National Taiwan University

Ashes

Chieh-Tu Yu, Tung-Chun Lin, Chan-Yu Yeh

Contents

1 Basic **1**
1.1 Fast Integer Input . . . 1
1.2 Increase stack size . . . 1
1.3 Default Code . . . 1
1.4 Beat Match . . . 2
1.5 Mo's Algorithm . . . 2
1.6 Big Integer . . . 2
1.7 CDQ . . . 3

2 Flows, Matching **4**
2.1 Dinic's Algorithm . . . 4
2.2 Minimum Cost Maximum Flow . . . 4
2.3 Khun's Algorithm . . . 5
2.4 Bipartite Matching . . . 5
2.5 Weighted Bipartite Match . . . 5
2.6 Maximum Matching on General Graph . . . 5
2.7 Flow Models . . . 6

3 Data Structure **6**
3.1 Disjoint Set Union . . . 6
3.2 Segment Tree . . . 7
3.3 Lazy Segment Tree . . . 7
3.4 Treap . . . 8
3.5 Binary Search on BIT . . . 8
3.6 LiChaoTree . . . 8
3.7 HJT Tree . . . 8
3.8 HJT Tree on Tree . . . 9
3.9 Dynamic Open Point Segment Tree . . . 9
3.10 Linear Basis . . . 9
3.11 Fenwick Tree Sparse2D Offline . . . 10

4 Graph **10**
4.1 Domination . . . 10
4.2 Edge Bi-Connected Component . . . 10
4.3 Vertex Bi-Connected Component . . . 10
4.4 Strongly Connected Component . . . 11
4.5 2 SAT . . . 11
4.6 Round Square Tree . . . 11
4.7 Heavy Light Decomposition . . . 11
4.8 Centroid Decomposition . . . 12
4.9 Directed MST . . . 12
4.10 Dominator Tree . . . 13
4.11 Stoer Wagner Minimum Cut . . . 13
4.12 Maximum Clique . . . 13
4.13 Edge Coloring . . . 13
4.14 Mo's on Tree . . . 14

5 String **14**
5.1 Knuth Morris Pratt Algorithm . . . 14
5.2 Z Algorithm . . . 14
5.3 AC Automaton . . . 14
5.4 Suffix Automaton . . . 15
5.5 Suffix Array . . . 15
5.6 Least Rotation . . . 15
5.7 Manacher . . . 16

6 Math **16**
6.1 Sieve of Eratosthenes . . . 16
6.2 Euler's Totient Function . . . 16
6.3 Extended GCD . . . 16
6.4 Floor Sum . . . 16
6.5 Primitive Root . . . 16
6.6 Discrete Logarithm . . . 16
6.7 Discrete Root . . . 17
6.8 Linear Function Mod Min . . . 17
6.9 Fast Inverse . . . 17
6.10 Factorial Mod . . . 17
6.11 Miller Rabin . . . 18
6.12 Pollard's Rho . . . 18
6.13 Linear Equation . . . 18
6.14 Simplex Construction . . . 18
6.15 Simplex . . . 18
6.16 NTT . . . 19
6.17 NTT Prime . . . 19
6.18 FFT . . . 19
6.19 XOR Convolution . . . 20
6.20 SG . . . 20
6.21 Theorem . . . 20
6.22 Mobius Inversion . . . 20
6.23 Chinese Remainder Theorem . . . 21

7 Geometry **21**
7.1 Basic . . . 21
7.2 Half Plane Intersection . . . 21
7.3 Slope & Fraction . . . 21
7.4 Convex order . . . 21
7.5 Convex Hull . . . 22
7.6 Area . . . 22
7.7 Circle . . . 22
7.8 Area of Union of Circles . . . 22
7.9 Minimun Distance of 2 Polygons . . . 23
7.10 Closest Pair . . . 23
7.11 Minkowski Sum . . . 23
7.12 Point in Polygon . . . 23

8 Dynamic Programming **23**
8.1 DP-Optimization . . . 23
8.2 DP $1D/1D$ . . . 23
8.3 Conditon . . . 24
8.3.1 Totally Monotone (Concave/Convex) . . . 24
8.3.2 Monge Condition (Concave/Convex) . . . 24
8.3.3 Optimal Split Point . . . 24
8.4 Dynamic Convex Hull . . . 24
8.5 $1D/1D$ Convex Optimization . . . 24
8.6 Convex Monotone . . . 24
8.7 Convex NonMonotone . . . 24
8.8 Sum over Subsets Optimization . . . 25

1 Basic

1.1 Fast Integer Input

```
inline int gtx() {
  const int N = 4096;
  static char buffer[N];
  static char *p = buffer, *end = buffer;
  if (p == end) {
    if ((end = buffer + fread(buffer, 1, N, stdin)) == buffer)
        return EOF;
    p = buffer;
  }
  return *p++;
}
template <typename T>
inline bool rit(T& x) {
  char c = 0; bool flag = false;
  while (c = getchar(), (c < '0' && c != '-') || c > '9') if (c
      == -1) return false;
  c == '-' ? (flag = true, x = 0) : (x = c - '0');
  while (c = getchar(), c >= '0' && c <= '9') x = x * 10 + c -
      '0';
  if (flag) x = -x;
  return true;
}
```

1.2 Increase stack size

```
const int size = 256 << 20;
register long rsp asm("rsp");
char *p = (char*)malloc(size) + size, *bak = (char*)rsp;
__asm__("movq %0, %%rsp\n"::"r"(p));
// main
__asm__("movq %0, %%rsp\n"::"r"(bak));
```

1.3 Default Code

```
#include <bits/stdc++.h>
#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")
using namespace std;
#define DBG(x)   cout << (#x " = ") << x << endl;
#define ALL(x) x.begin(), x.end()
#define push_back emplace_back
#define spIO ios::sync_with_stdio(false);cin.tie(0)
#define mem(arr, value) memset(arr, value, sizeof(arr))
#define for0(i, n) for(int i = 0; i < n; i++)
#define for1(i, n) for(int i = 1; i <= n; i++)
typedef long long ll;
typedef long double ld;
const ll mod = 1e9 + 7;
inline ll pow2(ll target, ll p, ll MOD = mod)
{
    ll ret = 1;
    while (p)
    {
        if (p & 1)
            (ret *= target) %= MOD;
        p >>= 1;
        (target *= target) %= MOD;
    }
    return ret;
}
inline ll inv(ll x, ll MOD = mod)
{
    return pow2(x, MOD - 2, MOD);
}
inline ll gcd(ll x, ll y)
{
    if (!y)
        return x;
    if ((x & 1) && (y & 1))
```

```
    {
        if (x < y)
            swap(x, y);
        return gcd(y, x - y);
    }
    if (x & 1)
        return gcd(x, y >> 1);
    if (y & 1)
        return gcd(y, x >> 1);
    return 2 * gcd(x >> 1, y >> 1);
}
inline ll sub(ll x, ll y, ll MOD = mod)
{
    return (x - y < 0 ? x - y + MOD : x - y);
}
inline ll add(ll x, ll y, ll MOD = mod)
{
    return (x + y >= MOD ? x + y - MOD : x + y);
}
vector<pair<int, int>> directions = {
     {1,0},{1,1},{0,1},{-1,1},{-1,0},{-1,-1},{0,-1},{1,-1} };
```

1.4 Beat Match

```
#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")
#include <bits/stdc++.h>
using namespace std;

int main(){
  int t = 0;
  while(true){
    cout << "Test Case: " << ++t << '\n';
    system("gen.exe>data.in");
    system("Sol.exe<data.in>Sol.out");
    system("Bf.exe<data.in>Bf.out");
    if (system("fc Sol.out Bf.out")){
      break;
    }
  }
  return 0;
}
```

1.5 Mo's Algorithm

```

const int N = 2e5+5e4;
const int S = 1e6+5;
int k;

struct query{
  int l, r, t, id;
  bool operator < (query b){
    return (l/k==b.l/k  ? (r/k == b.r/k ? t < b.t : r/k < b.r/k
         ) : l/k < b.l/k);
  }
} Q[N];


struct upd{
  int pos,x;
} M[N];

int arr[N], cnt[S], ans[N], tmp;

void add(int val){
  if(!cnt[val]) tmp++;
  cnt[val]++;
}

void sub(int val){
  cnt[val]--;
  if(!cnt[val]) tmp--;
}

void modify(query x, upd &y){
  if(x.l <= y.pos && y.pos <= x.r){
    sub(arr[y.pos]);
    add(y.x);
  }
  swap(arr[y.pos],y.x);
}

signed main(){
  fastio
  int n, m;
  cin >> n >> m;
  k = pow(n,(double)2/(double)3);
  for(int i = 1;i <= n;i++) cin >> arr[i];
  int tid = 0, qid = 0;
  for(int i = 0;i < m;i++){
    char q;
    cin >> q;
    if(q=='Q'){
      int l, r;
      cin >> l >> r;
      Q[qid] = {l,r,tid,qid};
      qid++;
    }else{
      int x, val;
      cin >> x >> val;
      ++tid;
      M[tid] = {x,val};
    }
  }
  sort(Q,Q+qid);
  int l = 1, r = 0, t = 0;
  for(int i = 0;i < qid;i++){
    auto q = Q[i];
    while(l < q.l) sub(arr[l++]);
    while(l > q.l) add(arr[--l]);
    while(r < q.r) add(arr[++r]);
    while(r > q.r) sub(arr[r--]);
    while(t < q.t) modify(Q[i],M[++t]);
    while(t > q.t) modify(Q[i],M[t--]);
    ans[q.id] = tmp;
  }
  for(int i = 0;i < qid;i++) cout << ans[i] << "\n";
}
```

1.6 Big Integer

```
#include <bits/stdc++.h>

struct Int {
  static const int inf = 1e9;
  std::vector<int> dig;
  bool sgn;
  Int() {
    dig.push_back(0);
    sgn = true;
  }
  Int(int n) {
    sgn = n >= 0;
    while (n) {
      dig.push_back(n % 10);
      n /= 10;
    }
    if (dig.size() == 0) dig.push_back(0);
  }
  Int(std::string s) {
    int i = 0; sgn = true;
    if (s[i] == '-') sgn = false, ++i;
    for (;i < s.length(); ++i) dig.push_back(s[i] - '0');
    reverse(dig.begin(), dig.end());
    if (dig.size() == 1 && dig[0] == '0') sgn = true;
  }
  Int(const std::vector<int>& d, const bool& s = true) {
    dig = std::vector<int>(d.begin(), d.end());
    sgn = s;
  }
  Int(const Int& n) {
    sgn = n.sgn;
    dig = n.dig;
  }
  bool operator<(const Int& rhs) const {
    if (sgn && !rhs.sgn) return true;
    if (!sgn && rhs.sgn) return false;
    if (!sgn && !rhs.sgn) return Int(dig) > Int(rhs.dig);
    if (dig.size() < rhs.dig.size()) return true;
    if (dig.size() > rhs.dig.size()) return false;
    for (int i = dig.size() - 1; i >= 0; --i) {
      if (dig[i] != rhs.dig[i]) return dig[i] < rhs.dig[i];
    }
    return false;
  }
  bool operator==(const Int& rhs) const {
    if (sgn != rhs.sgn) return false;
    return dig == rhs.dig;
  }
  bool operator>(const Int& rhs) const {
    return !(*this < rhs) && !(*this == rhs);
  }
  bool operator<(const int& n) const {
```

```
    return *this < Int(n);
  }
  bool operator>(const int& n) const {
    return *this > Int(n);
  }
  bool operator==(const int& n) const {
    return *this == Int(n);
  }
  Int operator-() const {
    return Int(dig, !sgn);
  }
  Int operator+(const Int& rhs) const {
    bool res = true;
    if (!sgn && !rhs.sgn) res = false;
    else if (!sgn && rhs.sgn) return rhs - (-*this);
    else if (sgn && !rhs.sgn) return *this - -rhs;
    std::vector<int> v1 = dig, v2 = rhs.dig;
    if (v2.size() > v1.size()) swap(v1, v2);
    int car = 0;
    std::vector<int> nvec;
    for (int i = 0; i < v2.size(); ++i) {
      int k = v1[i] + v2[i] + car;
      nvec.push_back(k % 10);
      car = k / 10;
    }
    for (int i = v2.size(); i < v1.size(); ++i) {
      int k = v1[i] + car;
      nvec.push_back(k % 10);
      car = k / 10;
    }
    return Int(nvec, res);
  }
  Int operator-(const Int& rhs) const {
    if (*this < rhs) {
      std::vector<int> nvec = (rhs - *this).dig;
      return Int(nvec, false);
    }
    if (*this == rhs) return Int(0);
    std::vector<int> v1 = dig, v2 = rhs.dig;
    std::vector<int> nvec;
    for (int i = 0; i < v2.size(); ++i) {
      int k = v1[i] - v2[i];
      if (k < 0) {
        for (int j = i + 1; j < v1.size(); ++j) if (v1[j] > 0)
             {
          --v1[j]; k += 10;
          break;
        }
      }
      nvec.push_back(k);
    }
    int rind = v1.size() - 1;
    while (rind >= v2.size() && v1[rind] == 0) --rind;
    for (int i = v2.size(); i <= rind; ++i) {
      nvec.push_back(v1[i]);
    }
    return Int(nvec);
  }
  Int operator*(const Int& rhs) const {
    if (sgn && !rhs.sgn || !sgn && rhs.sgn) return -(Int(dig,
         true) * Int(rhs.dig, true));
    if (*this == 0) return Int();
    if (rhs == 0) return Int();
    std::vector<int> v1 = dig, v2 = rhs.dig;
    if (v1.size() < v2.size()) swap(v1, v2);
    std::vector<int> res(v1.size() * v2.size(), 0);
    for (int i = 0; i < v2.size(); ++i) {
      int car = 0;
      for (int j = 0; j < v1.size(); ++j) {
        int k = car + v1[j] * v2[i];
        res[j + i] += k % 10;
        car = k / 10;
      }
    }
    int car = 0;
    for (int i = 0; i < res.size(); ++i) {
      int k = car + res[i];
      res[i] = k % 10;
      car = k / 10;
    }
    while (car) {
      res.push_back(car % 10);
      car /= 10;
    }
    int ind = res.size() - 1;
    while (ind >= 0 && res[ind] == 0) --ind;
    std::vector<int> nvec;
    for (int i = 0; i <= ind; ++i) nvec.push_back(res[i]);
    return Int(nvec);
  }
  Int operator+(const int& n) const {
    return *this + Int(n);
  }
  Int operator-(const int& n) const {
    return *this - Int(n);
  }
  Int& operator+=(const Int& n) {
    *this = (*this + n);
    return *this;
  }
  Int& operator-=(const Int& n) {
    *this = (*this - n);
    return *this;
  }
  Int& operator+=(const int& n) {
    *this += Int(n);
    return *this;
  }
  Int& operator-=(const int& n) {
    *this -= Int(n);
    return *this;
  }
  Int& operator*=(const Int& n) {
    *this = *this * n;
    return *this;
  }
  Int& operator*=(const int& n) {
    *this *= Int(n);
    return *this;
  }
  Int& operator++(int) {
    *this += 1;
    return *this;
  }
  Int& operator--(int) {
    *this -= 1;
    return *this;
  }
  friend std::istream& operator>>(std::istream& in, Int& n) {
    std::string s; in >> s;
    n = Int(s);
    return in;
  }
  friend std::ostream& operator<<(std::ostream& out, const Int&
       n) {
    if (!n.sgn) out << "-";
    for (int i = n.dig.size() - 1; i >= 0; --i) out << n.dig[i
         ];
    return out;
  }
};
```

1.7 CDQ

```
struct node{
    int a, b, c;
    int id;
};

bool cmpA(node x, node y) {
  if (x.a != y.a) return x.a < y.a;
  if (x.b != y.b) return x.b < y.b;
  return x.c < y.c;
}

bool cmpB(node x, node y) {
  if (x.b != y.b) return x.b < y.b;
  return x.c < y.c;
}

// res[i] = #(j, a[j] < a[i], b[j] < b[i], c[j] < c[i])
// bit.query(i) = pref(0~i-1)
void solve(){
    vector<node> A(n+1);
    for (int i = 0; i <= n; i++){
        A[i].a = a[i];
        A[i].b = b[i];
        A[i].c = c[i];
        A[i].id = i;
    }
    sort(ALL(A), cmpA);

    fenwick bit(n+1);
    vector<int> res(n+1);
```

```
    function<void(int, int)> CDQ = [&](int l, int r){
        if (l == r || A[l].a == A[r].a) return;

        int mid = (l + r) >> 1;
        int target = 0;
        for (int i = l; i < r; i++){
            if (A[i].a != A[i+1].a){
                if (abs(i - mid) < abs(target - mid)){
                    target = i;
                }
            }
        }

        mid = target;
        CDQ(l, mid); CDQ(mid+1, r);

        sort(A.begin()+l, A.begin()+mid+1, cmpB);
        sort(A.begin()+mid+1, A.begin()+r+1, cmpB);

        for (int i = l, j = mid+1; j <= r; j++){
            while (i <= mid && A[i].b < A[j].b){
                bit.add(A[i].c, 1);
                i++;
            }
            res[A[j].id] = max(res[A[j].id], bit.sum(A[j].c));
        }

        for (int i = l; i <= mid; i++){
            bit.add(A[i].c, -1);
        }

    };
}
```

