return sqrt(x * x + y * y); }
};
ostream &operator<<(ostream &os, const P &a)
{
    return os << a.x << ' ' << a.y << '|';
}
int ori(const P &a, const P &b, const P &c)
{
    double k = (b - a) ^ (c - a);
    if (-eps < k && k < eps)
        return 0;
    return k > 0 ? 1 : -1;
}
inline bool ud(const P &a)
{
    if (-eps < a.y && a.y < eps)
        return a.x > eps;
    return a.y > eps;
}
bool cmp(const P &a, const P &b)
{
  P bs(0, 0);
    bool ba = ud(a), bb = ud(b);
    if (ba ^ bb)
        return ba;
    return ori(bs, a, b) > 0;
};
bool within(const P &a, const P &b, const P &c)
{
    return (b - a) * (c - a) < eps;
}
bool
    its(const P &a, const P &b, const P &c, const P &d)
{
    int abc = ori(a, b, c);
    int abd = ori(a, b, d);
    int cda = ori(c, d, a);
    int cdb = ori(c, d, b);
    if (!abc && !abd)
        return within(a, c, d) || within(b, c, d) ||
               within(c, a, b) || within(d, a, b);
    return abc * abd <= 0 && cda * cdb <= 0;
}

P itp(const P &a, const P &b, const P &c, const P &d)
{
    double abc = (b - a) ^ (c - a);
    double abd = (b - a) ^ (d - a);
    return (d * abc - c * abd) / (abc - abd);
}
void fdhl(vector<P> &ar, vector<P> &hl, int lnar)
{
    int lnhl;
    for (int i = 0; i < 2; i++)
    {
        int prln = hl.size();
        for (int j = 0; j < lnar; j++)
        {
            lnhl = hl.size();
            while (lnhl - prln > 1 && ori(hl
                [lnhl - 1], hl[lnhl - 2], ar[j]) >= 0)
            {
                lnhl--;
                hl.pop_back();
            }
            hl.push_back(ar[j]);
        }
        if (hl.size() > 1)
            hl.pop_back();
        reverse(ar.begin(), ar.end());
    }
    if (hl.size() > 1 && hl.front() == hl.back())
        hl.pop_back();
}
bool in(const P &a, vector<P> &hl)
{
    int ln = hl.size();
```

```
    if (ln == 1)
        return a == hl[0];
    if (ln == 2)
        return within(a, hl[0], hl[1]);
    int l = 1, r = ln - 1, m;
    while (r - l > 1)
    {
        m = (l + r) >> 1;
        if (ori(hl[0], a, hl[m]) < 0)
            l = m;
        else
            r = m;
    }
    return ori(hl[0], hl[l], a) >= 0 && ori(hl[l
        ], hl[r], a) >= 0 && ori(hl[r], hl[0], a) >= 0;
}
```

2.2 Minkowski [542fed]

```
//need Geometry template
void reorder_polygon(vector<P> & pt){
    int pos = 0;
    for(int i = 1; i < size(pt); i++){
        if(pt[i].y < pt[pos].y || (pt
            [i].y == pt[pos].y && pt[i].x < pt[pos].x))
            pos = i;
    }
    rotate(pt.begin(), pt.begin() + pos, pt.end());
}
vector<P> minkowski(vector<P> A, vector<P> B){
    reorder_polygon(A);
    reorder_polygon(B);
    A.push_back(A[0]);A.push_back(A[1]);
    B.push_back(B[0]);B.push_back(B[1]);
    vector<P> result;
    int i = 0, j = 0;
    while(i < size(A) - 2 || j < size(B) - 2){
        result.push_back(A[i] + B[j]);
        auto cross
             = (A[i + 1] - A[i])^(B[j + 1] - B[j]);
        if(cross >= 0 && i < size(A) - 2)
            ++i;
        if(cross <= 0 && j < size(B) - 2)
            ++j;
    }
    return result;
}
```

3 Graph

3.1 Block Cut [81081b]

