সমাহিত (BUET)

February 2020

halfblindprince, solaimanope, upobir

1 Aho Corasick

```
const int sigma = 26;   ///Alphabet Size
struct Vertex {
  int next[sigma];     /// indices of child node
  bool leaf = false;   /// if it is a last char
  int p = -1;          /// index of parent node
  char pch;            /// parent character
  int link = -1;       /// suffix link for vertex
  int go[sigma];       /// save the values of go
  Vertex(int p=-1, char ch='$'): p(p), pch(ch) {
    fill(next, next+sigma, -1);
    fill(go, go+sigma, -1);
  }
};
vector<Vertex> t(1);   ///Warning: multiple TCase
void add_string(string const& s) {
  int v = 0;
  for (char ch : s) {
    int c = ch - 'a';
    if (t[v].next[c] == -1) {
      t[v].next[c] = t.size();
      t.emplace_back(v, ch);
    }
    v = t[v].next[c];
  }
  t[v].leaf = true;
}
int go(int v, char ch);
int get_link(int v) {
  if (t[v].link == -1) {
    if (v == 0 || t[v].p == 0) t[v].link = 0;
    else t[v].link=go(get_link(t[v].p),t[v].pch);
  }
  return t[v].link;
}
///returns node to go from node 'v' with edge 'ch'
int go(int v, char ch) {
  int c = ch - 'a';
  if (t[v].go[c] == -1) {
    if (t[v].next[c] != -1)
      t[v].go[c] = t[v].next[c];
    else t[v].go[c] = v == 0 ? 0 :
                          go(get_link(v), ch);
  }
  return t[v].go[c];
}
```

2 Aliens' Trick

You are given an array v of integers (possibly negative) of length $N(\leq 10^5)$ and a number $K(\leq N)$. Select at most K disjoint subarrays of the initial sequence such that the sum of the elements included in the subarrays is maximized.

The trick behind the "aliens optimization" is that we can add a cost (penalty) which we will denote by λ for each taken subarray. If $\lambda = 0$ the solution would be taking a subarray for each positive element, but by increasing the value of λ, the optimum solution shifts to taking fewer subarrays. Now we just have to find a λ that allows us to take as many subarrays as possible, but still fewer than K.

To do a small recap, λ is the cost we assign to adding a new subarray, and increasing λ will decrease the number of subarrays in an optimal solution or keep it the same, but never increase it. That suggests that we could just binary search the smallest value of λ that yields an optimal solution with less than K elements.

$$dp_\lambda[n] = \max\{dp_\lambda[n-1], \max_{i=1}^{n-1}\{(\sum_{k=i}^{n} v_k) + dp_\lambda[i-1] - \lambda\}\}$$

Besides just the dp, we will store another auxiliary array: $cnt_\lambda[n]$ = ["How many subarrays does $dp_\lambda[n]$ employ in its solution"]

These recurrences are implementable in $O(N)$. The pseudocode of the solution would look like this:

```
minbound = 0, maxbound = 1e18
while maxbound - minbound > 1e-6:
    lambda = (maxbound - minbound) / 2
    #compute dp and aux for lambda
    if cnt[n] <= k:
        minbound = lambda
    else:
        maxbound = lambda
#compute dp and cnt for the final lambda
return dp[n] + cnt[n] * lambda #if there are less
↪  than k positive values, then cnt[n] < k
```

Let's denote by $ans[k]$ the answer for the problem, but using exactly k subarrays. The key observation in proving that our solving method is correct is that the $ans[k]$ sequence is concave, that is $ans[k] - ans[k-1] \leq ans[k-1] - ans[k-2]$. A more natural way of thinking about this and the actual way most people "feel" the concavity/convexity is by interpreting it as *if I have k subarrays and add another one, it will help me more than if I had $k+1$ subarrays and added one more.*

3 Articulation

```
const int N = 100000+7, root = 1;
int d[N];// discovery time of u
int low[N];// min backedge discover in subtr of u
vector<int> g[N];// graph
bool art_pt[N];// articulation points
// vector< pair<int, int> > art_briz;  // bridges
bool vis[N]; int tt = 0;  // initialize
void articulate(int u, int p) {
  tt++; low[u] = d[u] = tt; vis[u] = true;
  int noc = 0;
  for(int v : g[u]) if(v != p) {
    if(vis[v]) low[u] = min(low[u], d[v]);
    else {
      // par[v] = u
      articulate(v, u);
      low[u] = min(low[u], low[v]);
      if(d[u] <= low[v] and u != root)
        art_pt[u] = true;
      // if(d[u] < low[v])
      //  art_briz.push_back(make_pair(u, v));
      noc++;
    }
  }
  if(u == root and noc > 1) art_pt[u] = true;
}
```