2 Flows, Matching

2.1 Dinic's Algorithm

```
(a) Bounded Maxflow Construction:
1. add two node ss, tt
2. add_edge(ss, tt, INF)
3. for each edge u -> v with capacity [l, r]:
        add_edge(u, tt, l)
        add_edge(ss, v, l)
        add_edge(u, v, r-l)
4. see (b), check if it is possible.
5. answer is maxflow(ss, tt) + maxflow(s, t)
-------------------------------------------------------
(b) Bounded Possible Flow:
1. same construction method as (a)
2. run maxflow(ss, tt)
3. for every edge connected with ss or tt:
        rule: check if their rest flow is exactly 0
4. answer is possible if every edge do satisfy the rule;
5. otherwise, it is NOT possible.
-------------------------------------------------------
(c) Bounded Minimum Flow:
1. same construction method as (a)
2. answer is maxflow(ss, tt)
-------------------------------------------------------
(d) Bounded Minimum Cost Flow:
* the concept is somewhat like bounded possible flow.
1. same construction method as (a)
2. answer is maxflow(ss, tt) + (∑ l * cost for every edge)
-------------------------------------------------------
(e) Minimum Cut:
1. run maxflow(s, t)
2. run cut(s)
3. ss[i] = 1: node i is at the same side with s.
-------------------------------------------------------
using T = long long;
struct Edge {
  int to, rev;
  T cap;
  Edge(int a, T b, int c) : to(a), cap(b), rev(c) {}
};
T Flow(vector<vector<Edge>> &g, int s, int t){
  int n = g.size();
  T res = 0;
  vector<int> lev(n, -1), iter(n);
  while(true){
    vector<int> que(1, s);
    fill(lev.begin(), lev.end(), -1);
    fill(iter.begin(), iter.end(), 0);
    lev[s] = 0;
    for (int i = 0; i < (int)que.size(); i++){
      int x = que[i];
      for (Edge &e : g[x]){
        if (e.cap > 0 && lev[e.to] == -1){
          lev[e.to] = lev[x] + 1;
          que.push_back(e.to);
        }
      }
    }
    if (lev[t] == -1) break;
    auto Dfs = [&](auto dfs, int x, T f = 2e18){
      if (x == t) return f;
      T res = 0;
      for (int &i = iter[x]; i < (int)g[x].size(); i++){
        Edge &e = g[x][i];
        if (e.cap > 0 && lev[e.to] == lev[x] + 1){
          T p = dfs(dfs, e.to, min(f - res, e.cap));
          res += p;
          e.cap -= p;
          g[e.to][e.rev].cap += p;
        }
      }
      if (res == 0) lev[x] = -1;
      return res;
    };
    res += Dfs(Dfs, s);
  }
  return res;
}
auto Add = [&](int a, int b, ll c){
  g[a].emplace_back(b, c, (int)g[b].size());
  g[b].emplace_back(a, 0, (int)g[a].size()-1);
};
```

2.2 Minimum Cost Maximum Flow

```
struct Edge {
  int to, cap, rev, w;
  Edge(int t, int c, int r, int w) : to(t), cap(c), rev(r), w(w
      ) {}
};
pair<int, int> Flow(vector<vector<Edge>> g, int s, int t) {
  int N = g.size();
  vector<int> dist(N), ed(N), pv(N);
  vector<bool> inque(N);
  int flow = 0, cost = 0;
  while (true) {
    dist.assign(N, kInf);
    inque.assign(N, false);
    pv.assign(N, -1);
    dist[s] = 0;
    queue<int> que;
    que.push(s);
    while (!que.empty()) {
      int x = que.front(); que.pop();
      inque[x] = false;
      for (int i = 0; i < g[x].size(); ++i) {
        Edge &e = g[x][i];
        if (e.cap > 0 && dist[e.to] > dist[x] + e.w) {
          dist[e.to] = dist[x] + e.w;
          pv[e.to] = x;
          ed[e.to] = i;
          if (!inque[e.to]) {
            inque[e.to] = true;
            que.push(e.to);
          }
        }
      }
    }
    if (dist[t] == kInf) break;
    int f = kInf;
    for (int x = t; x != s; x = pv[x]) f = min(f, g[pv[x]][ed[x
        ]].cap);
    for (int x = t; x != s; x = pv[x]) {
      Edge &e = g[pv[x]][ed[x]];
      e.cap -= f;
      g[e.to][e.rev].cap += f;
    }
    flow += f;
    cost += f * dist[t];
  }
  return make_pair(flow, cost);
}
auto AddEdge = [&](int from, int to, int cap, int weight) {
  g[from].emplace_back(to, cap, g[to].size(), weight);
  g[to].emplace_back(from, 0, g[from].size() - 1, -weight);
```

```
};
```

2.3 Khun's Algorithm

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

int main() {
    //... reading the graph ...

    mt.assign(k, -1);
    for (int v = 0; v < n; ++v) {
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

2.4 Bipartite Matching

```
const int INF = 1e9;
struct Match {  // O( ( V + E ) * sqrt( V ) )
  vector<vector<int>> G;
  vector<int> match, dist;
  int n, m;
  // n: number of nodes on left side, nodes are numbered 1 to n
  // m: number of nodes on right side, nodes are numbered n to
       n+m
  // G = NIL[0]   G1[G[1---n]]   G2[G[n+1---n+m]]
  Match (int _n, int _m) : n(_n), m(_m), G(_n+_m+1), match(_n+
       _m+1), dist(_n+_m+1) {}
  bool BFS() {
    queue< int > Q;
    for ( int i = 1; i <= n; i++ ) {
      if ( match[ i ] == 0 ) {
        dist[ i ] = 0;
        Q.push( i );
      }
      else
        dist[ i ] = INF;
    }
    dist[ 0 ] = INF;
    while ( !Q.empty() ) {
      int u = Q.front();
      Q.pop();
      if ( dist[ u ] < dist[ 0 ] )
        for ( int v : G[ u ] )
          if ( dist[ match[ v ] ] == INF ) {
            dist[ match[ v ] ] = dist[ u ] + 1;
            Q.push( match[ v ] );
          }
    }
    return ( dist[ 0 ] != INF );
  }
  bool DFS( int u ) {
    if ( u != 0 ) {
      for ( int v : G[ u ] )
        if ( dist[ match[ v ] ] == dist[ u ] + 1 && DFS( match[
             v ] ) ) {
          match[ v ] = u;
          match[ u ] = v;
          return true;
        }
      dist[ u ] = INF;
      return false;
    }
    return true;
  }
  int Max_Match() {
    int matching = 0;
    fill_n( match.begin(), n + m + 1, 0 );
    while ( BFS() )
      for ( int i = 1; i <= n; i++ )
        if ( match[ i ] == 0 && DFS( i ) ) matching++;
    return matching;
  }
  void AddEdge( int u, int v ) { G[ u ].push_back( n + v ); }
  void DFS2( int u ) {
  if (dist[u]) return;
    dist[ u ] = 1;
    for ( int v : G[ u ] )
      if ( v != match[ u ] ) {
        dist[ v ] = 1;
        if ( match[ v ] != 0 ) DFS2( match[ v ] );
      }
  }
  vector<int> Min_Vertex_Cover( ) {
    // after calling Max_Match
    fill_n( dist.begin() + 1, n + m + 1, 0 );
    for ( int i = 1; i <= n; i++ )
      if ( match[ i ] == 0 ) DFS2( i );
    vector<int> res(n);
    for ( int i = 1; i <= n; i++ )
      if ( dist[ i ] == 0 ) res[i-1] = true;
    for ( int i = n + 1; i <= n + m; i++ )
      if ( dist[ i ] == 1 ) res[i-n-1] = true;
    return res;
  }
};
```

2.5 Weighted Bipartite Match

```
// this is one-indexed
// jobs X workers cost matrix
// cost[i][j] is cost of job i done by worker j
// #jobs must be <= #workers
// Default finds min cost; to find max cost set all costs[i][j]
      to -costs[i][j]
// O(n^2m)
const ll inf = 2e18;
pair<ll, vector<int>> HungarianMatch(const vector<vector<ll>>&
     a) {
  int n = a.size()-1;
  int m = a[0].size()-1;
  vector<ll> u(n+1), v(m+1), p(m+1), way(m+1);
  for(int i = 1; i <= n; ++i) {
    p[0] = i;
    int j0 = 0;
    vector<ll> minv(m+1, inf);
    vector<bool> used(m+1, false);
    do {
      used[j0] = true;
      int i0 = p[j0], j1 = -1;
      ll delta = inf;
      for(int j = 1; j <= m; ++j)
        if(!used[j]) {
          ll cur = a[i0][j] - u[i0] - v[j];
          if(cur < minv[j])
            minv[j] = cur, way[j] = j0;
          if(minv[j] < delta)
            delta = minv[j], j1 = j;
        }
      for(int j = 0; j <= m; ++j)
        if(used[j])
          u[p[j]] += delta, v[j] -= delta;
        else
          minv[j] -= delta;
        j0 = j1;
    } while(p[j0] != 0);
    do {
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    } while(j0);
  }

  // For each N, it contains the M it selected
  vector<int> ans(n+1);
  for(int j = 1; j <= m ; ++j)
    ans[p[j]] = j;

  return {-v[0], ans};
}
```

2.6 Maximum Matching on General Graph

```
const int kN = 500;
namespace matching {
  int fa[kN], pre[kN], match[kN], s[kN], v[kN];
  vector<int> g[kN];
  queue<int> q;
  void Init(int n) {
    for (int i = 0; i <= n; ++i) match[i] = pre[i] = n;
    for (int i = 0; i < n; ++i) g[i].clear();
  }
  void AddEdge(int u, int v) {
    g[u].push_back(v);
    g[v].push_back(u);
  }
  int Find(int u) {
    return u == fa[u] ? u : fa[u] = Find(fa[u]);
  }
  int LCA(int x, int y, int n) {
    static int tk = 0;
    tk++;
    x = Find(x), y = Find(y);
    for (; ; swap(x, y)) {
      if (x != n) {
        if (v[x] == tk) return x;
        v[x] = tk;
        x = Find(pre[match[x]]);
      }
    }
  }
  void Blossom(int x, int y, int l) {
    while (Find(x) != l) {
      pre[x] = y, y = match[x];
      if (s[y] == 1) q.push(y), s[y] = 0;
      if (fa[x] == x) fa[x] = l;
      if (fa[y] == y) fa[y] = l;
      x = pre[y];
    }
  }
  bool Bfs(int r, int n) {
    for (int i = 0; i <= n; ++i) fa[i] = i, s[i] = -1;
    while (!q.empty()) q.pop();
    q.push(r);
    s[r] = 0;
    while (!q.empty()) {
      int x = q.front(); q.pop();
      for (int u : g[x]) {
        if (s[u] == -1) {
          pre[u] = x, s[u] = 1;
          if (match[u] == n) {
            for (int a = u, b = x, last; b != n; a = last, b =
                 pre[a])
              last = match[b], match[b] = a, match[a] = b;
            return true;
          }
          q.push(match[u]);
          s[match[u]] = 0;
        } else if (!s[u] && Find(u) != Find(x)) {
          int l = LCA(u, x, n);
          Blossom(x, u, l);
          Blossom(u, x, l);
        }
      }
    }
    return false;
  }
  int Solve(int n) {
    int res = 0;
    for (int x = 0; x < n; ++x) {
      if (match[x] == n) res += Bfs(x, n);
    }
    return res;
  }
}
```

2.7 Flow Models

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x, y, l, u), connect $x \to y$ with capacity $u - l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     - To maximize, connect $t \to s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     - To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f + f' \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
  1. Redirect every edge: $y \to x$ if $(x, y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u, v, w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  6. T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
  1. For each $v \in V$ create a copy v', and connect $u' \to v'$ with weight $w(u, v)$.
  2. Connect $v \to v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  3. Find the minimum weight perfect matching on G'.
- Project selection problem
  1. If $p_v > 0$, create edge (s, v) with capacity p_v; otherwise, create edge (v, t) with capacity $-p_v$.
  2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v.
  3. The mincut is equivalent to the maximum profit of a subset of projects.
- 0/1 quadratic programming

  $$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x\bar{y} + \sum_{xyx'y'} c_{xyx'y'}(x\bar{y} + x'\bar{y'})$$

  can be minimized by the mincut of the following graph:
  1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y.
  2. Create edge (x, y) with capacity c_{xy}.
  3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3 Data Structure

3.1 Disjoint Set Union

```
struct DisjointSet {
    int p[maxn], sz[maxn], n, cc;
    vector<pair<int*, int>> his;
    vector<int> sh;
    void init(int _n) {
        n = _n; cc = n;
        for (int i = 0; i < n; ++i) sz[i] = 1, p[i] = i;
        sh.clear(); his.clear();
    }
    void assign(int *k, int v) {
        his.emplace_back(k, *k);
        *k = v;
    }
    void save() {
        sh.push_back((int)his.size());
    }
    void undo() {
        int last = sh.back(); sh.pop_back();
        while (his.size() != last) {
            int *k, v;
            tie(k, v) = his.back(); his.pop_back();
            *k = v;
        }
    }
    int find(int x) {
        if (x == p[x]) return x;
        return find(p[x]);
    }
    void merge(int x, int y) {
        x = find(x); y = find(y);
        if (x == y) return;
        if (sz[x] > sz[y]) swap(x, y);
        assign(&sz[y], sz[x] + sz[y]);
        assign(&p[x], y);
        assign(&cc, cc - 1);
    }
} dsu;
```

3.2 Segment Tree

```
template<typename T>
struct segtree {
  int n;
  vector<T> seg;
    const T ID;

    T comb(T a, T b) {
        assert(a.size() == b.size());
        T c;
        for (int i = 0; i < a.size(); i++){
            c.push_back(b[a[i]]);
        }
        return c;
    }
  // 0-based
  segtree(vector<T> leaves): n(leaves.size()), seg(2*leaves.
      size()), ID(leaves[0]) {
    copy(begin(leaves), end(leaves), begin(seg)+n);
    for (int i = n-1; i >= 1; i--) {
      seg[i] = comb(seg[2*i], seg[2*i+1]);
    }
  }

  void modify(int i, T v) {
    i += n;
    seg[i] = v;
    for (i /= 2; i; i /= 2) {
      seg[i] = comb(seg[2*i], seg[2*i+1]);
    }
  }
  // [l, r)
  T query(int i, int j) {
    i += n, j += n;
    T resl = ID, resr = ID;
    for (; i < j; i /= 2, j /= 2) {
      if (i & 1) resl = comb(resl, seg[i++]);
      if (j & 1) resr = comb(seg[--j], resr);
    }
    return comb(resl, resr);
  }
};


struct node {
    node *l, *r;
    data d;
    node() : l(0), r(0), d() {}
} *root;


#define mid ((l + r) >> 1)
void build(node *now, int l, int r) {
    if (l == r) return;
    build(now->l = new node(), l, mid);
    build(now->r = new node(), mid + 1, r);
}

void modify(node *now, int l, int r, int x, int v) {
    if (l == r) {
        now->d.modify(v);
        return;
    }
    if (x <= mid) modify(now->l, l, mid, x, v);
    else modify(now->r, mid + 1, r, x, v);
    now->d = now->l->d + now->r->d;
}

data query(node *now, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) return now->d;
    if (qr <= mid) return query(now->l, l, mid, ql, qr);
    if (mid < ql) return query(now->r, mid + 1, r, ql, qr);
    return query(now->l, l, mid, ql, qr) + query(now->r, mid +
         1, r, ql, qr);
}
```