```
struct BCC
{
  int n, dft, nbcc;
  vector<pii> Edges;
  vector<int>
      low, dfn, bln, stk, is_ap, bln_, BCCofE, bct_vst;
  vector<vector<pii>> G;
  vector<vector<int>> bcc, nG;
  void make_bcc(int u)
  {
    bcc.emplace_back(1, u);
    for (; stk.back() != u; stk.pop_back())
      bln[stk.back()
          ] = nbcc, bcc[nbcc].emplace_back(stk.back());
    stk.pop_back(), bln[u] = nbcc++;
  }
  void dfs(int u, int f)
  {
    int child = 0;
    low[u] = dfn[u] = ++dft, stk.emplace_back(u);
    for (auto [v, id] : G[u])
      if (!dfn[v])
      {
        dfs(v, u), ++child;
        low[u] = min(low[u], low[v]);
        if (dfn[u] <= low[v])
        {
          is_ap[u] = 1, bln[u] = nbcc;
          make_bcc(v), bcc.back().emplace_back(u);
        }
      }
      else if (dfn[v] < dfn[u] && v != f)
        low[u] = min(low[u], dfn[v]);
    if (f == -1 && child < 2)
      is_ap[u] = 0;
    if (f == -1 && child == 0)
      make_bcc(u);
  }
  BCC(int _n) : n(_n), dft(0), nbcc
      (0), low(n), dfn(n), bln(n), is_ap(n), G(n) {}
  void solve()
  {
    for (int i = 0; i < size(Edges); ++i)
    {
      G[Edges
          [i].first].emplace_back(Edges[i].second, i);
      G[Edges
          [i].second].emplace_back(Edges[i].first, i);
    }
    for (int i = 0; i < n; ++i)
      if (!dfn[i])
        dfs(i, -1);
  }
  void block_cut_tree()
  {
    int tmp = nbcc;
    for (int i = 0; i < n; ++i)
      if (is_ap[i])
        bln[i] = tmp++;
    bln_.resize(tmp);
    for (int i = 0; i < n; ++i)
      bln_[bln[i]] = i;
    nG.assign(tmp, vector<int>(0));
    for (int i = 0; i < nbcc; ++i)
      for (int j : bcc[i])
        if (is_ap[j])
          nG[i].emplace_back
              (bln[j]), nG[bln[j]].emplace_back(i);
  } // up to 2 * n - 2 nodes!! bln[i] for id
  void dfs_bct(int cur, int fa)
  {
    if (cur < nbcc)
      bct_vst[cur] = 1;
    for (auto i : nG[cur])
    {
      if (i == fa)
        continue;
      if (cur < nbcc)
      {
        for (auto [j, id] : G[bln_[i]])
        {
          auto it = lower_bound(iter(bcc[cur]), j);
          if (it != bcc[cur].end() && *it == j)
            BCCofE[id] = cur;
        }
      }
      else
      {
        for (int j : bcc[i])
        {
          auto it = lower_bound
              (iter(G[bln_[cur]]), make_pair(j, 0));
          if (it
              != G[bln_[cur]].end() && it->first == j)
            BCCofE[it->second] = i;
        }
      }
      dfs_bct(i, cur);
    }
  }
  void find_BCCofE()
  {
```

```
    for (auto &i : G)
      sort(iter(i));
    for (auto &i : bcc)
      sort(iter(i));
    BCCofE.assign
        (size(Edges), -1), bct_vst.assign(nbcc, 0);
    for (int i = 0; i < nbcc; ++i)
      if (!bct_vst[i])
        dfs_bct(i, i);
    for (int i = 0; i < size(Edges); ++i)
      if (is_ap
          [Edges[i].first] || is_ap[Edges[i].second])
        assert(BCCofE[i] != -1);
      else
      {
        assert(BCCofE[i] == -1);
        BCCofE[i] = bln[Edges[i].first];
      }
  }
};
```

3.2 Centroid Decomp [314317]

```
struct CentroidDecomposition {
    vector<vector<int> > g;
    vector<int> sub;
    vector<bool> v;
    vector<vector<int>> tree;
    int root;
    void add_edge(int a, int b) {
        g[a].push_back(b);
        g[b].push_back(a);
    }
    CentroidDecomposition(const vector<vector
        <int> > &g_, int isbuild = true) : g(g_) {
        sub.resize(size(g), 0);
        v.resize(size(g), false);
        if (isbuild) build();
    }

    void build() {
        tree.resize(size(g));
        root = build_dfs(0);
    }

    int get_size(int cur, int par) {
        sub[cur] = 1;
        for (auto &dst : g[cur]) {
            if (dst == par || v[dst]) continue;
            sub[cur] += get_size(dst, cur);
        }
        return sub[cur];
    }

    int get_centroid(int cur, int par, int mid) {
        for (auto &dst : g[cur]) {
            if (dst == par || v[dst]) continue;
            if (sub[dst] > mid
                ) return get_centroid(dst, cur, mid);
        }
        return cur;
    }

    int build_dfs(int cur) {
        int centroid = get_centroid
            (cur, -1, get_size(cur, -1) / 2);
        v[centroid] = true;
        for (auto &dst : g[centroid]) {
            if (!v[dst]) {
                int nxt = build_dfs(dst);
                if (centroid != nxt
                    ) tree[centroid].emplace_back(nxt);
            }
        }
        v[centroid] = false;
        return centroid;
    }
};
```

3.3 Dominater Tree [602345]

```
struct DOT {
    static const int N = 2e5 + 5; // change
    int dfn
        [N], id[N], dfc, fa[N], idm[N], sdm[N], bst[N];
    vector<int> G[N], rG[N];
    void ini(int n) { // remember to initialize
        for (int i = 1; i <= n; i++)
            G[i].clear(), rG[i].clear();
        fill(dfn, dfn + n + 1, 0);
    }
    inline void addedge
        (int u, int v) { G[u].pb(v), rG[v].pb(u); }
    int f(int x, int lm) {
        if (x <= lm)
            return x;
        int cr = f(fa[x], lm);
        if (sdm[bst[fa[x]]] < sdm[bst[x]])
            bst[x] = bst[fa[x]];
        return fa[x] = cr;
    }
    void dfs(int u) {
        id[dfn[u] = ++dfc] = u;
        for (int v : G[u])
            if (!dfn[v])
                dfs(v), fa[dfn[v]] = dfn[u];
    }
    void tar(vector<int> *eg, int rt) {
        dfc = 0, dfs(rt);
        for (int i = 1; i <= dfc; i++)
            sdm[i] = bst[i] = i;
        for (int i = dfc; i > 1; i--) {
            int u = id[i];
            for (int v : rG[u])
                if ((v = dfn[v]))
                    f(v, i), tmin(sdm[i], sdm[bst[v]]);
            eg[sdm[i]].pb(i), u = fa[i];
            for (int v : eg[u])
                f(v, u), idm[v]
                     = (sdm[bst[v]] == u ? u : bst[v]);
            eg[u].clear();
        }
        for (int i = 2; i <= dfc; i++) {
            if (sdm[i] != idm[i])
                idm[i] = idm[idm[i]];
            eg[id[idm[i]]].pb(id[i]);
        }
    }
};
```

3.4 Matching [becd87]

```
struct Matching {
    static const int maxn
         = 505, p = (int)1e9 + 7;// change this, 1-base
    int sizen = 0;
    int sub_n=0;
    int id[maxn], vertices[maxn], matches[maxn];
    bool row_marked
        [maxn] = {false}, col_marked[maxn] = {false};
    int A[maxn][maxn], B[maxn][maxn], t[maxn][maxn];
    vector<pair<int,int> > sidearr;
    void init(int _n) {
        sizen = _n;
        sub_n = 0;
        fill(id,id+_n+1,0);
        fill(vertices,vertices+_n+1,0);
        fill(matches,matches+_n+1,0);
        fill(row_marked,row_marked+_n+1,0);
        fill(col_marked,col_marked+_n+1,0);
        for(int i=0; i<=_n; i++) {
            fill(A[i],A[i]+_n+1,0);
            fill(B[i],B[i]+_n+1,0);
            fill(t[i],t[i]+_n+1,0);
```

```
        }
        sidearr.clear();
    }
    Matching(int _n) {
        init(_n);
    }
    int qpow(int a, int b) {
        int ans = 1;
        while (b) {
            if (b & 1) ans = (long long)ans * a % p;
            a = (long long)a * a % p;
            b >>= 1;
        }
        return ans;
    }
    void Gauss(int A[][maxn], int B[][maxn], int n) {
        if (B) {
            memset(B, 0, sizeof(t));
            for (int i = 1; i <= n; i++) B[i][i] = 1;
        }
        for (int i = 1; i <= n; i++) {
            if (!A[i][i]) {
                for (int j = i + 1; j <= n; j++) {
                    if (A[j][i]) {
                        swap(id[i], id[j]);
                        for (int k = i; k <= n; k
                            ++) swap(A[i][k], A[j][k]);
                        if (B
                            ) for (int k = 1; k <= n; k
                            ++) swap(B[i][k], B[j][k]);
                        break;
                    }
                }
                if (!A[i][i]) continue;
            }
            int inv = qpow(A[i][i], p - 2);
            for (int j = 1; j <= n; j++) {
                if (i != j && A[j][i]) {
                    int t
                        = (long long)A[j][i] * inv % p;
                    for (int k = i; k <= n
                        ; k++) if (A[i][k]) A[j][k] = (
                        A[j][k] - (ll)t * A[i][k]) % p;
                    if (B) {
                        for (int k = 1; k <= n; k++) if
                             (B[i][k]) B[j][k] = (B[j][
                            k] - (ll)t * B[i][k]) % p;
                    }
                }
            }
        }
        if (B) {
            for (int i = 1; i <= n; i++) {
                int inv = qpow(A[i][i], p - 2);
                for (int j = 1; j <= n; j++) {
                    if (B[i][j]) B[i][j]
                        = (long long)B[i][j] * inv % p;
                }
            }
        }
    }
    void eliminate(int r, int c) {
        row_marked[r] = col_marked[c] = true;
        int inv = qpow(B[r][c], p - 2);
        for (int i = 1; i <= sub_n; i++) {
            if (!row_marked[i] && B[i][c]) {
                int t = (long long)B[i][c] * inv % p;

                for (int j = 1; j <= sub_n; j++)
                    if (!col_marked[j] && B[r][j])
                        B[i][j] = (B[i][j] - (
                            long long)t * B[r][j]) % p;
            }
        }
    }
    void add_edge(int a,int b) {
        sidearr.pb(min(a,b),max(a,b));
    }
    void build_matching() {
        auto rng = mt19937(chrono::steady_clock
            ::now().time_since_epoch().count());
        for(auto e : sidearr) {
            int x = e.first;
            int y = e.second;
            A[x][y] = rng() % p;
            A[y][x] = -A[x][y];
        }
        for (int i = 1; i <= sizen; i++) id[i] = i;
        memcpy(t, A, sizeof(t));
        Gauss(A, nullptr, sizen);
        for (int i = 1; i <= sizen; i++) {
            if (A[id[i]][id[i]]) vertices[++sub_n] = i;
        }
        for (int i = 1; i <= sub_n; i++) {
            for (int j = 1; j <= sub_n; j++)
                 A[i][j] = t[vertices[i]][vertices[j]];
        }
        Gauss(A, B, sub_n);
        for (int i = 1; i <= sub_n; i++) {
            if (!matches[vertices[i]]) {
                for (int j = i + 1; j <= sub_n; j++) {
                    if (!matches
                        [vertices[j]] && t[vertices
                        [i]][vertices[j]] && B[j][i]) {
                        matches[
                            vertices[i]] = vertices[j];
                        matches[
                            vertices[j]] = vertices[i];
                        eliminate(i, j);
                        eliminate(j, i);
                        break;
                    }
                }
            }
        }
    }
    int matched(int x) {
        return matches[x];
    }
};
```

3.5 Max Clique [2b1496]