3.3 Lazy Segment Tree

```
struct node{
  ll sum1;
    ll sum2;
    bool lz;
  ll lzAdd;
  ll lzSet;
  node(){};
}seg[maxN*4+5];

#define lc v<<1
#define rc (v<<1)+1

inline void pull(int v){
  seg[v].sum1 = seg[lc].sum1 + seg[rc].sum1;
    seg[v].sum2 = seg[lc].sum2 + seg[rc].sum2;
}

inline ll sum1(int l, int r){
    return r-l+1;
}

inline ll sum2(int l, int r){
    return 1LL * (r-l+1) * (l+r) / 2;
}

void st(int v, int l, int r, ll val){
    seg[v].lz = true;
    seg[v].lzSet = val;
    seg[v].lzAdd = 0;
    seg[v].sum1 = val * sum1(l, r);
    seg[v].sum2 = val * sum2(l, r);
}

void add(int v, int l, int r, ll val){
    seg[v].lzAdd += val;
    seg[v].sum1 += val * sum1(l, r);
    seg[v].sum2 += val * sum2(l, r);
}

void pushdown(int v, int l, int r){
    int m = (l + r) >> 1;
  //lazy: range set
  if(seg[v].lz){
        st(lc, l, m, seg[v].lzSet);
        st(rc, m+1, r, seg[v].lzSet);
    seg[v].lzSet = 0;
        seg[v].lz = false;
  }
    //lazy: range add
    if(seg[v].lzAdd != 0){
        add(lc, l, m, seg[v].lzAdd);
        add(rc, m+1, r, seg[v].lzAdd);
        seg[v].lzAdd = 0;
  }
}

void build(int v, int l, int r){
  seg[v].lzAdd = seg[v].lzSet = 0;
    seg[v].lz = false;
  if(l == r){
    seg[v].sum1 = a[l];
        seg[v].sum2 = a[l] * l;
    return;
  }
  int m = (l+r)>>1;
  build(lc, l, m);
  build(rc, m+1, r);
  pull(v);
}

void add(int v, int l, int r, int a, int b, ll val){
  if(a > r || b < l) return;
  if(a <= l && r <= b){
        add(v, l, r, val);
    return;
  }
    pushdown(v, l, r);
  int m = (l+r)>>1;
  add(lc, l, m,a ,b , val);
  add(rc, m+1, r, a, b, val);
  pull(v);
  return;
}

void st(int v, int l, int r, int a, int b, ll val){
  if(a > r || b < l)return;
  if(a <= l && r <= b){
    st(v, l, r, val);
    return;
  }
    pushdown(v, l, r);
  int m = (l+r)>>1;
    st(lc, l, m, a, b, val);
    st(rc, m+1, r, a, b, val);
    pull(v);
  return;
}
```

```
ll query1(int v, int l, int r, int a, int b){
  if(a > r || b < l)return 0;
  if(a <= l && r <= b) return seg[v].sum1;
    pushdown(v, l, r);
    int m = (l+r) >> 1;
    ll lres = query1(lc, l, m, a, b);
    ll rres = query1(rc, m+1, r, a, b);
    // pull(v);
    return lres + rres;
}
```

3.4 Treap

```
typedef struct item * pitem;
struct item {
  int prior, value, cnt;
  bool rev;
  pitem l, r;
};

int cnt (pitem it) {
  return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
  if (it)
    it->cnt = cnt(it->l) + cnt(it->r) + 1;
}

void push (pitem it) {
  if (it && it->rev) {
    it->rev = false;
    swap (it->l, it->r);
    if (it->l)  it->l->rev ^= true;
    if (it->r)  it->r->rev ^= true;
  }
}

void merge (pitem & t, pitem l, pitem r) {
  push (l);
  push (r);
  if (!l || !r)
    t = l ? l : r;
  else if (l->prior > r->prior)
    merge (l->r, l->r, r),  t = l;
  else
    merge (r->l, l, r->l),  t = r;
  upd_cnt (t);
}

void split (pitem t, pitem & l, pitem & r, int key, int add =
    0) {
  if (!t)
    return void( l = r = 0 );
  push (t);
  int cur_key = add + cnt(t->l);
  if (key <= cur_key)
    split (t->l, l, t->l, key, add),  r = t;
  else
    split (t->r, t->r, r, key, add + 1 + cnt(t->l)),  l = t;
  upd_cnt (t);
}

void reverse (pitem t, int l, int r) {
  pitem t1, t2, t3;
  split (t, t1, t2, l);
  split (t2, t2, t3, r-l+1);
  t2->rev ^= true;
  merge (t, t1, t2);
  merge (t, t, t3);
}

void output (pitem t) {
  if (!t)  return;
  push (t);
  output (t->l);
  printf ("%d ", t->value);
  output (t->r);
}
```

3.5 Binary Search on BIT

```
int kth(int k){
    int pos = 0;
    int n = maxn-1;
    for (int i = 1 << __lg(n); i; i>>=1){
        if (pos + i <= n && bit[pos + i] < k){
            pos += i;
            k -= bit[pos];
        }
    }
    if (pos == n) return -1;
    return pos+1;
}
```

3.6 LiChaoTree

```
namespace lichao {
    struct line {
        long long a = 0, b = inf;
        line() {}
        line(long long a, long long b) : a(a), b(b) {}
        long long operator()(ll x) const { return a * x + b; }
    };
    vector<line> st;
    vl pts;
    void init() { st.clear(); pts.clear(); }
    void addpt(ll x) { pts.push_back(x); }
    void orderpts() {
        sort(ALL(pts));
        pts.erase(unique(ALL(pts)), pts.end());
        st.resize(4LL * (pts.size() + 5));
    }
    int lc(int x) { return 2 * x + 1; }
    int rc(int x) { return 2 * x + 2; }
    void update(int o, int l, int r, line tl, ll a, ll b) {
        if (l > r || pts[r]<a || pts[l]>b) return;
        int mid = (l + r) / 2;
        if (a <= pts[l] && pts[r] <= b) {
            if (st[o].a < tl.a) swap(st[o], tl);
            if (st[o](pts[mid]) < tl(pts[mid])) {
                if (l != r) update(rc(o), mid + 1, r, tl, a, b)
                    ;
            }
            else {
                swap(st[o], tl);
                if (l != r) update(lc(o), l, mid, tl, a, b);
            }
            return;
        }
        update(lc(o), l, mid, tl, a, b);
        update(rc(o), mid + 1, r, tl, a, b);
    }
    ll query(int o, int l, int r, ll x) {
        if (l > r) return inf;
        if (l == r)return st[o](x);
        int mid = (l + r) / 2;
        if (x <= pts[mid]) return min(st[o](x), query(lc(o), l,
            mid, x));
        else return min(st[o](x), query(rc(o), mid + 1, r, x));
    }
}
```

3.7 HJT Tree

```
const int maxn = 1e5;
int tot, n, m;
int sum[(maxn << 5) + 10], rt[maxn + 10], ls[(maxn << 5) + 10],
    rs[(maxn << 5) + 10];
int a[maxn + 10], ind[maxn + 10], len;
inline int getid(const int &val) {
  return lower_bound(ind + 1, ind + len + 1, val) - ind;
}
int build(int l, int r) {
  int root = ++tot;
  if (l == r) return root;
  int mid = l + r >> 1;
  ls[root] = build(l, mid);
  rs[root] = build(mid + 1, r);
  return root;
}
int update(int k, int l, int r, int root) {
  int dir = ++tot;
  ls[dir] = ls[root], rs[dir] = rs[root], sum[dir] = sum[root]
      + 1;
  if (l == r) return dir;
  int mid = l + r >> 1;
  if (k <= mid)
    ls[dir] = update(k, l, mid, ls[dir]);
  else
    rs[dir] = update(k, mid + 1, r, rs[dir]);
  return dir;
}
int query(int u, int v, int l, int r, int k) {
  int mid = l + r >> 1,
```

```
    x = sum[ls[v]] - sum[ls[u]];
  if (l == r) return l;
  if (k <= x)
    return query(ls[u], ls[v], l, mid, k);
  else
    return query(rs[u], rs[v], mid + 1, r, k - x);
}
inline void init() {
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= n; ++i) scanf("%d", a + i);
  memcpy(ind, a, sizeof ind);
  sort(ind + 1, ind + n + 1);
  len = unique(ind + 1, ind + n + 1) - ind - 1;
  rt[0] = build(1, len);
  for (int i = 1; i <= n; ++i) rt[i] = update(getid(a[i]), 1,
      len, rt[i - 1]);
}
int l, r, k;
inline void work() {
  while (m--) {
    scanf("%d%d%d", &l, &r, &k);
    printf("%d\n", ind[query(rt[l - 1], rt[r], 1, len, k)]);
  }
}
int main() {
  init();
  work();
  return 0;
}
```

3.8 HJT Tree on Tree

```
int ls[mxN * LOG], rs[mxN * LOG], val[mxN * LOG];
int rt[mxN];
vector<pi> G[mxN];
int tot;
int insert(int x, int a, int b, int w) {
  int res = ++tot;
  ls[res] = ls[x], rs[res] = rs[x], val[res] = val[x] + 1;
  if (a == b) {
    return res;
  }
  int mid = (a + b) / 2;
  if (w <= mid) {
    ls[res] = insert(ls[res], a, mid, w);
  }
  else {
    rs[res] = insert(rs[res], mid + 1, b, w);
  }
  return res;
}
void dfs(int u, int p) {

  for (pi pp : G[u]) {
    int v = pp.first, w = pp.second;
    if (v == p) {
      continue;
    }
    rt[v] = insert(rt[u], 1, 100000, w);
    dfs(v, u);
  }
}
int query(int x, int y, int z, int a, int b, int k) {

  if (a == b) {
    return a;
  }
  int mid = (a + b) / 2;
  int num = val[ls[x]] + val[ls[y]] - val[ls[z]] * 2;
  if (k <= num) {
    return query(ls[x], ls[y], ls[z], a, mid, k);
  }
  else {
    return query(rs[x], rs[y], rs[z], mid + 1, b, k - num);
  }
}
int solve(){
  while (q--) {
    int a, b; cin >> a >> b;
    a--; b--;
    int p = lca(a, b);
    int num = dep[a] + dep[b] - dep[p] * 2;
    if (num % 2) {
      int res = query(rt[a], rt[b], rt[p], 1, 100000, num / 2 +
          1);
      cout << res << ".0" << "\n";
    }
    else {
      int res1 = query(rt[a], rt[b], rt[p], 1, 100000, num / 2)
          ;
      int res2 = query(rt[a], rt[b], rt[p], 1, 100000, num / 2
          + 1);
      int res = res1 + res2;
      if (res % 2) cout << res / 2 << ".5" << "\n";
      else cout << res / 2 << ".0" << "\n";
    }
  }
}
```

3.9 Dynamic Open Point Segment Tree

```
const int maxn = 2e5 + 5;//query
const int maxsz = 45;//log2(1e9)*const
const int lb = 1,ub = 1000000000;//1e9
typedef struct node {
  ll sum;
  ll lazy;
  int l;
  int r;
}node;
node sgt[maxn * maxsz];
int tot = 1;
void initnode(int x) {
  sgt[x].sum = 0;
  sgt[x].lazy = 0;
  sgt[x].l = sgt[x].r = 0;
}
inline int ls(int x) {return sgt[x].l;}
inline int rs(int x) {return sgt[x].r;}
inline void pull(int x, int l, int r) {
  if (r - l + 1 <= 1) return;
  sgt[x].sum = sgt[ls(x)].sum + sgt[rs(x)].sum;
}

inline void push(int x, int l, int r) {
  if (l > r)  return;
  ll lazy = sgt[x].lazy;
  sgt[x].lazy = 0;
  if (l == r) return;
  if (ls(x) == 0) sgt[x].l = ++tot;
  if (rs(x) == 0) sgt[x].r = ++tot;
  if (lazy == 0) return;
  int mid = (l + r - 1) / 2;
  sgt[ls(x)].lazy += lazy;
  sgt[ls(x)].sum += lazy * (mid - l + 1);
  sgt[rs(x)].lazy += lazy;
  sgt[rs(x)].sum += lazy * (r - mid);

}
inline ll query(int x, int l, int r, int a, int b) {
  if (l > r || r<a || l>b) return 0;
  if (a <= l && r <= b) return sgt[x].sum;
  push(x, l, r);
  int mid = (l + r - 1) / 2;
  ll lres = 0, rres = 0;
  if (a <= mid) lres = query(ls(x), l, mid, a, b);
  if (mid < b)  rres = query(rs(x), mid + 1, r, a, b);
  pull(x, l, r);
  return lres + rres;
}
inline void update(int x, int l, int r, int a, int b, ll val) {
  if (l > r || r<a || l>b) return;
  if (a <= l && r <= b) {
    sgt[x].sum += val * (r - l + 1);
    sgt[x].lazy += val;
    return;
  }
  push(x, l, r);
  int mid = (l + r - 1) / 2;
  if (a <= mid) update(ls(x), l, mid, a, b, val);
  if (mid < b) update(rs(x), mid + 1, r, a, b, val);
  pull(x, l, r);
  return;
}
```

3.10 Linear Basis

```
vector<ll> p(61);
auto ins = [&](ll x){
    for (int i = 60; i >= 0; i--){
        if (x >> i & 1){
            if (!p[i]){
                p[i] = x;
                break;
```

```
            }else{
                x ^= p[i];
            }
        }
    }
};
```

3.11 Fenwick Tree Sparse2D Offline

```
template <class T, class IndexType1, class IndexType2>
struct FenwickTreeSparse2DOffline {
    int U; vector<int> st, cnt; vector<IndexType1> inds1;
    vector<IndexType2> inds2; vector<T> BIT;
    int getInd1(IndexType1 i) {
        return upper_bound(inds1.begin(), inds1.end(), i) -
             inds1.begin();
    }
    int getInd2(int i, IndexType2 j) {
        return upper_bound(inds2.begin() + st[i],
            inds2.begin() + st[i] + cnt[i], j)
            - inds2.begin() - st[i];
    }
    FenwickTreeSparse2DOffline(vector<pair<IndexType1,
         IndexType2>> updateInds)
        : inds1(updateInds.size()) {
        sort(updateInds.begin(), updateInds.end(),
            [&](const pair<IndexType1, IndexType2>& a,
                const pair<IndexType1, IndexType2>& b) {
                    return a.second < b.second;
            });
        for (int i = 0; i < int(updateInds.size()); i++)
            inds1[i] = updateInds[i].first;
        sort(inds1.begin(), inds1.end());
        inds1.erase(unique(inds1.begin(), inds1.end()), inds1.
             end());
        U = int(inds1.size()); st.assign(U + 1, 0); cnt.assign(
             U + 1, 0);
        vector<IndexType2> last(U + 1, T()); for (auto&& u :
             updateInds)
            for (int i = getInd1(u.first); i <= U; i += i & -i)
                if (cnt[i] == 0 || u.second != last[i]) {
                    cnt[i]++; last[i] = u.second;
                }
        for (int i = 1; i <= U; i++) st[i] = st[i - 1] + cnt[i
             - 1];
        inds2.resize(st[U] + cnt[U]); BIT.resize(st[U] + cnt[U
             ]);
        fill(cnt.begin(), cnt.end(), 0); for (auto&& u :
             updateInds)
            for (int i = getInd1(u.first); i <= U; i += i & -i)
                if (cnt[i] == 0 || u.second != inds2[st[i] +
                     cnt[i] - 1])
                    inds2[st[i] + cnt[i]++] = u.second;
    }
    void update(IndexType1 i, IndexType2 j, T v) {
        for (int x = getInd1(i); x <= U; x += x & -x)
            for (int s = st[x], c = cnt[x], y = getInd2(x, j);
                 y <= c; y += y & -y)
                BIT[s + y - 1] += v;
    }
    T rsq(IndexType1 d, IndexType2 r) {
        T ret = T(); for (int x = getInd1(d); x > 0; x -= x & -
             x)
            for (int s = st[x], y = getInd2(x, r); y > 0; y -=
                 y & -y)
                ret += BIT[s + y - 1];
        return ret;
    }
    T rsq(IndexType1 d, IndexType2 l, IndexType2 r) {
        return rsq(d, r) - rsq(d, l - 1);
    }
    T rsq(IndexType1 u, IndexType1 d, IndexType2 l, IndexType2
         r) {
        return rsq(d, l, r) - rsq(u - 1, l, r);
    }
};
//Initialize: FenwickTreeSparse2DOffline<ll,int,int> bit(
     updateInds);
//updateInds: points that will be updated
//Usage: bit.update(x,y,w) or bit.rsq(x_lo, x_hi, y_lo, y_hi);
```

4 Graph

4.1 Domination

```
Maximum Independent Set
General: [NPC] maximum clique of complement of G
Tree: [P] Greedy
Bipartite Graph: [P] Maximum Cardinality Bipartite Matching
-----------------------------------------------------
Minimum Dominating Set
General: [NPC]
Tree: [P] DP
Bipartite Graph: [NPC]
-----------------------------------------------------
Minimum Vertex Cover
General: [NPC] (?)maximum clique of complement of G
Tree: [P] Greedy, from leaf to root
Bipartite Graph: [P] Maximum Cardinality Bipartite Matching
-----------------------------------------------------
Minimum Edge Cover
General: [P] V - Maximum Matching
Bipartite Graph: [P] Greedy, strategy: cover small degree node
     first.
(Min/Max)Weighted: [P]: Minimum/Minimum Weight Matching
```