```
struct MaxClique { // fast when N <= 100
  static const int N = 105;
  bitset<N> G[N], cs[N];
  int ans, sol[N], q, cur[N], d[N], n;
  MaxClique(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) G[i].reset();
  }
  void add_edge(int u, int v) {
    G[u][v] = G[v][u] = 1;
  }
  void pre_dfs(vector<int> &r, int l, bitset<N> mask) {
    if (l < 4) {
      for (int i : r) d[i] = (G[i] & mask).count();
      sort(r.begin(),r.end()
          , [&](int x, int y) { return d[x] > d[y]; });
    }
    vector<int> c(size(r));
    int lft = max(ans - q + 1, 1), rgt = 1, tp = 0;
    cs[1].reset(), cs[2].reset();
    for (int p : r) {
      int k = 1;
      while ((cs[k] & G[p]).any()) ++k;
      if (k > rgt) cs[++rgt + 1].reset();
      cs[k][p] = 1;
      if (k < lft) r[tp++] = p;
    }
    for (int k = lft; k <= rgt; ++k)
```

```
      for (int p = cs[k]._Find_first
          (); p < N; p = cs[k]._Find_next(p))
        r[tp] = p, c[tp] = k, ++tp;
    dfs(r, c, l + 1, mask);
  }
  void dfs(vector<
      int> &r, vector<int> &c, int l, bitset<N> mask) {
    while (!r.empty()) {
      int p = r.back();
      r.pop_back(), mask[p] = 0;
      if (q + c.back() <= ans) return;
      cur[q++] = p;
      vector<int> nr;
      for (int i : r) if (G[p][i]) nr.pb(i);
      if (!nr.empty()) pre_dfs(nr, l, mask & G[p]);
      else if (q > ans) ans = q, copy_n(cur, q, sol);
      c.pop_back(), --q;
    }
  }
  int solve() {
    vector<int> r(n);
    ans = q = 0, iota(r.begin(),r.end(), 0);
    pre_dfs(r, 0, bitset<N>(string(n, '1')));
    return ans;
  }
};
```

3.6 Max Flow [14be51]

```
struct FLOW {
    static const int N
        = 1e3 + 5, M = N * 10, s = 0, t = 1; // change
    int dp[N], cr[N], hd[N], ct = 2;
    struct E {
        int to, cap, nxt;
    } eg[M];
    inline void adeg(int u, int v, int w) {
        eg[ct] = {v, w, hd[u]};
        hd[u] = ct++;
        eg[ct] = {u, 0, hd[v]};
        hd[v] = ct++;
    }
    bool bfs() {
        memset(dp + 1, INF, ct << 2);
        queue<int> q;
        q.push(s), dp[s] = 0;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = hd[u]; i; i = eg[i].nxt) {
                const int v = eg[i].to;
                if (!eg[i].cap || dp[u] + 1 >= dp[v])
                    continue;
                dp[v] = dp[u] + 1, q.push(v);
            }
        }
        return dp[t] != INF;
    }

    int dfs(int u, int fl) {
        if (u == t)
            return fl;
        int sm = 0;
        for (int &i = cr[u]; i; i = eg[i].nxt) {
            int v = eg[i].to, &w = eg[i].cap;
            if (!w || dp[u] + 1 != dp[v])
                continue;
            int tp = dfs(v, min(w, fl - sm));
            w -= tp, sm += tp, eg[i ^ 1].cap += tp;
            if (sm == fl)
                return fl;
        }
        return sm;
    }
    int getflow() {
        int ans = 0;
        while (bfs())
            memcpy(cr + 1,
                hd + 1, ct << 2), ans += dfs(s, INF);
        return ans;
    }
};
```

4 Math

4.1 Mod Int [030eea]

```
template<unsigned P>
struct mint {// P not prime break /=
    unsigned v;
    mint(ll v=0) : v(v % P) {}
    mint &operator+=(mint const&o) {
        v = (v += o.v) >= P ? v - P : v;
        return *this;
    }
    mint &operator-=(mint const&o) {
        v = (v < o.v) ? v + P -o.v : v-o.v;
        return *this;
    }
    mint &operator*=(mint const&o) {
        ll ret =
             ll(v)*ll(o.v) - P*ll(1.L/P*ll(v)*ll(o.v));
    v = ret + P * (ret < 0) - P * (ret >= (ll)P);
        return *this;
    }
    friend mint operator+(mint const&a, mint const&b) {
        return mint(a) += b;
    }
    friend mint operator-(mint const&a, mint const&b) {
        return mint(a) -= b;
    }
    friend mint operator*(mint const&a,mint const&b) {
        return mint(a) *= b;
    }
    mint pow(ll n) const {
        mint r(1);
        mint a = v;
        for(; n; a*=a,n>>=1) r*=(n&1)?(a):(mint(1));
        return r;
    }
    mint &operator/=(mint const&o) {
        *this*=o.pow(P-2);
        return *this;
    }
    friend mint operator/(mint const&a,mint const&b) {
        return mint(a)/=b;
    }
    friend auto operator
        <<(ostream& os, mint const&m) -> ostream& {
        return os << m.v;
    }
};
```

4.2 Fraction [c7b9b9]

```
struct frac {
    ll n, d;
    frac(const
         ll &_n = 0, const ll &_d = 1) : n(_n), d(_d) {
        ll t = __gcd(n, d);
        n /= t, d /= t;
        if (d < 0)
            n = -n, d = -d;
    }
    frac operator-() const {
        return frac(-n, d);
    }
    frac operator+(const frac &b) const {
        return frac(n * b.d + b.n * d, d * b.d);
    }
    void operator+=(const frac &b) {
        *this = frac(n * b.d + b.n * d, d * b.d);
    }
    frac operator-(const frac &b) const {
        return frac(n * b.d - b.n * d, d * b.d);
```

```
    }
    void operator-=(const frac &b) {
        *this = frac(n * b.d - b.n * d, d * b.d);
    }
    frac operator*(const frac &b) const {
        return frac(n * b.n, d * b.d);
    }
    void operator*=(const frac &b) {
        *this = frac(n * b.n, d * b.d);
    }
    frac operator/(const frac &b) const {
        return frac(n * b.d, d * b.n);
    }
    void operator/=(const frac &b) {
        *this = frac(n * b.d, d * b.n);
    }
    friend ostream
     &operator<<(ostream &os, frac const &f) {
        if (f.d == 1)
            return os << f.n;
        return os << f.n << '/' << f.d;
    }
};
```

4.3 FFT [6e0d63]

```
typedef complex<double> cp;
void fft(vector<cp> &f,int rev){
  const double PI = 3.14159265358979363;
  int n = size(f);
  if(n==1) return ;
  vector<cp> o(n/2),e(n/2);
  for(int i=0;i<n;i++){
    if(i&1) o[i/2] = (f[i]);
    else e[i/2] = (f[i]);
  }
  fft(o,rev);fft(e,rev);
  cp cur(1,0);cp step(cos(2*PI/n),sin(rev*2*PI/n));
  for(int i=0;i<n/2;i++){
    f[i] = e[i]+cur*o[i];
    f[i+n/2] = e[i]-cur*o[i];
    if(rev<0){
      f[i]/=2;
      f[i+n/2]/=2;
    }
    cur*=step;
  }
}
```

4.4 Miller Rabin [67a711]

```
bool isPrime(const uint64_t n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  uint64_t A[] = {2,
       325, 9375, 28178, 450775, 9780504, 1795265022},
  s = __builtin_ctzll(n-1), D = n >> s;
  for (auto a : A) {
    uint64_t p =1,g=a%n,i=s,d=D;
    for(;d;g=__int128
        (g)*g%n,d/=2) if(d&1) p = __int128(p)*g%n;
    while (p != 1 && p
         != n - 1 && a % n && i--) p = __int128(p)*p%n;
    if (p != n-1 && i != s) return 0;
  }
  return 1;
}
```

4.5 Pollard's Rho [8252e7]

```
ll PollardRho(
    ll x) {// get a factor of x(not prime) in O(x^0.25)
    ll s = 0, t = 0;
    ll c = (ll)rand() % (x - 1) + 1;
    int step = 0, g = 1;
    ll val = 1;
    for (g = 1;; g <<= 1, s = t, val = 1) {
        for (step = 1; step <= g; ++step) {
            t =  (__int128(t)*t+c)%x;
            val = __int128(val) * abs(t - s) % x;
            if ((step % 127) == 0) {
                ll d = __gcd(val, x);
                if (d > 1) return d;
            }
        }
        ll d = __gcd(val, x);
        if (d > 1) return d;
    }
}
```

4.6 FWT [b14f6a]

```
struct FWT { // Modint needed
    string op; // and, or, xor
    void fwt(vector<mint> &v, bool ifwt) {
        int n = __lg(size(v));
        mint iv2 = mint(1) / 2;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < 1 << n; ++j)
                if (op == "and" && (~j >> i &
                    1) || op == "or" && (j >> i & 1)) {
                    if (!ifwt)
                        v[j] += v[j ^ (1 << i)];
                    else
                        v[j] -= v[j ^ (1 << i)];
                } else
                     if (op == "xor" && (j >> i & 1)) {
                    mint x = v[j ^ (1 << i)], y = v[j];
                    if (!ifwt)
                        v[j ^ (1 <<
                             i)] = x + y, v[j] = x - y;
                    else
                        v[j ^ (1 << i)] = (x + y) *
                             iv2, v[j] = (x - y) * iv2;
                }
    }
    vector<mint> v1, v2; // size(v1) = size(v2) = 2^k
    Fast_Walsh_Transform(const vector
        <mint> &_v1, const vector<mint> &_v2, const
         string &_op) : v1(_v1), v2(_v2), op(_op) {}
    vector<mint> solve
        () { // ans_k = \sum_{i op j = k} a_i * b_j
        fwt(v1, 0), fwt(v2, 0);
        for (int i = 0; i < size(v1); ++i)
            v1[i] *= v2[i];
        fwt(v1, 1);
        return v1;
    }
};
```

5 String

5.1 AC auto [6e5a60]

```
struct ACauto {
    const static int N = 2e5 + 5; // change
    int tr
        [26][N], fail[N], ctn = 0, cnt[N], endat[N], n;
    vector<int> top; // fail tree topological order
    inline void clr(int p) {
        fail[p] = cnt[p] = 0;
        for (int i = 0; i < 26; i++)
            tr[i][p] = 0;
    }
    inline int add(const string &s) {
        int cr = 1;
        string tmp;
        for (int c : s) {
            tmp += c;
            c -= 'a';
            if (!tr[c][cr])
                clr(tr[c][cr] = ++ctn);
            cr = tr[c][cr];
        }
        return cr;
    }
    void blt(const vector<string> &ar) {
```