4.2 Edge Bi-Connected Component

```
vector<int> dfn(n), low(n);
int timer = 0, bcc = 0;
vector<int> id(n);
stack<int> stk;
function<void(int, int)> tarjan = [&](int u, int fa){
    dfn[u] = low[u] = ++timer;
    stk.push(u);
    for (auto e:adj[u]){
        int v = e.first;
        int w = e.second;
        if (w == fa) continue;
        if (!dfn[v]){
            tarjan(v, w);
            low[u] = min(low[u], low[v]);
        }else{
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (low[u] == dfn[u]){
        while (true) {
            int v = stk.top();
      stk.pop();
            id[v] = bcc;
            if (v == u) break;
        }
        bcc++;
    }
};
```

4.3 Vertex Bi-Connected Component

```
vector<int> dfn(n), low(n);
int timer = 0, bcc = 0;
vector<int> id(n, -1);
stack<int> stk;
vector<vector<int>> comp(n);
vector<int> res;
function<void(int, int)> tarjan = [&](int u, int fa){
    dfn[u] = low[u] = ++timer;
    stk.push(u);
    for (int e:adj[u]){
        int v = u ^ a[e] ^ b[e];
        if (v == fa) continue;
        if (!dfn[v]){
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u]){
                int z;
                do {
                    z = stk.top();
                    stk.pop();
                    comp[bcc].push_back(z);
                    id[z] = bcc;
                } while (z != v);
                comp[bcc].push_back(u);
                id[u] = bcc++; // id[u] can have multiple
                     values
            }
        }else{
            low[u] = min(low[u], dfn[v]);
        }
    }
};
```

4.4 Strongly Connected Component

```
   u        if and only if      1. or 2.
1. u            u
2. u                    (u,v)          (              u    v
                     )       DFN(u) <= Low(v)

-------------------------------------------------------

           (u,v)      if and only if (u,v)                  DFN
     (u) < Low(v)

vector<int> dfn(n), low(n), ins(n);
int timer = 0, scc = 0;
vector<int> id(n);
stack<int> stk;
void tarjan(int u){
  low[u] = dfn[u] = ++timer;
  ins[u] = 1;
  stk.push(u);
  for (int v:adj[u]){
    if (!dfn[v]){
      tarjan(v);
      low[u] = min(low[u], low[v]);
    }else if (ins[v]){
      low[u] = min(low[u], dfn[v]);
    }
  }
  if (low[u] == dfn[u]){
    int v;
    do {
      v = stk.top(); stk.pop();
      id[v] = scc;
      ins[v] = 0;
    } while(v != u);
    scc++;
  }
}
```

4.5 2 SAT

```
//G[i] := x_i is true; G[i+n] := x_i is false
//key : ( a or b ) is equal to ( !a => b and !b => a)
for (int i = 0; i < n; i++) {
  for (int j = i + 1; j < n; j++) {
    if (!(x_i && x_j)) {
      // ! (x_i && x_j) => !x_i or !x_j
      addedge(i, j + n);
      addedge(j, i + n);
    }
    if (! (x_i && !x_j)) {
      // ! (x_i && !x_j) => !x_i or x_j
      addedge(i, j);
      addedge(j + n, i + n);
    }
    if (! (!x_i && x_j)) {
      // ! (!x_i && x_j) -> x_i or !x_j
      addedge(i + n, j + n);
      addedge(j, i);
    }
    if (!(!x_i && !x_j)) {
      // !(!x_i && !x_j) => x_i or x_j
      addedge(i + n, j);
      addedge(j + n, i);
    }
  }
}
int num = scc(v);
for (int i = 0; i < n; i++) {
  if (id[i] == id[i + n]) cout << "NO\n",return;
}
cout << "YES\n";
for (int i = 0; i < n; i++) x[i] = (id[i + n] < id[i]);
//toposort !x before x <=> x is true. If tarjan, then (id[i + n
    ] > id[i])
```

4.6 Round Square Tree

```
const int maxn = 2e5 + 5;

int n, m, q;
int od[maxn], cir[maxn], sz;
namespace RST{
  const int log = 20;
  int dis[maxn], dep[maxn], f[maxn][log];
  vector<pi> adj[maxn];
  inline void add(int a, int b, int c){
    adj[a].emplace_back(b, c);
  }
  void dfs(int u, int fa){
    f[u][0] = fa;
    dep[u] = dep[fa] + 1;
    for (int i = 1; i < log; i++){
      f[u][i] = f[f[u][i-1]][i-1];
    }
    for (auto [v, w] :adj[u]){
      dis[v] = dis[u] + w;
      dfs(v, u);
    }
  }
  int query(int x, int y){
    if (dep[x] < dep[y]) swap(x, y);
    int u = x, v = y;
    int d = dep[u] - dep[v];
    for (int i = 0; i < log; i++){
      if (d >> i & 1){
        u = f[u][i];
      }
    }
    if (u == v) return dis[x] - dis[y];
    for (int i = log-1; i >= 0; i--){
      if (f[u][i] != f[v][i]){
        u = f[u][i];
        v = f[v][i];
      }
    }
    int lca = f[u][0];
    if (lca <= n) return dis[x] + dis[y] - 2 * dis[lca];
    d = abs(od[u] - od[v]);
    return dis[x] - dis[u] + dis[y] - dis[v] + min(d, cir[lca]
        - d);
  }
}

vector<pi> adj[maxn];
int dfn[maxn], low[maxn], timer;
int be[maxn], stk[maxn], top;
void tarjan(int u, int fa){
  dfn[u] = low[u] = ++timer;
  stk[++top] = u;
  for (auto[v, w]:adj[u]){
    if (v == fa) continue;
    if (!dfn[v]){
      od[v] = od[u] + w;
      tarjan(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] == dfn[u]){
        cir[++sz] = be[u];
        RST::add(u, sz, 0);
        for (int t = -1, j = top; t != v; ){
          t = stk[j--];
          cir[sz] += od[t] - od[stk[j]];
        }
        while(true){
          int d = od[stk[top]] - od[u];
          RST::add(sz, stk[top], min(d, cir[sz] - d));
          if (stk[top--] == v) break;
        }
      }else if (low[v] > dfn[u]){
        RST::add(u, v, w);
        top--;
      }
    }else{
      be[v] = w;
      low[u] = min(low[u], dfn[v]);
    }
  }
}
```

4.7 Heavy Light Decomposition

```
const int maxn = 1e5 + 5;
int u[maxn], v[maxn], w[maxn];
int p[maxn], dep[maxn], sz[maxn], to[maxn];
int tp[maxn], dfn[maxn], st[maxn * 4], tk;
vector<int> g[maxn];

void dfs(int u, int fa) {
    dep[u] = ~fa ? dep[fa] + 1 : 0;
    fa[u] = fa;
    sz[u] = 1;
    to[u] = -1;
```

```
    for (int i = 0; i < (int)g[u].size(); ++i) {
        int v = g[u][i];
        if (v == fa) continue;
        dfs(v, u);
        sz[u] += sz[v];
        if (to[u] == -1 || sz[v] > sz[to[u]]) swap(to[u], v);
    }
}

void hld(int u, int f) {
    tp[u] = f; dfn[u] = tk++;
    if (~to[u]) hld(to[u], f);

    for (int i = 0; i < (int)g[u].size(); ++i) {
        int v = g[u][i];
        if (v == fa[u] || v == to[u]) continue;
        hld(v, v);
    }
}

void modify(int l, int r, int ql, int qr, int v, int o = 0) {
    if (l >= qr || ql >= r) return;
    if (l >= ql && r <= qr) return st[o] = min(st[o], v), void
        ();
    modify(l, (l + r) >> 1, ql, qr, v, o * 2 + 1);
    modify((l + r) >> 1, r, ql, qr, v, o * 2 + 2);
}

int query(int l, int r, int p, int o = 0) {
    if (r - l == 1) return st[o];
    if (p < (l + r) >> 1) return min(st[o], query(l, (l + r) >>
         1, p, o * 2 + 1));
    else return min(st[o], query((l + r) >> 1, r, p, o * 2 + 2)
        );
}

vector<pair<int, int>> get(int x, int y) {
  int fx = tp[x], fy = tp[y];
  vector<pair<int, int>> res;
  while (fx != fy) {
    if (dep[fx] < dep[fy]) {
      swap(fx, fy);
      swap(x, y);
    }
    res.emplace_back(dfn[fx], dfn[x] + 1);
    x = fa[fx];
    fx = tp[x];
  }
  res.emplace_back(min(dfn[x], dfn[y]), max(dfn[x], dfn[y]) +
       1);
  int lca = (dep[x] < dep[y] ? x : y);
  return res;
}
```

4.8 Centroid Decomposition

```
const int maxn = 2e5 + 5;
vector<int> adj[maxn], cand;
int sz[maxn], mx[maxn], vis[maxn], cnt[maxn];
int max_dep;

int n, k;
ll ans = 0;

void get_size(int u, int fa) {
  cand.push_back(u);
  sz[u] = 1, mx[u] = 0;
  for (int v:adj[u]){
    if (vis[v] || v == fa) continue;
    get_size(v, u);
    mx[u] = max(mx[u], sz[v]);
    sz[u] += sz[v];
  }
}
void get_dis(int u, int fa, bool add, int dep) {
  if (dep > k) return;
  max_dep = max(max_dep, dep);
  if (add) ans += cnt[k - dep];
  else cnt[dep]++;
  for (int v:adj[u]) if (!vis[v] && v != fa) {
    get_dis(v, u, add, dep+1);
  }
}
void dfs(int u) {
  get_size(u, u);
  int c = -1, all = cand.size();
  for (int v:cand){
    if (max(mx[v], all - sz[v]) <= all / 2) c = v;
  }
  cand.clear();
  vis[c] = true;

  max_dep = 0;
  cnt[0] = 1;
  for (int v:adj[c]){
    if (!vis[v]){
      get_dis(v, c, true, 1);
      get_dis(v, c, false, 1);
    }
  }
  fill(cnt + 1, cnt + max_dep + 1, 0);
  for (int v:adj[c]){
    if (!vis[v]){
        dfs(v);
    }
  }
}
```

4.9 Directed MST

```

struct Edge {
    int u, v, w;
    Edge(int u, int v, int w) : u(u), v(v), w(w) {}
};
const int inf = 1e9;
pair<int, vector<int>> DMST(int n, int r, vector<Edge> e) {
    // e is REVERSE graph
    int ans = 0;
    vector<int> mn(n, inf), out(n, -1), id(n, -1), vis(n, -1);
    for (int i = 0; i < int(e.size()); i++) {
        if (e[i].u != e[i].v && e[i].w < mn[e[i].u]) {
            mn[e[i].u] = e[i].w;
            out[e[i].u] = i;
        }
    }
    for (int i = 0; i < n; i++){
        if (i != r && out[i] == -1){
            return {-1, {}};
        }
    }
    int cnt = 0;
    auto newe(e);
    for (int u = 0; u < n; u++) {
        if (u == r) {
            continue;
        }
        ans += mn[u];
        int x = u;
        while (vis[x] != u && x != r) {
            vis[x] = u;
            x = e[out[x]].v;
        }
        if (x != r && id[x] == -1) {
            id[x] = cnt;
            for (int i = e[out[x]].v; i != x; i = e[out[i]].v)
                 {
                id[i] = cnt;
            }
            cnt++;
        }
    }
    int circ = cnt;
    if (cnt == 0) {
        out.erase(out.begin() + r);
        return {ans, out};
    }
    for (int i = 0; i < n; i++) {
        if (id[i] == -1) {
            id[i] = cnt++;
        }
    }
    for (auto &[u, v, w] : newe) {
        w -= mn[u];
        u = id[u];
        v = id[v];
    }
    auto [newans, edges] = DMST(cnt, id[r], newe);
    if (newans == -1) return {-1, {}};
    ans += newans;
    for (int i = 0; i < cnt - 1; i++) {
        int u = e[edges[i]].u;
        if (id[u] < circ) {
```

```
            for (int i = e[out[u]].v; i != u; i = e[out[i]].v)
                {
                edges.push_back(out[i]);
            }
        }
    }
    return {ans, edges};
}
```

4.10 Dominator Tree

```
vector<int> BuildDominatorTree(vector<vector<int>> g, int s) {
  int N = g.size();
  vector<vector<int>> rdom(N), r(N);
  vector<int> dfn(N, -1), rev(N, -1), fa(N, -1), sdom(N, -1),
      dom(N, -1), val(N, -1), rp(N, -1);
  int stamp = 0;
  auto Dfs = [&](auto dfs, int x) -> void {
    rev[dfn[x] = stamp] = x;
    fa[stamp] = sdom[stamp] = val[stamp] = stamp;
    stamp++;
    for (int u : g[x]) {
      if (dfn[u] == -1) {
        dfs(dfs, u);
        rp[dfn[u]] = dfn[x];
      }
      r[dfn[u]].push_back(dfn[x]);
    }
  };
  function<int(int, int)> Find = [&](int x, int c) {
    if (fa[x] == x) return c ? -1 : x;
    int p = Find(fa[x], 1);
    if (p == -1) return c ? fa[x] : val[x];
    if (sdom[val[x]] > sdom[val[fa[x]]]) val[x] = val[fa[x]];
    fa[x] = p;
    return c ? p : val[x];
  };
  auto Merge = [&](int x, int y) { fa[x] = y; };
  Dfs(Dfs, s);
  for (int i = stamp - 1; i >= 0; --i) {
    for (int u : r[i]) sdom[i] = min(sdom[i], sdom[Find(u, 0)])
        ;
    if (i) rdom[sdom[i]].push_back(i);
    for (int u : rdom[i]) {
      int p = Find(u, 0);
      if (sdom[p] == i) dom[u] = i;
      else dom[u] = p;
    }
    if (i) Merge(i, rp[i]);
  }
  vector<int> res(N, -2);
  res[s] = -1;
  for (int i = 1; i < stamp; ++i) {
    if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
  }
  for (int i = 1; i < stamp; ++i) res[rev[i]] = rev[dom[i]];
  return res;
}
```

4.11 Stoer Wagner Minimum Cut

```
int w[kN][kN], g[kN], del[kN], v[kN];
void AddEdge(int x, int y, int c) {
    w[x][y] += c;
    w[y][x] += c;
}
pair<int, int> Phase(int n) {
    fill(v, v + n, 0), fill(g, g + n, 0);
    int s = -1, t = -1;
    while (true) {
        int c = -1;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            if (c == -1 || g[i] > g[c]) c = i;
        }
        if (c == -1) break;
        v[c] = 1, s = t, t = c;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            g[i] += w[c][i];
        }
    }
    return make_pair(s, t);
}
int GlobalMinCut(int n) {
    int cut = INT_MAX;
    memset(del, 0, sizeof(del));
    for (int i = 0; i < n - 1; ++i) {
        int s, t; //tie(s, t) = Phase(n);
        pair<int, int> pp = Phase(n);
        s = pp.first;t = pp.second;
        del[t] = 1, cut = min(cut, g[t]);
        for (int j = 0; j < n; ++j) {
            w[s][j] += w[t][j];
            w[j][s] += w[j][t];
        }
    }
    return cut;
}
```

4.12 Maximum Clique

```
// change to bitset for n > 64.
int n, deg[maxn];
uint64_t adj[maxn], ans;
vector<pair<int, int>> edge;
void init(int n_) {
    n = n_;
    fill(adj, adj + n, 0ull);
    fill(deg, deg + n, 0);
    edge.clear();
}
void add_edge(int u, int v) {
    edge.emplace_back(u, v);
    ++deg[u], ++deg[v];
}
#define pcount __builtin_popcountll
void dfs(uint64_t r, uint64_t p) {
    if (p == 0) {
        if (pcount(r) > pcount(ans)) ans = r;
        return;
    }
    if (pcount(r | p) <= pcount(ans)) return;
    int x = __builtin_ctzll(p & -p);
    uint64_t c = p & ~adj[x];
    while (c > 0) {
        // bitset._Find_first(); bitset._Find_next();
        x = __builtin_ctzll(c & -c);
        r |= (1ull << x);
        dfs(r, p & adj[x]);
        r &= ~(1ull << x);
        p &= ~(1ull << x);
        c ^= (1ull << x);
    }
}
vector<int> sv() {
    vector<int> ord(n);
    iota(ord.begin(), ord.end(), 0);
    sort(ord.begin(), ord.end(), [&](int u, int v) { return deg
        [u] < deg[v]; });
    vector<int> id(n);
    for (int i = 0; i < n; ++i) id[ord[i]] = i;
    for (auto e : edge) {
        int u = id[e.first], v = id[e.second];
        adj[u] |= (1ull << v);
        adj[v] |= (1ull << u);
    }
    uint64_t r = 0, p = (1ull << n) - 1;
    ans = 0;
    dfs(r, p);
    vector<int> res;
    for (int i = 0; i < n; ++i) {
        if (ans >> i & 1) res.push_back(ord[i]);
    }
    return res;
}
```