```
        for (int i = 0; i < 26; i++)
            tr[i][0] = 1;
        clr(ctn = 1), n = size(ar);
        for (int i = 0; i < n; i++)
            endat[i] = add(ar[i]);
        queue<int> q;
        q.push(1);
        while (!q.empty()) {
            int pr = q.front();
            q.pop();
            top.pb(pr);
            for (int i = 0; i < 26; i++) {
                int &cr = tr[i][pr];
                if (cr)
                    fail[cr]
                        = tr[i][fail[pr]], q.push(cr);
                else
                    cr = tr[i][fail[pr]];
            }
        }
        reverse(iter(top));
    }
    void qry(const string &s) {
        int cr = 1;
        for (char c : s) // ways to walk
            cr = tr[c - 'a'][cr], cnt[cr]++;
        for (int i : top)
            cnt[fail[i]] += cnt[i];
    }
};
```

5.2 SA [58db4a]

```
struct SA {
    static const int N = 5e5 + 5; // change
    int sa[N], rk[N], cnt[N], lcp[N], tmp[N], n;
    void blt(const string &s) {
        n = s.length();
        int m = 128;
        memset(cnt + 1, 0, m << 2);
        for (int i = 0; i < n; i++)
            cnt[rk[i] = s[i]]++;
        for (int i = 1; i <= m; i++)
            cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--)
            sa[--cnt[rk[i]]] = i;
        for (int k = 1;; k <<= 1) {
            int ln = 0;
            for (int i = n - k; i < n; i++)
                tmp[ln++] = i;
            for (int i = 0; i < n; i++)
                if (sa[i] >= k)
                    tmp[ln++] = sa[i] - k;
            memset(cnt + 1, 0, m << 2);
            for (int i = 0; i < n; i++)
                cnt[rk[i]]++;
            for (int i = 1; i <= m; i++)
                cnt[i] += cnt[i - 1];
            for (int i = n - 1; i >= 0; i--)
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            memcpy(tmp, rk, n << 2), rk[sa[0]] = m = 1;
            for (int i = 1; i < n; i++) {
                if (tmp[sa[i]] != tmp[sa[i - 1]] ||
                    tmp[sa
                        [i] + k] != tmp[sa[i - 1] + k])
                    m++;
                rk[sa[i]] = m;
            }
            if (m == n)
                break;
        }
        for (int i = 0, k = 0; i < n; i++, k -= !!k) {
            if (rk[i] == n)
                continue;
            int j = sa[rk[i]];
            while (i + k <
                n && j + k < n && s[i + k] == s[j + k])
                k++;
            lcp[rk[i]] = k;
        } // lcp[1~n-1],sa[0~n-1]
    }
};
```

5.3 KMP [2b5dca]

```
vector<int> kmp(string &s) {
    int n = size(s);
    vector<int> pi(n, 0);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j])
            j = pi[j - 1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

5.4 Z [29dad4]

```
vector<int> z_algo(string &s) { // 0-base
    int n = size(s);
    vector<int> z(n, 0);
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r)
            z[i] = min(z[i - l], r - i + 1);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            z[i]++;
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    z[0] = n;
    return z;
}
```

5.5 Manacher [c08260]

```
vector<int> manacher(string &ss) {
    // biggest k such [i-k+1,i]=[i~i+k-1],padded with $
    string s;
    s.resize(size(ss) * 2 + 1, '$');
    for (int i = 0; i < size(ss); i++) {
        s[i * 2 + 1] = ss[i];
    }
    vector<int> p(size(s), 1);
    for (int i = 0, l = 0, r = 0; i < size(s); i++) {
        p[i] = max(min(p[l * 2 - i], r - i), 1);
        while (0 <= i - p[i] && i + p[i]
             < size(s) && s[i - p[i]] == s[i + p[i]]) {
            l = i, r = i + p[i], p[i]++;
        }
    }
    return p;
}
```

6 Data Sructure

6.1 Li-Chao [f2885c]

```
struct LiChaoMin {
    struct line {
        ll m, k, id;
        line(ll _m = 0, ll _k
            = 0, ll _id = 0) : m(_m), k(_k), id(_id) {}
        ll at(ll x) { return m * x + k; }
    };
    struct node {
        node *l, *r;
        line f;
        node(line v) : f(v), l(NULL), r(NULL) {}
    };
    node *root;
    int sz;
    void insert(node *&x, int l, int r, line &ln) {
        if (!x) {
            x = new node(ln);
```

```
            return;
        }
        ll trl = x->f.at(l), trr = x->f.at(r);
        ll vl = ln.at(l), vr = ln.at(r);
        if (trl <= vl && trr <= vr)
            return;
        if (trl > vl && trr > vr) {
            x->f = ln;
            return;
        }
        if (trl > vl)
            swap(x->f, ln);
        int mid = (l + r) >> 1;
        if (x->f.at(mid) < ln.at(mid))
            insert(x->r, mid + 1, r, ln);
        else
            swap(x->f, ln), insert(x->l, l, mid, ln);
    }
    ll query(node *&x, int l, int r, ll idx) {
        if (!x)
            return LONG_LONG_MAX;
        if (l == r)
            return x->f.at(idx);
        int mid = (l + r) >> 1;
        if (mid >= idx)
            return min(x
                ->f.at(idx), query(x->l, l, mid, idx));
        return min(x
            ->f.at(idx), query(x->r, mid + 1, r, idx));
    }
    LiChaoMin(int _sz) : sz(_sz + 1), root(NULL) {}
    void add_line(ll m, ll k, ll id = 0) {
        auto ln = line(m, k, id);
        insert(root, -sz, sz, ln);
    } // -sz <= query_x <= sz
    ll query
        (ll idx) { return query(root, -sz, sz, idx); }
};
```

6.2 Treap [5ab1a1]

```
struct node {
  int data, sz;
  node *l, *r;
  node(int k) : data(k), sz(1), l(0), r(0) {}
  void up() {
    sz = 1;
    if (l) sz += l->sz;
    if (r) sz += r->sz;
  }
  void down() {}
};
int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
  if (!a || !b) return a ? a : b;
  if (rand() % (sz(a) + sz(b)) < sz(a))
    return a->down(), a->r = merge(a->r, b), a->up(),
                        a;
  return b->down(), b->l = merge(a, b->l), b->up(), b;
}
void split(node *o, node *&a, node *&b, int k) {
  if (!o) return a = b = 0, void();
  o->down();
  if (o->data <= k)
    a = o, split(o->r, a->r, b, k), a->up();
  else b = o, split(o->l, a, b->l, k), b->up();
}
void split2(node *o, node *&a, node *&b, int k) {
  if (sz(o) <= k) return a = o, b = 0, void();
  o->down();
  if (sz(o->l) + 1 <= k)
    a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
  else b = o, split2(o->l, a, b->l, k);
  o->up();
}
node *kth(node *o, int k) {
  if (k <= sz(o->l)) return kth(o->l, k);
  if (k == sz(o->l) + 1) return o;
  return kth(o->r, k - sz(o->l) - 1);
}
int Rank(node *o, int key) {
  if (!o) return 0;
  if (o->data < key)
    return sz(o->l) + 1 + Rank(o->r, key);
  else return Rank(o->l, key);
}
bool erase(node *&o, int k) {
  if (!o) return 0;
  if (o->data == k) {
    node *t = o;
    o->down(), o = merge(o->l, o->r);
    delete t;
    return 1;
  }
  node *&t = k < o->data ? o->l : o->r;
  return erase(t, k) ? o->up(), 1 : 0;
}
void insert(node *&o, int k) {
  node *a, *b;
  split(o, a, b, k),
    o = merge(a, merge(new node(k), b));
}
void interval(node *&o, int l, int r) {
  node *a, *b, *c;
  split2(o, a, b, l - 1), split2(b, b, c, r);
  // operate
  o = merge(a, merge(b, c));
}
```