4.13 Edge Coloring

```
namespace vizing { // returns edge coloring in adjacent matrix
    G. 1 - based
int C[kN][kN], G[kN][kN];
void clear(int N) {
  for (int i = 0; i <= N; i++) {
    for (int j = 0; j <= N; j++) C[i][j] = G[i][j] = 0;
  }
}
void solve(vector<pair<int, int>> &E, int N) {
  int X[kN] = {}, a;
  auto update = [&](int u) {
    for (X[u] = 1; C[u][X[u]]; X[u]++);
  };
  auto color = [&](int u, int v, int c) {
```

```
    int p = G[u][v];
    G[u][v] = G[v][u] = c;
    C[u][c] = v, C[v][c] = u;
    C[u][p] = C[v][p] = 0;
    if (p) X[u] = X[v] = p;
    else update(u), update(v);
    return p;
  };
  auto flip = [&](int u, int c1, int c2) {
    int p = C[u][c1];
    swap(C[u][c1], C[u][c2]);
    if (p) G[u][p] = G[p][u] = c2;
    if (!C[u][c1]) X[u] = c1;
    if (!C[u][c2]) X[u] = c2;
    return p;
  };
  for (int i = 1; i <= N; i++) X[i] = 1;
  for (int t = 0; t < E.size(); t++) {
    int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u], c
        = c0, d;
    vector<pair<int, int>> L;
    int vst[kN] = {};
    while (!G[u][v0]) {
      L.emplace_back(v, d = X[v]);
      if (!C[v][c]) for (a = (int)L.size() - 1; a >= 0; a--) c
          = color(u, L[a].first, c);
      else if (!C[u][d]) for (a = (int)L.size() - 1; a >= 0; a
          --) color(u, L[a].first, L[a].second);
      else if (vst[d]) break;
      else vst[d] = 1, v = C[u][d];
    }
    if (!G[u][v0]) {
      for (; v; v = flip(v, c, d), swap(c, d));
      if (C[u][c0]) {
        for (a = (int)L.size() - 2; a >= 0 && L[a].second != c;
            a--);
        for (; a >= 0; a--) color(u, L[a].first, L[a].second);
      } else t--;
    }
  }
}}
```

4.14 Mo's on Tree

```
void MoAlgoOnTree() {
  Dfs(0, -1);
  vector<int> euler(tk);
  for (int i = 0; i < n; ++i) {
    euler[tin[i]] = i;
    euler[tout[i]] = i;
  }
  vector<int> l(q), r(q), qr(q), sp(q, -1);
  for (int i = 0; i < q; ++i) {
    if (tin[u[i]] > tin[v[i]]) swap(u[i], v[i]);
    int z = GetLCA(u[i], v[i]);
    sp[i] = z[i];
    if (z == u) l[i] = tin[u[i]], r[i] = tin[v[i]];
    else l[i] = tout[u[i]], r[i] = tin[v[i]];
    qr[i] = i;
  }
  sort(qr.begin(), qr.end(), [&](int i, int j) {
      if (l[i] / kB == l[j] / kB) return r[i] < r[j];
      return l[i] / kB < l[j] / kB;
      });
  vector<bool> used(n);
  // Add(v): add/remove v to/from the path based on used[v]
  for (int i = 0, tl = 0, tr = -1; i < q; ++i) {
    while (tl < l[qr[i]]) Add(euler[tl++]);
    while (tl > l[qr[i]]) Add(euler[--tl]);
    while (tr > r[qr[i]]) Add(euler[tr--]);
    while (tr < r[qr[i]]) Add(euler[++tr]);
    // add/remove LCA(u, v) if necessary
  }
}
```

5 String

5.1 Knuth Morris Pratt Algorithm

```
vector<int> Failure(const string &s) {
  vector<int> f(s.size(), 0);
  // f[i] = length of the longest prefix (excluding s[0:i])
       such that it coincides with the suffix of s[0:i] of the
       same length
  // i + 1 - f[i] is the length of the smallest recurring
       period of s[0:i]
  int k = 0;
  for (int i = 1; i < (int)s.size(); ++i) {
    while (k > 0 && s[i] != s[k]) k = f[k - 1];
    if (s[i] == s[k]) ++k;
    f[i] = k;
  }
  return f;
}
vector<int> Search(const string &s, const string &t) {
  // return 0-indexed occurrence of t in s
  vector<int> f = Failure(t), res;
  for (int i = 0, k = 0; i < (int)s.size(); ++i) {
    while (k > 0 && (k == (int)t.size() || s[i] != t[k])) k = f
        [k - 1];
    if (s[i] == t[k]) ++k;
    if (k == (int)t.size()) res.push_back(i - t.size() + 1);
  }
  return res;
}
```

5.2 Z Algorithm

```
int z[maxn];
// z[i] = LCP of suffix i and suffix 0
void z_function(const string& s) {
  memset(z, 0, sizeof(z));
  z[0] = (int)s.length();
  int l = 0, r = 0;
  for (int i = 1; i < s.length(); ++i) {
    z[i] = max(0, min(z[i - l], r - i + 1));
    while (i + z[i] < s.length() && s[z[i]] == s[i + z[i]]) {
      l = i; r = i + z[i];
      ++z[i];
    }
  }
}
```

5.3 AC Automaton

```
const int M = 300005;
struct acam {
    struct node {
        int fail;
        int ch[26] = { 0 };
    }tr[M];
    int cnt = 0;
    map<int, int>ref;
    vector<int>g[M];//par: node.fail -> child: node
    int ans[M];
    void insert(string& s, int id) {
        int u = 0;
        for (char c : s) {
            if (!tr[u].ch[c - 'a']) {
                tr[u].ch[c - 'a'] = ++cnt;
            }
            u = tr[u].ch[c - 'a'];
        }
        ref[id] = u;
    }
    void build() {
        queue<int>que;
        for (int i = 0; i < 26; i++)
            if (tr[0].ch[i])que.push(tr[0].ch[i]);
        while (que.size()) {
            int u = que.front(); que.pop();
            for (int i = 0; i < 26; i++) {
                if (tr[u].ch[i]) {
                    tr[tr[u].ch[i]].fail = tr[tr[u].fail].ch[i
                        ];
                    que.push(tr[u].ch[i]);
                }
                else tr[u].ch[i] = tr[tr[u].fail].ch[i];
            }
        }
        for (int i = 1; i <= cnt; i++)
            g[tr[i].fail].push_back(i);
    }

    void query(string& S) {
        int u = 0;
        for (char c : S) {
            u = tr[u].ch[c - 'a'];
            ans[u]++;
        }
```

```
    }
    void get_ans() {
        dfs(0);
    }
    void dfs(int u) {
        for (int v : g[u]) {
            dfs(v);
            ans[u] += ans[v];
        }
    }

}trie;
int n;
string T[M], S;
void slove() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> T[i];
        trie.insert(T[i], i);
    }
    trie.build();
    cin >> S;
    trie.query(S);
    trie.get_ans();
    for (int i = 1; i <= n; i++) {
        cout << trie.ans[trie.ref[i]] << endl;
    }
}
```

5.4 Suffix Automaton

```
struct SAM {
  static const int maxn = 5e5 + 5;
  int nxt[maxn][26], to[maxn], len[maxn];
  int root, last, sz;
  int gnode(int x) {
    for (int i = 0; i < 26; ++i) nxt[sz][i] = -1;
    to[sz] = -1;
    len[sz] = x;
    return sz++;
  }
  void init() {
    sz = 0;
    root = gnode(0);
    last = root;
  }
  void push(int c) {
    int cur = last;
    last = gnode(len[last] + 1);
    for (; ~cur && nxt[cur][c] == -1; cur = to[cur]) nxt[cur][c
        ] = last;
    if (cur == -1) return to[last] = root, void();
    int link = nxt[cur][c];
    if (len[link] == len[cur] + 1) return to[last] = link, void
        ();
    int tlink = gnode(len[cur] + 1);
    for (; ~cur && nxt[cur][c] == link; cur = to[cur]) nxt[cur
        ][c] = tlink;
    for (int i = 0; i < 26; ++i) nxt[tlink][i] = nxt[link][i];
    to[tlink] = to[link];
    to[link] = tlink;
    to[last] = tlink;
  }
  void add(const string &s) {
    for (int i = 0; i < s.size(); ++i) push(s[i] - 'a');
  }
  bool find(const string &s) {
    int cur = root;
    for (int i = 0; i < s.size(); ++i) {
      cur = nxt[cur][s[i] - 'a'];
      if (cur == -1) return false;
    }
    return true;
  }
  int solve(const string &t) {
    int res = 0, cnt = 0;
    int cur = root;
    for (int i = 0; i < t.size(); ++i) {
      if (~nxt[cur][t[i] - 'a']) {
        ++cnt;
        cur = nxt[cur][t[i] - 'a'];
      } else {
        for (; ~cur && nxt[cur][t[i] - 'a'] == -1; cur = to[cur
            ]);
        if (~cur) cnt = len[cur] + 1, cur = nxt[cur][t[i] - 'a'
            ];
        else cnt = 0, cur = root;
      }
      res = max(res, cnt);
    }
    return res;
  }
};
```

5.5 Suffix Array

```
// sa[i]: sa[i]-th suffix is the i-th lexigraphically smallest
    suffix.
// lcp[i]: longest common prefix of suffix sa[i] and suffix sa[
    i - 1].
namespace sfx {
vector<int> Build(const string &s) {
  int n = s.size();
  vector<int> str(n * 2), sa(n * 2), c(max(n, 256) * 2), x(max(
      n, 256)), p(n), q(n * 2), t(n * 2);
  for (int i = 0; i < n; ++i) str[i] = s[i];
  auto Pre = [&](int *sa, int *c, int n, int z) {
    memset(sa, 0, sizeof(int) * n);
    memcpy(x.data(), c, sizeof(int) * z);
  };
  auto Induce = [&](int *sa, int *c, int *s, int *t, int n, int
       z) {
    memcpy(x.data() + 1, c, sizeof(int) * (z - 1));
    for (int i = 0; i < n; ++i) if (sa[i] && !t[sa[i] - 1]) sa[
        x[s[sa[i] - 1]]++] = sa[i] - 1;
    memcpy(x.data(), c, sizeof(int) * z);
    for (int i = n - 1; i >= 0; --i) if (sa[i] && t[sa[i] - 1])
        sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
  };
  auto SAIS = [&](auto self, int *s, int *sa, int *p, int *q,
      int *t, int *c, int n, int z) -> void {
    bool uniq = t[n - 1] = true;
    int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n, last =
        -1;
    memset(c, 0, sizeof(int) * z);
    for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
    for (int i = 0; i < z - 1; ++i) c[i + 1] += c[i];
    if (uniq) {
      for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
      return;
    }
    for (int i = n - 2; i >= 0; --i) t[i] = (s[i] == s[i + 1] ?
        t[i + 1] : s[i] < s[i + 1]);
    Pre(sa, c, n, z);
    for (int i = 1; i <= n - 1; ++i) if (t[i] && !t[i - 1]) sa
        [--x[s[i]]] = p[q[i] = nn++] = i;
    Induce(sa, c, s, t, n, z);
    for (int i = 0; i < n; ++i) if (sa[i] && t[sa[i]] && !t[sa[
        i] - 1]) {
      bool neq = last < 0 || memcmp(s + sa[i], s + last, (p[q[
          sa[i]] + 1] - sa[i]) * sizeof(int));
      ns[q[last = sa[i]]] = nmxz += neq;
    }
    self(self, ns, nsa, p + nn, q + n, t + n, c + z, nn, nmxz +
        1);
    Pre(sa, c, n, z);
    for (int i = nn - 1; i >= 0; --i) sa[--x[s[p[nsa[i]]]]] = p
        [nsa[i]];
    Induce(sa, c, s, t, n, z);
  };
  SAIS(SAIS, str.data(), sa.data(), p.data(), q.data(), t.data
      (), c.data(), n + 1, 256);
  return vector<int>(sa.begin() + 1, sa.begin() + n + 1);
}
vector<int> BuildLCP(const vector<int> &sa, const string &s) {
  int n = s.size();
  vector<int> lcp(n), rev(n);
  for (int i = 0; i < n; ++i) rev[sa[i]] = i;
  for (int i = 0, ptr = 0; i < n; ++i) {
    if (!rev[i]) {
      ptr = 0;
      continue;
    }
    while (i + ptr < n && s[i + ptr] == s[sa[rev[i] - 1] + ptr
        ]) ptr++;
    lcp[rev[i]] = ptr ? ptr-- : 0;
  }
  return lcp;
}} // namespace sfx
```

5.6 Least Rotation

```
int least_rotation(string& s) {
    int n = s.length();
```

```
    vi f(2 * n, -1);
    int k = 0;
    for (int j = 1; j < 2 * n; j++) {
        int i = f[j - k - 1];
        while (i != -1 && s[j % n] != s[(k + i + 1) % n]) {
            if (s[j % n] < s[(k + i + 1) % n]) k = j - i - 1;
            i = f[i];
        }
        if (i == -1 && s[j % n] != s[(k + i + 1) % n]) {
            if (s[j % n] < s[(k + i + 1) % n]) k = j;
            f[j - k] = -1;
        }
        else {
            f[j - k] = i + 1;
        }
    }
    return k;
}
```

5.7 Manacher

```
int z[maxn];
int manacher(const string& s) {
  string t = ".";
  for (int i = 0; i < s.length(); ++i) t += s[i], t += '.';
  int l = 0, r = 0, ans = 0;
  for (int i = 1; i < t.length(); ++i) {
    z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
    while (i - z[i] >= 0 && i + z[i] < t.length() && t[i - z[i
        ]] == t[i + z[i]]) ++z[i];
    if (i + z[i] > r) r = i + z[i], l = i;
  }
  for (int i = 1; i < t.length(); ++i) ans = max(ans, z[i] - 1)
      ;
  return ans;
}
```

6 Math

6.1 Sieve of Eratosthenes

```
vector<int> pr, lf;
void comp(int n) {
  for (int i = 0; i < n; i++) {
    lf.push_back(0);
  }
  lf[0] = 1; lf[1] = 1;
  for (int i = 2; i < n; i++) {
    if (lf[i] == 0) {
      pr.push_back(i);
      lf[i] = i;
    }
    for (int p:pr){
      if (i * p >= n || p > lf[i]) break;
      lf[i * p] = p;
    }
  }
}
```

6.2 Euler's Totient Function

```
// O(nloglogn)
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
// O(sqrt(n))
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
```

6.3 Extended GCD

```
int ex_gcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  int d = ex_gcd(b, a % b, x, y);
  int temp = x;
  x = y;
  y = temp - a / b * y;
  return d;
}
bool liEu(int a, int b, int c, int& x, int& y) {
  int d = ex_gcd(a, b, x, y);
  if (c % d != 0) return 0;
  int k = c / d;
  x *= k;
  y *= k;
  return 1;
}
```

6.4 Floor Sum

```
//sum_{0<=i<n} floor((a*i+b)/m)
long long floor_sum(long long n, long long m, long long a, long
      long b) {
    long long ans = 0;
    if (a >= m) {
        ans += (n - 1) * n * (a / m) / 2;
        a %= m;
    }
    if (b >= m) {
        ans += n * (b / m);
        b %= m;
    }

    long long y_max = (a * n + b) / m, x_max = (y_max * m - b);
    if (y_max == 0) return ans;
    ans += (n - (x_max + a - 1) / a) * y_max;
    ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
    return ans;
}
```

6.5 Primitive Root

```
int powmod (int a, int b, int p) {
  int res = 1;
  while (b)
    if (b & 1)
      res = int (res * 1ll * a % p),  --b;
    else
      a = int (a * 1ll * a % p),  b >>= 1;
  return res;
}
int generator (int p) {
  vector<int> fact;
  int phi = p-1,  n = phi;
  for (int i=2; i*i<=n; ++i)
    if (n % i == 0) {
      fact.push_back (i);
      while (n % i == 0)
        n /= i;
    }
  if (n > 1)
    fact.push_back (n);
  for (int res=2; res<=p; ++res) {
    bool ok = true;
    for (size_t i=0; i<fact.size() && ok; ++i)
      ok &= powmod (res, phi / fact[i], p) != 1;
    if (ok)  return res;
  }
  return -1;
}
```

6.6 Discrete Logarithm

```
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
  a %= m, b %= m;
  int k = 1, add = 0, g;
  while ((g = gcd(a, m)) > 1) {
    if (b == k)
      return add;
```

```
    if (b % g)
      return -1;
    b /= g, m /= g, ++add;
    k = (k * 1ll * a / g) % m;
  }

  int n = sqrt(m) + 1;
  int an = 1;
  for (int i = 0; i < n; ++i)
    an = (an * 1ll * a) % m;

  unordered_map<int, int> vals;
  for (int q = 0, cur = b; q <= n; ++q) {
    vals[cur] = q;
    cur = (cur * 1ll * a) % m;
  }

  for (int p = 1, cur = k; p <= n; ++p) {
    cur = (cur * 1ll * an) % m;
    if (vals.count(cur)) {
      int ans = n * p - vals[cur] + add;
      return ans;
    }
  }
  return -1;
}
```

6.7 Discrete Root

```
int gcd(int a, int b) {
    return a ? gcd(b % a, a) : b;
}

int powmod(int a, int b, int p) {
    int res = 1;
    while (b > 0) {
        if (b & 1) {
            res = res * a % p;
        }
        a = a * a % p;
        b >>= 1;
    }
    return res;
}

// Finds the primitive root modulo p
int generator(int p) {
    vector<int> fact;
    int phi = p-1, n = phi;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0)
                n /= i;
        }
    }
    if (n > 1)
        fact.push_back(n);

    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (int factor : fact) {
            if (powmod(res, phi / factor, p) == 1) {
                ok = false;
                break;
            }
        }
        if (ok) return res;
    }
    return -1;
}

// This program finds all numbers x such that x^k = a (mod n)
int main() {
    int n, k, a;
    scanf("%d %d %d", &n, &k, &a);
    if (a == 0) {
        puts("1\n0");
        return 0;
    }

    int g = generator(n);

    // Baby-step giant-step discrete logarithm algorithm
    int sq = (int) sqrt (n + .0) + 1;
    vector<pair<int, int>> dec(sq);
    for (int i = 1; i <= sq; ++i)
        dec[i-1] = {powmod(g, i * sq * k % (n - 1), n), i};
    sort(dec.begin(), dec.end());
    int any_ans = -1;
    for (int i = 0; i < sq; ++i) {
        int my = powmod(g, i * k % (n - 1), n) * a % n;
        auto it = lower_bound(dec.begin(), dec.end(), make_pair
             (my, 0));
        if (it != dec.end() && it->first == my) {
            any_ans = it->second * sq - i;
            break;
        }
    }
    if (any_ans == -1) {
        puts("0");
        return 0;
    }

    // Print all possible answers
    int delta = (n-1) / gcd(k, n-1);
    vector<int> ans;
    for (int cur = any_ans % delta; cur < n-1; cur += delta)
        ans.push_back(powmod(g, cur, n));
    sort(ans.begin(), ans.end());
    printf("%d\n", ans.size());
    for (int answer : ans)
        printf("%d ", answer);
}
```

6.8 Linear Function Mod Min

```
ll topos(ll x, ll m)
{ x %= m; if (x < 0) x += m; return x; }
//min value of ax + b (mod m) for x \in [0, n - 1]. O(log m)
ll min_rem(ll n, ll m, ll a, ll b) {
  a = topos(a, m), b = topos(b, m);
  for (ll g = __gcd(a, m); g > 1;) return g * min_rem(n, m / g,
       a / g, b / g) + (b % g);
  for (ll nn, nm, na, nb; a; n = nn, m = nm, a = na, b = nb) {
    if (a <= m - a) {
      nn = (a * (n - 1) + b) / m;
      if (!nn) break;
      nn += (b < a);
      nm = a, na = topos(-m, a);
      nb = b < a ? b : topos(b - m, a);
    } else {
      ll lst = b - (n - 1) * (m - a);
      if (lst >= 0) {b = lst; break;}
      nn = -(lst / m) + (lst % m < -a) + 1;
      nm = m - a, na = m % (m - a), nb = b % (m - a);
    }
  }
  return b;
}
//min value of ax + b (mod m) for x \in [0, n - 1], also return
     min x to get the value. O(log m)
//{value, x}
pair<ll, ll> min_rem_pos(ll n, ll m, ll a, ll b) {
  a = topos(a, m), b = topos(b, m);
  ll mn = min_rem(n, m, a, b), g = __gcd(a, m);
  //ax = (mn - b) (mod m)
  ll x = (extgcd(a, m).first + m) * ((mn - b + m) / g) % (m / g
       );
  return {mn, x};
}
```

6.9 Fast Inverse

```
const int MAXN = 1000006;
int inv[MAXN];
void invTable(int bound, int p){
    inv[1] = 1;
    for (int i=2; i<bound; i++){
        inv[i] = (long long)inv[p%i] * (p-p/i) %p;
    }
}

int inv(int b, int p){
    if (b==1) return 1;
    return (long long)inv(p%b,p) * (p-p/b) %p;
}
```

6.10 Factorial Mod

```
// O(log_p(n))
int factmod(int n, int p) {
    vector<int> f(p);
    f[0] = 1;
    for (int i = 1; i < p; i++)
        f[i] = f[i-1] * i % p;
```

```
    int res = 1;
    while (n > 1) {
        if ((n/p) % 2)
            res = p - res;
        res = res * f[n%p] % p;
        n /= p;
    }
    return res;
}
```

6.11 Miller Rabin

```
//Deterministic version
long long quick_pow(long long x, long long p, long long mod) {
    long long ans = 1;
    while (p) {
        if (p & 1) ans = (__int128)ans * x % mod;
        x = (__int128)x * x % mod;
        p >>= 1;
    }
    return ans;
}
bool check_composite(ll n, ll a, ll d, int s) {
    ll x = quick_pow(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (__int128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};
bool MillerRabin(ll n) { // returns true if n is prime, else
    returns false.
    if (n < 2)
        return false;
    int r = 0;
    ll d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
        41, 43, 47}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

6.12 Pollard's Rho

```
using LLL = __int128;
ll modpow(ll e, ll p, ll m) {
  ll r = 1;
  while (p) {
    if (p & 1) r = (LLL)r * e % m;
    e = (LLL)e * e % m;
    p >>= 1;
  }
  return r;
}
ll modmul(ll a, ll b, ll m) {
  return (LLL)a * b % m;
}
map<long long, int> cnt;
void PollardRho(long long n) {
  if (n == 1) return;
  if (MillerRabin(n)) return ++cnt[n], void();
  if (n % 2 == 0) return PollardRho(n / 2), ++cnt[2], void();
  long long x = 2, y = 2, d = 1, p = 1;
  auto f = [&](auto x, auto n, int p) { return (modmul(x, x, n)
      + p) % n; };
  while (true) {
    if (d != n && d != 1) {
      PollardRho(n / d);
      PollardRho(d);
      return;
    }
    if (d == n) ++p;
    x = f(x, n, p); y = f(f(y, n, p), n, p);
    d = gcd(abs(x - y), n);
  }
}
```

6.13 Linear Equation

```
void linear_equation(vector<vector<double>> &d, vector<double>
    &aug, vector<double> &sol) {
    int n = d.size(), m = d[0].size();
    vector<int> r(n), c(m);
    iota(r.begin(), r.end(), 0);
    iota(c.begin(), c.end(), 0);
    for (int i = 0; i < m; ++i) {
        int p = -1, z = -1;
        for (int j = i; j < n; ++j) {
            for (int k = i; k < m; ++k) {
                if (fabs(d[r[j]][c[k]]) < eps) continue;
                if (p == -1 || fabs(d[r[j]][c[k]]) > fabs(d[r[p
                    ]][c[z]])) p = j, z = k;
            }
        }
        if (p == -1) continue;
        swap(r[p], r[i]), swap(c[z], c[i]);
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            double z = d[r[j]][c[i]] / d[r[i]][c[i]];
            for (int k = 0; k < m; ++k) d[r[j]][c[k]] -= z * d[
                r[i]][c[k]];
            aug[r[j]] -= z * aug[r[i]];
        }
    }
    vector<vector<double>> fd(n, vector<double>(m));
    vector<double> faug(n), x(n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) fd[i][j] = d[r[i]][c[j]];
        faug[i] = aug[r[i]];
    }
    d = fd, aug = faug;
    for (int i = n - 1; i >= 0; --i) {
        double p = 0.0;
        for (int j = i + 1; j < n; ++j) p += d[i][j] * x[j];
        x[i] = (aug[i] - p) / d[i][i];
    }
    for (int i = 0; i < n; ++i) sol[c[i]] = x[i];
}
```

6.14 Simplex Construction

Standard form: maximize $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$.
Dual LP: minimize $\mathbf{b}^T\mathbf{y}$ subject to $A^T\mathbf{y} \geq \mathbf{c}$ and $\mathbf{y} \geq 0$.
$\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ are optimal if and only if for all $i \in [1, n]$, either $\bar{x}_i = 0$ or $\sum_{j=1}^{m} A_{ji}\bar{y}_j = c_i$ holds and for all $i \in [1, m]$ either $\bar{y}_i = 0$ or $\sum_{j=1}^{n} A_{ij}\bar{x}_j = b_j$ holds.

1. In case of minimization, let $c'_i = -c_i$
2. $\sum_{1 \leq i \leq n} A_{ji}x_i \geq b_j \rightarrow \sum_{1 \leq i \leq n} -A_{ji}x_i \leq -b_j$
3. $\sum_{1 \leq i \leq n} A_{ji}x_i = b_j$
   - $\sum_{1 \leq i \leq n} A_{ji}x_i \leq b_j$
   - $\sum_{1 \leq i \leq n} A_{ji}x_i \geq b_j$
4. If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.15 Simplex

```
const double eps = 1e-9;
const double inf = 1e+9;
int n, m;
vector<vector<double>> d;
vector<int> p, q;
void pivot(int r, int s) {
  double inv = 1.0 / d[r][s];
  for (int i = 0; i < m + 2; ++i) {
    for (int j = 0; j < n + 2; ++j) {
      if (i != r && j != s) d[i][j] -= d[r][j] * d[i][s] * inv;
    }
  }
  for (int i = 0; i < m + 2; ++i) if (i != r) d[i][s] *= -inv;
  for (int j = 0; j < n + 2; ++j) if (j != s) d[r][j] *= +inv;
  d[r][s] = inv;
  swap(p[r], q[s]);
}
bool phase(int z) {
  int x = m + z;
  while (true) {
    int s = -1;
    for (int i = 0; i <= n; ++i) {
      if (!z && q[i] == -1) continue;
      if (s == -1 || d[x][i] < d[x][s]) s = i;
    }
    if (d[x][s] > -eps) return true;
```

```
    int r = -1;
    for (int i = 0; i < m; ++i) {
      if (d[i][s] < eps) continue;
      if (r == -1 || d[i][n + 1] / d[i][s] < d[r][n + 1] / d[r
          ][s]) r = i;
    }
    if (r == -1) return false;
    pivot(r, s);
  }
}
vector<double> solve(const vector<vector<double>> &a, const
    vector<double> &b, const vector<double> &c) {
  m = b.size(), n = c.size();
  d = vector<vector<double>>(m + 2, vector<double>(n + 2));
  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) d[i][j] = a[i][j];
  }
  p.resize(m), q.resize(n + 1);
  for (int i = 0; i < m; ++i) p[i] = n + i, d[i][n] = -1, d[i][
      n + 1] = b[i];
  for (int i = 0; i < n; ++i) q[i] = i, d[m][i] = -c[i];
  q[n] = -1, d[m + 1][n] = 1;
  int r = 0;
  for (int i = 1; i < m; ++i) if (d[i][n + 1] < d[r][n + 1]) r
      = i;
  if (d[r][n + 1] < -eps) {
    pivot(r, n);
    if (!phase(1) || d[m + 1][n + 1] < -eps) return vector<
        double>(n, -inf);
    for (int i = 0; i < m; ++i) if (p[i] == -1) {
      int s = min_element(d[i].begin(), d[i].end() - 1) - d[i].
          begin();
      pivot(i, s);
    }
  }
  if (!phase(0)) return vector<double>(n, inf);
  vector<double> x(n);
  for (int i = 0; i < m; ++i) if (p[i] < n) x[p[i]] = d[i][n +
      1];
  return x;
}
```

6.16 NTT

```

constexpr int kMod = 998244353;
constexpr int kN = 1<<20;
constexpr int kRoot = 3;
constexpr int kP = kMod;

int fpow(int a, int n) {
  int res = 1;
  while (n > 0) {
    if (n & 1) {
      res = 1LL * res * a % kP;
    }
    a = 1LL * a * a % kP;
    n >>= 1;
  }
  return res;
}

namespace NTT {

vector<int> omega;
void Init() {
  omega.resize(kN + 1);
  long long x = fpow(kRoot, (kMod - 1) / kN);
  omega[0] = 1;
  for (int i = 1; i <= kN; ++i) {
    omega[i] = 1LL * omega[i - 1] * x % kMod;
  }
}

void BitReverse(vector<int> &v, int n) {
  int z = __builtin_ctz(n) - 1;
  for (int i = 0; i < n; ++i) {
    int x = 0;
    for (int j = 0; (1 << j) < n; ++j) {
      x ^= (i >> j & 1) << (z - j);
    }
    if (x > i) {
      swap(v[x], v[i]);
    }
  }
}

void Transform(vector<int> &v, int n) {
  BitReverse(v, n);
  for (int s = 2; s <= n; s <<= 1) {
    int z = s >> 1;
    for (int i = 0; i < n; i += s) {
      for (int k = 0; k < z; ++k) {
        int x = 1LL * v[i + k + z] * omega[kN / s * k] % kMod;
        v[i + k + z] = (v[i + k] + kMod - x) % kMod;
        (v[i + k] += x) %= kMod;
      }
    }
  }
}

void InverseTransform(vector<int> &v, int n) {
  Transform(v, n);
  for (int i = 1; i < n / 2; ++i) swap(v[i], v[n - i]);
  const int kInv = fpow(n, kMod - 2);
  for (int i = 0; i < n; ++i) v[i] = 1LL * v[i] * kInv % kMod;
}

vector<int> Multiply(vector<int> a, vector<int> b) {
  int n = a.size(), m = b.size();
  int s = 1;
  while (s < n + m - 1) {
    s <<= 1;
  }
  a.resize(s);
  b.resize(s);
  Transform(a, s);
  Transform(b, s);
  for (int i = 0; i < s; ++i) {
    a[i] = 1LL * a[i] * b[i] % kP;
  }
  InverseTransform(a, s);
  a.resize(n + m - 1);
  return a;
}
}  // namespace NTT

// remember to call NNT::Init()
// check kN >= n + m && kN = 2 ^ p
```

6.17 NTT Prime

Prime	Root	Prime	Root
7681	17	167772161	3
12289	11	104857601	3
40961	3	985661441	3
65537	3	998244353	3
786433	10	1107296257	10
5767169	3	2013265921	31
7340033	3	2810183681	11
23068673	3	2885681153	3
469762049	3	605028353	3

6.18 FFT

```
typedef complex<double> cd;
void fft(vector<cd> &a)
{
    int n=a.size(),L=31-__builtin_clz(n);
    vector<int> rev(n);
    for(int i=0;i<n;i++) rev[i]=(rev[i/2]+((i&1)<<L))/2;
    for(int i=0;i<n;i++) if(i<rev[i]) swap(a[i],a[rev[i]]);
    static vector<complex<long double>> R(2,1);
    static vector<cd> rt(2,1);
    for(static int k=2;k<n;k*=2)
    {
        R.resize(n); rt.resize(n);
        auto z=polar(1.0L,acos(-1.0L)/k);
        for(int i=k;i<2*k;i++) rt[i]=R[i]=R[i/2]*((i&1)?z:1);
    }
    for(int k=1;k<n;k*=2)
    {
        for(int i=0;i<n;i+=2*k)
        {
            for(int j=0;j<k;j++)
            {
                cd z=rt[j+k]*a[i+j+k];
                a[i+j+k]=a[i+j]-z;
                a[i+j]+=z;
            }
        }
    }
}

template<ll mod> vector<ll> mul_mod(vector<ll> &a,vector<ll> &b
    )
```

```
{
    int sa=a.size(),sb=b.size();
    if(sa==0||sb==0) return {};
    int n=1<<(32-__builtin_clz(sa+sb-2)),S=(1<<15);
    vector<cd> f(n),g(n),hs(n),hl(n);
    for(int i=0;i<sa;i++) f[i]=cd(a[i]&(S-1),(a[i]>>15));
    for(int i=0;i<sb;i++) g[i]=cd(b[i]&(S-1),(b[i]>>15));
    fft(f); fft(g);
    for(int i=0;i<n;i++)
    {
        int j=(-i)&(n-1);
        hs[i]=(f[j]+conj(f[i]))*g[j]/(2.0*n);
        hl[i]=(f[j]-conj(f[i]))*conj(g[i])/(2.0*n);
    }
    fft(hs); fft(hl);
    vector<ll> c(sa+sb-1);
    for(int i=0;i<sa+sb-1;i++)
    {
        ll ah=ll(real(hl[i])+0.5),ch=ll(real(hs[i])+0.5);
        ll bh=ll(imag(hl[i])+0.5)+ll(imag(hs[i])+0.5);
        c[i]=((ah%mod*S+bh)%mod*S+ch)%mod;
    }
    return c;
}
```

6.19 XOR Convolution

```
const int mod = 1e9 + 7;
vector<long long> fwt(vector<long long> A, bool inv){
  int N = A.size();
  for (int i = 1; i < N; i <<= 1){
    for (int j = 0; j < N; j++){
      if ((j & i) == 0){
        long long x = A[j];
        long long y = A[j | i];
        A[j] = (x + y) % mod;
        A[j | i] = (x - y + mod) % mod;
        if (inv){
          A[j] *= (mod + 1) / 2;
          A[j] %= mod;
          A[j | i] *= (mod + 1) / 2;
          A[j | i] %= mod;
        }
      }
    }
  }
  return A;
}
vector<long long> xor_convolution(vector<long long> A, vector<
    long long> B){
  int N = A.size();
  A = fwt(A, false);
  B = fwt(B, false);
  vector<long long> C(N);
  for (int i = 0; i < N; i++){
    C[i] = A[i] * B[i] % mod;
  }
  C = fwt(C, true);
  return C;
}
```

6.20 SG

```
Anti Nim (                    )

         if and only if
1.                        1          SG      0
2.                        1          SG        0

-----------------------------------------------------

Anti-SG (                  )

     SG      0
           if and only if
1.                      SG          1          SG        0
2.                      SG          1          SG          0

-----------------------------------------------------
Sprague-Grundy

1.
2.
3.
4.
5.
6.