6.3 Ultimate Segment Tree [6e7e86]

```
struct SegBeat {
    int n,n0;
    vector<ll> max_v,smax_v
        ,max_c,min_v,smin_v,min_c,sum,len,ladd,lval;
    void update_node_max(int k, ll x) {
        sum[k] += (x - max_v[k]) * max_c[k];
        if(max_v
            [k] == min_v[k]) max_v[k] = min_v[k] = x;
        else if(max_v
            [k] == smin_v[k]) max_v[k] = smin_v[k] = x;
        else max_v[k] = x;
        if(lval[k] != 1e18 && x < lval[k]) lval[k] = x;
    }
    void update_node_min(int k, ll x) {
        sum[k] += (x - min_v[k]) * min_c[k];
        if(max_v
            [k] == min_v[k]) max_v[k] = min_v[k] = x;
        else if(smax_v
            [k] == min_v[k]) min_v[k] = smax_v[k] = x;
        else min_v[k] = x;
        if(lval[k] != 1e18 && lval[k] < x) lval[k] = x;
    }
    void push(int k) {
        if(n0-1 <= k) return;
        if(lval[k] != 1e18) {
            updateall(2*k+1, lval[k]);
            updateall(2*k+2, lval[k]);
            lval[k] = 1e18;
            return;
        }
        if(ladd[k] != 0) {
            addall(2*k+1, ladd[k]);
            addall(2*k+2, ladd[k]);
            ladd[k] = 0;
        }
        if(max_v[k] < max_v
            [2*k+1]) update_node_max(2*k+1, max_v[k]);
        if(min_v[2*k+1] <
            min_v[k]) update_node_min(2*k+1, min_v[k]);
        if(max_v[k] < max_v
            [2*k+2]) update_node_max(2*k+2, max_v[k]);
        if(min_v[2*k+2] <
            min_v[k]) update_node_min(2*k+2, min_v[k]);
```

```
    }
    void update(int k) {
        sum[k] = sum[2*k+1] + sum[2*k+2];
        if(max_v[2*k+1] < max_v[2*k+2]) {
            max_v[k] = max_v[2*k+2];
            max_c[k] = max_c[2*k+2];
            smax_v
                [k] = max(max_v[2*k+1], smax_v[2*k+2]);
        } else if(max_v[2*k+1] > max_v[2*k+2]) {
            max_v[k] = max_v[2*k+1];
            max_c[k] = max_c[2*k+1];
            smax_v
                [k] = max(smax_v[2*k+1], max_v[2*k+2]);
        } else {
            max_v[k] = max_v[2*k+1];
            max_c[k] = max_c[2*k+1] + max_c[2*k+2];
            smax_v[
                k] = max(smax_v[2*k+1], smax_v[2*k+2]);
        }
        if(min_v[2*k+1] < min_v[2*k+2]) {
            min_v[k] = min_v[2*k+1];
            min_c[k] = min_c[2*k+1];
            smin_v
                [k] = min(smin_v[2*k+1], min_v[2*k+2]);
        } else if(min_v[2*k+1] > min_v[2*k+2]) {
            min_v[k] = min_v[2*k+2];
            min_c[k] = min_c[2*k+2];
            smin_v
                [k] = min(min_v[2*k+1], smin_v[2*k+2]);
        } else {
            min_v[k] = min_v[2*k+1];
            min_c[k] = min_c[2*k+1] + min_c[2*k+2];
            smin_v[
                k] = min(smin_v[2*k+1], smin_v[2*k+2]);
        }
    }
    void _chmin
        (ll x, int a, int b, int k, int l, int r) {
        if(b <= l || r <= a || max_v[k] <= x) return;
        if(a <= l && r <= b && smax_v[k] < x) {
            update_node_max(k, x);
            return;
        }
        push(k);
        _chmin(x, a, b, 2*k+1, l, (l+r)/2);
        _chmin(x, a, b, 2*k+2, (l+r)/2, r);
        update(k);
    }
    void _chmax
        (ll x, int a, int b, int k, int l, int r) {
        if(b <= l || r <= a || x <= min_v[k]) return;
        if(a <= l && r <= b && x < smin_v[k]) {
            update_node_min(k, x);
            return;
        }
        push(k);
        _chmax(x, a, b, 2*k+1, l, (l+r)/2);
        _chmax(x, a, b, 2*k+2, (l+r)/2, r);
        update(k);
    }
    void addall(int k, ll x) {
        max_v[k] += x;
        if(smax_v[k] != -1e18) smax_v[k] += x;
        min_v[k] += x;
        if(smin_v[k] != 1e18) smin_v[k] += x;
        sum[k] += len[k] * x;
        if(lval[k] != 1e18) lval[k] += x;
        else ladd[k] += x;
    }
    void updateall(int k, ll x) {
        max_v[k] = x;
        smax_v[k] = -1e18;
        min_v[k] = x;
        smin_v[k] = 1e18;
        max_c[k] = min_c[k] = len[k];
        sum[k] = x * len[k];
        lval[k] = x;
        ladd[k] = 0;
    }
    void _add_val
        (ll x, int a, int b, int k, int l, int r) {
        if(b <= l || r <= a) return;
        if(a <= l && r <= b) {
            addall(k, x);
            return;
        }
        push(k);
        _add_val(x, a, b, 2*k+1, l, (l+r)/2);
        _add_val(x, a, b, 2*k+2, (l+r)/2, r);
        update(k);
    }
    void _update_val
        (ll x, int a, int b, int k, int l, int r) {
        if(b <= l || r <= a) return;
        if(a <= l && r <= b) {
            updateall(k, x);
            return;
        }
        push(k);
        _update_val(x, a, b, 2*k+1, l, (l+r)/2);
        _update_val(x, a, b, 2*k+2, (l+r)/2, r);
        update(k);
    }
    ll _query_max(int a, int b, int k, int l, int r) {
        if(b <= l || r <= a) return -1e18;
        if(a <= l && r <= b) return max_v[k];
        push(k);
        ll lv = _query_max(a, b, 2*k+1, l, (l+r)/2);
        ll rv = _query_max(a, b, 2*k+2, (l+r)/2, r);
        return max(lv, rv);
    }
    ll _query_min(int a, int b, int k, int l, int r) {
        if(b <= l || r <= a) return 1e18;
        if(a <= l && r <= b) return min_v[k];
        push(k);
        ll lv = _query_min(a, b, 2*k+1, l, (l+r)/2);
        ll rv = _query_min(a, b, 2*k+2, (l+r)/2, r);
        return min(lv, rv);
    }
    ll _query_sum(int a, int b, int k, int l, int r) {
        if(b <= l || r <= a) return 0;
        if(a <= l && r <= b) return sum[k];
        push(k);
        ll lv = _query_sum(a, b, 2*k+1, l, (l+r)/2);
        ll rv = _query_sum(a, b, 2*k+2, (l+r)/2, r);
        return lv + rv;
    }
    SegBeat(int _n) : n(_n) {
        max_v.resize(4*_n+4,0);smax_v.resize
            (4*_n+4,-1e18);max_c.resize(4*_n+4,1);
        min_v.resize(4*_n+4,0);smin_v.
            resize(4*_n+4,1e18);min_c.resize(4*_n+4,1);
        sum.resize(4*_n+4,0);len.resize(4*_n+4,0);
        ladd.resize(4*_n+4,0);lval.resize(4*_n+4,1e18);
        n0 = 1;
        while(n0 < n) n0 <<= 1;
        len[0] = n0;
        for(int i=0; i<n0-1; ++
            i) len[2*i+1] = len[2*i+2] = (len[i] >> 1);
        for(int i=n; i<n0; ++i) {
            max_v[n0-1+i] = smax_v[n0-1+i] = -1e18;
            min_v[n0-1+i] = smin_v[n0-1+i] = 1e18;
            max_c[n0-1+i] = min_c[n0-1+i] = 0;
        }
        for(int i=n0-2; i>=0; i--) update(i);
    }
    void chmin(int
         a, int b, ll x) {_chmin(x, a-1, b, 0, 0, n0);}
    void chmax(int
         a, int b, ll x) {_chmax(x, a-1, b, 0, 0, n0);}
```

```
    void add_val(int a
        , int b, ll x) {_add_val(x, a-1, b, 0, 0, n0);}
    void update_val(int a, int
         b, ll x) {_update_val(x, a-1, b, 0, 0, n0);}
    ll query_max(int a
        , int b) {return _query_max(a-1, b, 0, 0, n0);}
    ll query_min(int a
        , int b) {return _query_min(a-1, b, 0, 0, n0);}
    ll query_sum(int a
        , int b) {return _query_sum(a-1, b, 0, 0, n0);}
};
```