SG(S)        0      (P)
     0      (N)


int mex(set S) {
  // find the min number >= 0 that not in the S
  // e.g. S = {0, 1, 3, 4} mex(S) = 2
}

state = []
int SG(A) {
  if (A not in state) {
    S = sub_states(A)
    if( len(S) > 1 ) state[A] = reduce(operator.xor, [SG(B) for
         B in S])
    else state[A] = mex(set(SG(B) for B in next_states(A)))
  }
  return state[A]
}
```

6.21 Theorem

```
/*
Lucas's Theorem
  For non-negative integer n,m and prime P,
  C(m,n) mod P = C(m/M,n/M) * C(m%M,n%M) mod P
  = mult_i ( C(m_i,n_i) )
  where m_i is the i-th digit of m in base P.
-----------------------------------------------------
Kirchhoff's theorem
  A_{ii} = deg(i), A_{ij} = (i,j) \in E ? -1 : 0
  Deleting any one row, one column, and cal the det(A)
-----------------------------------------------------
Nth Catalan recursive function:
C_0 = 1, C_{n+1} = C_n * 2(2n + 1)/(n+2)
-----------------------------------------------------
Mobius Formula
u(n) = 1        , if n = 1
       (-1)^m   ,    n                  n = p1*p2*p3*...*pk
       0        ,    n        1
- Property
1. (         ) u(a)u(b) = u(ab)
2. ∑_{d|n} u(d) = [n == 1]
-----------------------------------------------------
Mobius Inversion Formula
if      f(n) = ∑_{d|n} g(d)
then    g(n) = ∑_{d|n} u(n/d)f(d)
             = ∑_{d|n} u(d)f(n/d)
- Application
the number/power of gcd(i, j) = k
- Trick
    , O(sqrt(n))
-----------------------------------------------------
Chinese Remainder Theorem (m_i         )
  x = a_1 (mod m_1)
  x = a_2 (mod m_2)
  ....
  x = a_i (mod m_i)
construct a solution:
  Let M = m_1 * m_2 * m_3 * ... * m_n
  Let M_i = M / m_i
  t_i = 1 / M_i
  t_i * M_i = 1 (mod m_i)
  solution x = a_1 * t_1 * M_1 + a_2 * t_2 * M_2 + ... + a_n *
       t_n * M_n + k * M
  = k*M + ∑ a_i * t_i * M_i, k is positive integer.
  under mod M, there is one solution x = ∑ a_i * t_i * M_i
-----------------------------------------------------
Burnside's lemma
|G| * |X/G|  = sum( |X^g| ) where g in G
        :
*/
```

6.22 Mobius Inversion

```
int mu[kC], dv[kC];
vector<int> prime;
void Sieve() {
  mu[1] = dv[1] = 1;
  for (int i = 2; i < kC; ++i) {
    if (!dv[i]) {
      dv[i] = i, mu[i] = -1;
      prime.push_back(i);
    }
    for (int j = 0; i * prime[j] < kC; ++j) {
      dv[i * prime[j]] = prime[j];
      mu[i * prime[j]] = -mu[i];
```

```
      if (i % prime[j] == 0) {
        mu[i * prime[j]] = 0;
        break;
      }
    }
  }
}
```

6.23 Chinese Remainder Theorem

```
template <typename T> tuple<T, T, T> extgcd(T a, T b) {
  if (!b) return make_tuple(a, 1, 0);
  T d, x, y;
  tie(d, x, y) = extgcd(b, a % b);
  return make_tuple(d, y, x - (a / b) * y);
}
long long crt(vector<int> mod, vector<int> a) {
  long long mult = mod[0];
  int n = (int)mod.size();
  long long res = a[0];
  for (int i = 1; i < n; ++i) {
    long long d, x, y;
    tie(d, x, y) = extgcd(mult, mod[i] * 1ll);
    if ((a[i] - res) % d) return -1;
    long long new_mult = mult / __gcd(mult, 1ll * mod[i]) * mod
        [i];
    res += x * ((a[i] - res) / d) % new_mult * mult % new_mult;
    mult = new_mult;
    ((res %= mult) += mult) %= mult;
  }
  return res;
}
```

7 Geometry

7.1 Basic

```
bool same(double a, double b) { return abs(a - b) < eps; }

struct P {
  double x, y;
  P() : x(0), y(0) {}
  P(double x, double y) : x(x), y(y) {}
  P operator + (P b) { return P(x + b.x, y + b.y); }
  P operator - (P b) { return P(x - b.x, y - b.y); }
  P operator * (double b) { return P(x * b, y * b); }
  P operator / (double b) { return P(x / b, y / b); }
  double operator * (P b) { return x * b.x + y * b.y; }
  double operator ^ (P b) { return x * b.y - y * b.x; }
  double abs() { return hypot(x, y); }
  P unit() { return *this / abs(); }
  P rot(double o) {
    double c = cos(o), s = sin(o);
    return P(c * x - s * y, s * x + c * y);
  }
  double angle() { return atan2(y, x); }
};

struct L {
  // ax + by + c = 0
  double a, b, c, o;
  P pa, pb;
  L() : a(0), b(0), c(0), o(0), pa(), pb() {}
  L(P pa, P pb) : a(pa.y - pb.y), b(pb.x - pa.x), c(pa ^ pb), o
      (atan2(-a, b)), pa(pa), pb(pb) {}
  P project(P p) { return pa + (pb - pa).unit() * ((pb - pa) *
      (p - pa) / (pb - pa).abs()); }
  P reflect(P p) { return p + (project(p) - p) * 2; }
  double get_ratio(P p) { return (p - pa) * (pb - pa) / ((pb -
      pa).abs() * (pb - pa).abs()); }
};

bool SegmentIntersect(P p1, P p2, P p3, P p4) {
  if (max(p1.x, p2.x) < min(p3.x, p4.x) || max(p3.x, p4.x) <
      min(p1.x, p2.x)) return false;
  if (max(p1.y, p2.y) < min(p3.y, p4.y) || max(p3.y, p4.y) <
      min(p1.y, p2.y)) return false;
  return sign((p3 - p1) ^ (p4 - p1)) * sign((p3 - p2) ^ (p4 -
      p2)) <= 0 &&
    sign((p1 - p3) ^ (p2 - p3)) * sign((p1 - p4) ^ (p2 - p4))
        <= 0;
}

bool parallel(L x, L y) { return same(x.a * y.b, x.b * y.a); }

P Intersect(L x, L y) { return P(-x.b * y.c + x.c * y.b, x.a *
    y.c - x.c * y.a) / (-x.a * y.b + x.b * y.a); }
```

7.2 Half Plane Intersection

```
bool jizz(L l1,L l2,L l3){
  P p=Intersect(l2,l3);
  return ((l1.pb-l1.pa)^(p-l1.pa))<-eps;
}

bool cmp(const L &a,const L &b){
  return same(a.o,b.o)?(((b.pb-b.pa)^(a.pb-b.pa))>eps):a.o<b.o;
}

// availble area for L l is (l.pb-l.pa)^(p-l.pa)>0
vector<P> HPI(vector<L> &ls){
  sort(ls.begin(),ls.end(),cmp);
  vector<L> pls(1,ls[0]);
  for(int i=0;i<(int)ls.size();++i)if(!same(ls[i].o,pls.back().
      o))pls.push_back(ls[i]);
  deque<int> dq; dq.push_back(0); dq.push_back(1);
#define meow(a,b,c) while(dq.size()>1u && jizz(pls[a],pls[b],
    pls[c]))
  for(int i=2;i<(int)pls.size();++i){
    meow(i,dq.back(),dq[dq.size()-2])dq.pop_back();
    meow(i,dq[0],dq[1])dq.pop_front();
    dq.push_back(i);
  }
  meow(dq.front(),dq.back(),dq[dq.size()-2])dq.pop_back();
  meow(dq.back(),dq[0],dq[1])dq.pop_front();
  if(dq.size()<3u)return vector<P>(); // no solution or
      solution is not a convex
  vector<P> rt;
  for(int i=0;i<(int)dq.size();++i)rt.push_back(Intersect(pls[
      dq[i]],pls[dq[(i+1)%dq.size()]]));
  return rt;
}
```

7.3 Slope & Fraction

```
struct P {
  int x, y, i;
  P() : x(0), y(0), i(-1) {}
};
struct Frac {
  int u, d;
  void norm() {
    if (d == 0) {
      u = u > 0 ? 1 : u < 0 ? -1 : 0;
      return;
    }
    int g = __gcd(u, d);
    u /= g;
    d /= g;
    if (d < 0) {
      d *= -1;
      u *= -1;
    }
  }
};
bool operator > (const Frac &a, const Frac &b) {
  return 1ll * a.u * b.d > 1ll * b.u * a.d;
}
bool operator >= (const Frac &a, const Frac &b) {
  return 1ll * a.u * b.d >= 1ll * b.u * a.d;
}
bool operator < (const Frac &a, const Frac &b) {
  return 1ll * a.u * b.d < 1ll * b.u * a.d;
}
bool operator <= (const Frac &a, const Frac &b) {
  return 1ll * a.u * b.d <= 1ll * b.u * a.d;
}
ostream& operator << (ostream &o, const Frac &f) {
  o << f.u << "/" << f.d;
  return o;
}

Frac Slope(P &a, P &b) {
  Frac f;
  f.u = b.y - a.y;
  f.d = b.x - a.x;
  f.norm();
  return f;
}
```

7.4 Convex order

```
int quard(P p) {
    if (p.x > 0 && p.y >= 0) return 1;
    if (p.x <= 0 && p.y > 0) return 2;
    if (p.x < 0 && p.y <= 0) return 3;
```

```
    if (p.x >= 0 && p.y < 0) return 4;
    return -1;
}
P getcenter(vector<P>& p) {
    P res(0, 0); double n = (double)p.size();
    for (P it : p) res.x += it.x, res.y += it.y;
    res.x /= n; res.y /= n;
    return res;
}
void convex_order(vector<P>& p) {
    P center = getcenter(p);
    auto cmp = [&](P a, P b) {
        P tmpa = a - center,tmpb = b - center;
        int qa = quard(tmpa), qb = quard(tmpb);
        if (qa != qb) return qa < qb;
        return (tmpa ^ tmpb) > 0;
    };
    sort(ALL(p), cmp);
}
```

7.5 Convex Hull

```
vector<P> convex_hull(vector<P>& ps, int n) {
  auto cmp_x = [&](P& p, P& q) {
    if (p.x != q.x) return p.x < q.x;
    return p.y < q.y;
  };
  sort(ps.begin(), ps.begin() + n, cmp_x);
  int k = 0;
  vector<P> qs(n * 2);
  for (int i = 0; i < n; i++) {
    while (k > 1 && (((qs[k - 1] - qs[k - 2]) ^ (ps[i] - qs[k -
          1])) <= 0)) k--;
    qs[k++] = ps[i];
  }
  for (int i = n - 2, t = k; i >= 0; i--) {
    while (k > t && (((qs[k - 1] - qs[k - 2]) ^ (ps[i] - qs[k -
          1])) <= 0)) k--;
    qs[k++] = ps[i];
  }
  qs.resize(k - 1);
  return qs;
}
```

7.6 Area

```
double area(const vector<P>& fig) {
    double res = 0;
    for (unsigned i = 0; i < fig.size(); i++) {
        P p = i ? fig[i - 1] : fig.back();
        P q = fig[i];
        res += (p.x - q.x) * (p.y + q.y);
    }
    return fabs(res) / 2;
}
```

7.7 Circle

```
struct C {
  P c;
  double r;
  C(P c = P(0, 0), double r = 0) : c(c), r(r) {}
};

vector<P> Intersect(C a, C b) {
  if (a.r > b.r) swap(a, b);
  double d = (a.c - b.c).abs();
  vector<P> p;
  if (same(a.r + b.r, d)) p.push_back(a.c + (b.c - a.c).unit()
       * a.r);
  else if (a.r + b.r > d && d + a.r >= b.r) {
    double o = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d)
         );
    P i = (b.c - a.c).unit();
    p.push_back(a.c + i.rot(o) * a.r);
    p.push_back(a.c + i.rot(-o) * a.r);
  }
  return p;
}
double IntersectArea(C a, C b) {
  if (a.r > b.r) swap(a, b);
  double d = (a.c - b.c).abs();
  if (d >= a.r + b.r - eps) return 0;
  if (d + a.r <= b.r + eps) return sq(a.r) * acos(-1);
  double p = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d));
  double q = acos((sq(b.r) + sq(d) - sq(a.r)) / (2 * b.r * d));
  return p * sq(a.r) + q * sq(b.r) - a.r * d * sin(p);
}
// remove second level if to get points for line (defalut:
    segment)
vector<P> CircleCrossLine(P a, P b, P o, double r) {
  double x = b.x - a.x, y = b.y - a.y, A = sq(x) + sq(y), B = 2
       * x * (a.x - o.x) + 2 * y * (a.y - o.y);
  double C = sq(a.x - o.x) + sq(a.y - o.y) - sq(r), d = B * B -
       4 * A * C;
  vector<P> t;
  if (d >= -eps) {
    d = max(0., d);
    double i = (-B - sqrt(d)) / (2 * A);
    double j = (-B + sqrt(d)) / (2 * A);
    if (i - 1.0 <= eps && i >= -eps) t.emplace_back(a.x + i * x
         , a.y + i * y);
    if (j - 1.0 <= eps && j >= -eps) t.emplace_back(a.x + j * x
         , a.y + j * y);
  }
  return t;
}
// calc area intersect by circle with radius r and triangle OAB
double AreaOfCircleTriangle(P a, P b, double r) {
  bool ina = a.abs() < r, inb = b.abs() < r;
  auto p = CircleCrossLine(a, b, P(0, 0), r);
  if (ina) {
    if (inb) return abs(a ^ b) / 2;
    return SectorArea(b, p[0], r) + abs(a ^ p[0]) / 2;
  }
  if (inb) return SectorArea(p[0], a, r) + abs(p[0] ^ b) / 2;
  if (p.size() == 2u) return SectorArea(a, p[0], r) +
       SectorArea(p[1], b, r) + abs(p[0] ^ p[1]) / 2;
  else return SectorArea(a, b, r);
}
// for any triangle
double AreaOfCircleTriangle(vector<P> ps, double r) {
  double ans = 0;
  for (int i = 0; i < 3; ++i) {
    int j = (i + 1) % 3;
    double o = atan2(ps[i].y, ps[i].x) - atan2(ps[j].y, ps[j].x
         );
    if (o >= pi) o = o - 2 * pi;
    if (o <= -pi) o = o + 2 * pi;
    ans += AreaOfCircleTriangle(ps[i], ps[j], r) * (o >= 0 ? 1
         : -1);
  }
  return abs(ans);
}
```

7.8 Area of Union of Circles

```
vector<pair<double, double>> CoverSegment(C &a, C &b) {
  double d = (a.c - b.c).abs();
  vector<pair<double, double>> res;
  if (same(a.r + b.r, d)) ;
  else if (d <= abs(a.r - b.r) + eps) {
    if (a.r < b.r) res.emplace_back(0, 2 * pi);
  } else if (d < abs(a.r + b.r) - eps) {
    double o = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d)
         ), z = (b.c - a.c).angle();
    if (z < 0) z += 2 * pi;
    double l = z - o, r = z + o;
    if (l < 0) l += 2 * pi;
    if (r > 2 * pi) r -= 2 * pi;
    if (l > r) res.emplace_back(l, 2 * pi), res.emplace_back(0,
         r);
    else res.emplace_back(l, r);
  }
  return res;
}
double CircleUnionArea(vector<C> c) { // circle should be
    identical
  int n = c.size();
  double a = 0, w;
  for (int i = 0; w = 0, i < n; ++i) {
    vector<pair<double, double>> s = {{2 * pi, 9}}, z;
    for (int j = 0; j < n; ++j) if (i != j) {
      z = CoverSegment(c[i], c[j]);
      for (auto &e : z) s.push_back(e);
    }
    sort(s.begin(), s.end());
    auto F = [&] (double t) { return c[i].r * (c[i].r * t + c[i
         ].c.x * sin(t) - c[i].c.y * cos(t)); };
    for (auto &e : s) {
      if (e.first > w) a += F(e.first) - F(w);
      w = max(w, e.second);
    }
  }
  return a * 0.5;
```

```
}
```

7.9 Minimun Distance of 2 Polygons

```
// p, q is convex
int sign(double a){
    return fabs(a) < eps ? 0 : a > 0 ? 1 : -1;
}
double PointSegDist(pdd q0, pdd q1, pdd p) {
    if (sign(abs(q0 - q1)) == 0) return abs(q0 - p);
    if (sign(dot(q1 - q0, p - q0)) >= 0 && sign(dot(q0 - q1, p
         - q1)) >= 0)
        return fabs(cross(q1 - q0, p - q0) / abs(q0 - q1));
    return min(abs(p - q0), abs(p - q1));
}
double TwoConvexHullMinDist(vector<pll> A, vector<pll> B) {
    for (auto& p : B) p = { -p.X, -p.Y };
    vector<pll> C = minkowski(A, B); // assert SZ(C) > 0
    if (PointInConvex(C, pll(0, 0))) return 0;
    double ans = PointSegDist(C.back(), C[0], pll(0, 0));
    for (int i = 0; i + 1 < C.size(); ++i) {
        ans = min(ans, PointSegDist(C[i], C[i + 1], pll(0, 0)))
            ;
    }
    return ans;
}
```

7.10 Closest Pair

```
double closest_pair(int l, int r) {
  // p should be sorted increasingly according to the x-
      coordinates.
  if (l == r) return 1e9;
  if (r - l == 1) return dist(p[l], p[r]);
  int m = (l + r) >> 1;
  double d = min(closest_pair(l, m), closest_pair(m + 1, r));
  vector<int> vec;
  for (int i = m; i >= l && fabs(p[m].x - p[i].x) < d; --i) vec
       .push_back(i);
  for (int i = m + 1; i <= r && fabs(p[m].x - p[i].x) < d; ++i)
       vec.push_back(i);
  sort(vec.begin(), vec.end(), [&](int a, int b) { return p[a].
      y < p[b].y; });
  for (int i = 0; i < vec.size(); ++i) {
    for (int j = i + 1; j < vec.size() && fabs(p[vec[j]].y - p[
        vec[i]].y) < d; ++j) {
      d = min(d, dist(p[vec[i]], p[vec[j]]));
    }
  }
  return d;
}
```

7.11 Minkowski Sum

```
void reorder_polygon(vector<P>& P) {
    size_t pos = 0;
    for (size_t i = 1; i < P.size(); i++) {
        if (P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x
            < P[pos].x))
            pos = i;
    }
    rotate(P.begin(), P.begin() + pos, P.end());
}
vector<P> minkowski(vector<P> p, vector<P> q) {
    // the first vertex must be the lowest
    reorder_polygon(p);
    reorder_polygon(q);
    // we must ensure cyclic indexing
    p.push_back(p[0]);
    p.push_back(p[1]);
    q.push_back(q[0]);
    q.push_back(q[1]);
    // main part
    vector<P> result;
    size_t i = 0, j = 0;
    while (i < p.size() - 2 || j < q.size() - 2) {
        result.push_back(p[i] + q[j]);
        auto cross = (p[i + 1] - p[i]) ^ (q[j + 1] - q[j]);
        if (cross >= 0 && i < p.size() - 2) ++i;
        if (cross <= 0 && j < q.size() - 2) ++j;
    }
    return result;
}
```

7.12 Point in Polygon

```
//can use it to check if line is entirely in polygon
//if line intersect at an edge of polygon => false
//then check midpoint(maybe more points) is in polygon
int ori(pdd a, pdd b, pdd c){
    return sign(cross(b - a, c - a));
}
bool collinearity(pdd p1, pdd p2, pdd p3){
    return sign(cross(p1 - p3, p2 - p3)) == 0;
}
bool btw(pdd p1, pdd p2, pdd p3) {
    if (!collinearity(p1, p2, p3)) return 0;
    return sign(dot(p1 - p3, p2 - p3)) <= 0;
}
bool PointInConvex(const vector<pll>& C, pll p, bool strict =
     true) {
    int a = 1, b = SZ(C) - 1, r = !strict;
    if (SZ(C) == 0) return false;
    if (SZ(C) < 3) return r && btw(C[0], C.back(), p);
    if (ori(C[0], C[a], C[b]) > 0) swap(a, b);
    if (ori(C[0], C[a], p) >= r || ori(C[0], C[b], p) <= -r)
        return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (ori(C[0], C[c], p) > 0 ? b : a) = c;
    }
    return ori(C[a], C[b], p) < r;
}
```

8 Dynamic Programming

8.1 DP-Optimization

```
Monotonicity & 1D/1D DP & 2D/1D DP
-------------------------------------------------------
Definition xD/yD
1D/1D DP[j] = min(0≤i<j) { DP[i] + w(i, j) }; DP[0] = k
2D/1D DP[i][j] = min(i<k≤j) { DP[i][k - 1] + DP[k][j] } + w(i,
     j); DP[i][i] = 0
-------------------------------------------------------
Monotonicity
       c       d
  ----------------
a | w(a, c) w(a, d)
b | w(b, c) w(b, d)

Monge Condition
Concave(              ): w(a, c) + w(b, d) >= w(a, d) + w(b, c)
Convex (              ): w(a, c) + w(b, d) <= w(a, d) + w(b, c)

Totally Monotone
Concave(      ): w(a, c) <= w(b, d) -----> w(a, d) <= w(b, c)
Convex (      ): w(a, c) >= w(b, d) -----> w(a, d) >= w(b, c)
-------------------------------------------------------
1D/1D DP O(n^2) -> O(nlgn)
**CONSIDER THE TRANSITION POINT**
Solve 1D/1D Concave by Stack
Solve 1D/1D Convex by Deque
-------------------------------------------------------
2D/1D Convex DP (Totally Monotone) O(n^3) -> O(n^2)
h(i, j - 1) ≤ h(i, j) ≤ h(i + 1, j)
```

8.2 DP $1D/1D$

```
#include<bits/stdc++.h>

int t, n, L;
int p;
char s[MAXN][35];
ll sum[MAXN] = {0};
long double dp[MAXN] = {0};
int prevd[MAXN] = {0};

long double pw(long double a, int n) {
    if ( n == 1 ) return a;
    long double b = pw(a, n/2);
    if ( n & 1 ) return b*b*a;
    else return b*b;
}
long double f(int i, int j) {
//    cout << (sum[i] - sum[j]+i-j-1-L) << endl;
    return pw(abs(sum[i] - sum[j]+i-j-1-L), p) + dp[j];
}
struct INV {
    int L, R, pos;
};
```

```
INV stk[MAXN*10];
int top = 1, bot = 1;
void update(int i) {
    while ( top > bot && i < stk[top].L && f(stk[top].L, i) < f
         (stk[top].L, stk[top].pos) ) {
        stk[top - 1].R = stk[top].R;
        top--;
    }
    int lo = stk[top].L, hi = stk[top].R, mid, pos = stk[top].
         pos;
    //if ( i >= lo ) lo = i + 1;
    while ( lo != hi ) {
        mid = lo + (hi - lo) / 2;
        if ( f(mid, i) < f(mid, pos) ) hi = mid;
        else lo = mid + 1;
    }
    if ( hi < stk[top].R ) {
        stk[top + 1] = (INV) { hi, stk[top].R, i };
        stk[top++].R = hi;
    }
}

int main() {
    cin >> t;
    while ( t-- ) {
        cin >> n >> L >> p;
        dp[0] = sum[0] = 0;
        for ( int i = 1 ; i <= n ; i++ ) {
            cin >> s[i];
            sum[i] = sum[i-1] + strlen(s[i]);
            dp[i] = numeric_limits<long double>::max();
        }
        stk[top] = (INV) {1, n + 1, 0};
        for ( int i = 1 ; i <= n ; i++ ) {
            if ( i >= stk[bot].R ) bot++;
            dp[i] = f(i, stk[bot].pos);
            update(i);
//            cout << (ll) f(i, stk[bot].pos) << endl;
        }
        if ( dp[n] > 1e18 ) {
            cout << "Too hard to arrange" << endl;
        } else {
            vector<PI> as;
            cout << (ll)dp[n] << endl;
        }
    }
    return 0;
}
```

8.3 Conditon

8.3.1 Totally Monotone (Concave/Convex)

$\forall i < i', j < j',\ B[i][j] \le B[i'][j] \implies B[i][j'] \le B[i'][j']$
$\forall i < i', j < j',\ B[i][j] \ge B[i'][j] \implies B[i][j'] \ge B[i'][j']$

8.3.2 Monge Condition (Concave/Convex)

$\forall i < i', j < j',\ B[i][j] + B[i'][j'] \ge B[i][j'] + B[i'][j]$
$\forall i < i', j < j',\ B[i][j] + B[i'][j'] \le B[i][j'] + B[i'][j]$

8.3.3 Optimal Split Point

If

$$B[i][j] + B[i+1][j+1] \ge B[i][j+1] + B[i+1][j]$$

then

$$H_{i,j-1} \le H_{i,j} \le H_{i+1,j}$$

8.4 Dynamic Convex Hull

```
struct Line {
  mutable int64_t a, b, p;
  bool operator<(const Line &rhs) const { return a < rhs.a; }
  bool operator<(int64_t x) const { return p < x; }
};
struct DynamicHull : multiset<Line, less<>> {
  static const int64_t kInf = 1e18;
  int64_t Div(int64_t a, int64_t b) { return a / b - ((a ^ b) <
       0 && a % b); }
  bool Isect(iterator x, iterator y) {
    if (y == end()) { x->p = kInf; return false; }
    if (x->a == y->a) x->p = x->b > y->b ? kInf : -kInf;
    else x->p = Div(y->b - x->b, x->a - y->a);
    return x->p >= y->p;
  }
  void Insert(int64_t a, int64_t b) {
    auto z = insert({a, b, 0}), y = z++, x = y;
    while (Isect(y, z)) z = erase(z);
    if (x != begin() && Isect(--x, y)) Isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) Isect(x,
         erase(y));
  }
  int64_t Query(int64_t x) {
    auto l = *lower_bound(x);
    return l.a * x + l.b;
  }
};
```

8.5 $1D/1D$ Convex Optimization

```
struct segment {
  int i, l, r;
  segment(int a, int b, int c): i(a), l(b), r(c) {}
};
inline long long f(int l, int r) { return dp[l] + w(l + 1, r);
     }
void solve() {
  dp[0] = 0ll;
  deque<segment> deq; deq.push_back(segment(0, 1, n));
  for (int i = 1; i <= n; ++i) {
    dp[i] = f(deq.front().i, i);
    while (deq.size() && deq.front().r < i + 1) deq.pop_front()
         ;
    deq.front().l = i + 1;
    segment seg = segment(i, i + 1, n);
    while (deq.size() && f(i, deq.back().l) < f(deq.back().i,
         deq.back().l)) deq.pop_back();
    if (deq.size()) {
      int d = 1048576, c = deq.back().l;
      while (d >>= 1) if (c + d <= deq.back().r) {
        if (f(i, c + d) > f(deq.back().i, c + d)) c += d;
      }
      deq.back().r = c; seg.l = c + 1;
    }
    if (seg.l <= n) deq.push_back(seg);
  }
}
```

8.6 Convex Monotone

```
struct line {
    double a, b;
    inline double operator()(const double &x) const { return a
         * x + b; }
    inline bool checkfront(const line &l, const double &x)
         const { return (*this)(x) < l(x); }
    inline double intersect(const line &l) const { return (l.b
         - b) / (a - l.a); }
    inline bool checkback(const line &l, const line &pivot)
         const { return pivot.intersect((*this)) <= pivot.
         intersect(l); }
};

void solve() {
    for (int i = 1; i < maxn; ++i) dp[0][i] = inf;
    for (int i = 1; i <= k; ++i) {
        deque<line> dq; dq.push_back((line){ 0.0, dp[i - 1][0]
             });
        for (int j = 1; j <= n; ++j) {
            while (dq.size() >= 2 && dq[1].checkfront(dq[0],
                 invt[j])) dq.pop_front();
            dp[i][j] = st[j] + dq.front()(invt[j]);
            line nl = (line){ -s[j], dp[i - 1][j] - st[j] + s[j
                 ] * invt[j] };
            while (dq.size() >= 2 && nl.checkback(dq[dq.size()
                 - 1], dq[dq.size() - 2])) dq.pop_back();
            dq.push_back(nl);
        }
    }
}
```

8.7 Convex NonMonotone

```
struct line {
    int m, y;
    int l, r;
    line(int m = 0,int y = 0, int l = -5, int r = 1000000009):
         m(m), y(y), l(l), r(r) {}
    int get(int x) const { return m * x + y; }
    int useful(line le) const {
        return (int)(get(l) >= le.get(l)) + (int)(get(r) >= le.
             get(r));
    }
```

```
};

int magic;
bool operator < (const line &a, const line &b) {
    if (magic) return a.m < b.m;
    return a.l < b.l;
}

set<line> st;

void addline(line l) {
    magic = 1;
    auto it = st.lower_bound(l);
    if (it != st.end() && it->useful(l) == 2) return;
    while (it != st.end() && it->useful(l) == 0) it = st.erase(
        it);
    if (it != st.end() && it->useful(l) == 1) {
        int L = it->l, R = it->r, M;
        while (R > L) {
            M = (L + R + 1) >> 1;
            if (it->get(M) >= l.get(M)) R = M - 1;
            else L = M;
        }
        line cp = *it;
        st.erase(it);
        cp.l = L + 1;
        if (cp.l <= cp.r) st.insert(cp);
        l.r = L;
    }
    else if (it != st.end()) l.r = it->l - 1;
    it = st.lower_bound(l);
    while (it != st.begin() && prev(it)->useful(l) == 0) it =
        st.erase(prev(it));
    if (it != st.begin() && prev(it)->useful(l) == 1) {
        --it;
        int L = it->l, R = it->r, M;
        while (R > L) {
            M = (L + R) >> 1;
            if (it->get(M) >= l.get(M)) L = M + 1;
            else R = M;
        }
        line cp = *it;
        st.erase(it);
        cp.r = L - 1;
        if (cp.l <= cp.r) st.insert(cp);
        l.l = L;
    }
    else if (it != st.begin()) l.l = prev(it)->r + 1;
    if (l.l <= l.r) st.insert(l);
}

int getval(int d) {
    magic = 0;
    return (--st.upper_bound(line(0, 0, d, 0)))->get(d);
}
```

8.8 Sum over Subsets Optimization

```
// O(N * 2^N)
for (int mask = 0; mask < (1 << N); mask++)
    F[mask] = A[mask];

for (int i = 0; i < N; i++)
    for (int mask = 0; mask < (1 << N); mask++)
        if (mask & (1 << i))
            F[mask] += F[mask ^ (1 << i)];
```