4 Berlekamp Massey

```
const LL MOD = 1000000007;///MOD must be prime > 2
#define MOD_THRESHOLD 18
///MOD * MOD * MOD_THRESHOLD < 2^64
namespace bbl{ /// Black box linear algebra
```

```
int mod_inverse(int x){
  int u = 1, v = 0, t = 0, a = x, b = MOD;
  while (b) {
    t = a / b; a -= (t * b), u -= (t * v);
    swap(a, b), swap(u, v);
  }
  return (u + MOD) % MOD;
}
int convolution(const int*A,const int*B,int n){
  int i = 0, j = 0; ULL res = 0;
  for (i = 0; (i+MOD_THRESHOLD) <= n; res%=MOD)
    for (j = 0; j < MOD_THRESHOLD; j++, i++)
      res += (ULL)A[i] * B[i];
  for (j = 0; i < n; i++)
    res += (ULL)A[i] * B[i];
  return res % MOD;
} /// Berlekamp Massey algorithm in O(n ^ 2)
/// Finds the shortest linear recurrence that
/// will generate the sequence S & returns in C
/// S[i]*C[l-1]+S[i+1]*C[l-2]+...+S[j]*C[0] = 0
int berlekampMassey(vector<int>S,vector<int>&C){
  assert((S.size() % 2) == 0); int n = S.size();
  C.assign(n+1, 0); vector <int>T, B(n + 1, 0);
  reverse(S.begin(), S.end());
  C[0] = 1, B[0] = 1;
  int i, j, k, d, x, l=0, m=1, b=1, deg=0;
  for (i = 0; i < n; i++){
    d = S[n-i-1];
    if (l>0)
      d = (d+convolution(&C[1],&S[n-i],l))%MOD;
    if (d == 0) m++;
    else{
      if (l*2 <= i)
        T.assign(C.begin(), C.begin() + l + 1);
      x = (LL)mod_inverse(b)*(MOD - d);
      x = (x%MOD+MOD)%MOD;
      for (j = 0; j <= deg; j++)
        C[m+j] = (C[m+j]+(ULL)x*B[j]) % MOD;
      if (l*2 <= i){
        B.swap(T); deg = B.size() - 1;
        b = d, m = 1, l = i - l + 1;
      }
      else m++;
    }
  }
  C.resize(l + 1);
  return l;
} ///S{1, 1, 2, 3, 5, 8}, Call with(S, C)
} ///generates C = {1, 1000000006, 1000000006}
```

5 Bipartite Matching: Kuhn

```
const int MAXN = 5000 + 7; bool vis[MAXN];
//left pair of right node,right pair of left node
int lpair[MAXN], rpair[MAXN];
// adj[left_side_node].push_back(right_side_node)
vector<int> adj[MAXN];
bool bpm(int u) {
  for(int i=0; i<(int) adj[u].size(); ++i) {
    int v = adj[u][i]; if(vis[v]) continue;
    vis[v] = true;
    if(lpair[v] == -1 or bpm(lpair[v])) {
      lpair[v] = u; rpair[u] = v;
      return true;
    }
  }
  return false;
} // n is the number of nodes on the left side
int max_matching(int n) {
  memset(lpair, -1, sizeof lpair);
  memset(rpair, -1, sizeof rpair);
  int ret = 0;
  for(int i=1; i<=n; ++i) {
    memset(vis, false, sizeof vis);
    ret += bpm(i);
  }
  return ret;
} // O(VE), Practically fast
```

6 Binary Indexed Tree

```
ll tr[N]; int upto = N-1;
void update(int p, ll x) {
  while(p <= upto) { tr[p] += x; p += (p & -p); }
}  // adds x to point p
ll query(int p) {
  ll sum = 0;
  while(p > 0) { sum += tr[p]; p -= (p & -p); }
  return sum;
}  // returns sum of 1...p
```

7 Centroid Decomposition

```
const int MAXN = 100003;
vector<int>edg[MAXN];
int done[MAXN], sbtr[MAXN];
//auxiliary function for CD
void solveForCentroid(int centroid) {
  cout << "centroid " << centroid << endl;
}
int subtreeSize(int u, int p) {
  sbtr[u] = 1;
  for (int v : edg[u]) {
    if (v==p || done[v]) continue;
    sbtr[u] += subtreeSize(v, u);
  }
  return sbtr[u];
}
int nextCentroid(int n, int u, int p) {
  for (int v : edg[u]) {
    if (v==p||done[v]) continue;
    if (sbtr[v]+sbtr[v] > n)
      return nextCentroid(n, v, u);
  }
  return u;
}
void decompose(int u, int pcent) {
  int n = subtreeSize(u, pcent);
  int centroid = nextCentroid(n, u, pcent);
  solveForCentroid(centroid);
  done[centroid] = 1;
  for (int v : edg[centroid]) {
    if (done[v]) continue;
    decompose(v, centroid);
  }
}
void start(int n) {
  for (int i = 1; i <= n; i++) done[i] = 0;
  decompose(1, 0);
}
// Call start(treesize)
```

8 Convex Hull

```
#define REMOVE_REDUNDANT
typedef double T;
const T EPS = 1e-7;
struct PT {
  T x, y;
  PT() {}
  PT(T x, T y) : x(x), y(y) {}
  bool operator<(const PT &rhs) const {
    return make_pair(y,x)<make_pair(rhs.y,rhs.x);
  }
```

```
  bool operator==(const PT &rhs) const {
    return make_pair(y,x)==make_pair(rhs.y,rhs.x);
  }
};
T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) {
  return cross(a,b) + cross(b,c) + cross(c,a);
}
#ifdef REMOVE_REDUNDANT
bool between(const PT &a,const PT &b,const PT &c){
  return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*
    (c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif
void ConvexHull(vector<PT> &pts) {
  sort(pts.begin(), pts.end());
  pts.erase(unique(pts.begin(), pts.end()),
    pts.end());
  vector<PT> up, dn;
  for (int i = 0; i < pts.size(); i++) {
    while (up.size() > 1 && area2(up[up.size()-2],
      up.back(), pts[i]) >= 0) up.pop_back();
    while (dn.size() > 1 && area2(dn[dn.size()-2],
      dn.back(), pts[i]) <= 0) dn.pop_back();
    up.push_back(pts[i]); dn.push_back(pts[i]);
  }
  pts = dn;
  for (int i = (int) up.size() - 2; i >= 1; i--)
    pts.push_back(up[i]);
#ifdef REMOVE_REDUNDANT
  if (pts.size() <= 2) return;
  dn.clear();
  dn.push_back(pts[0]);
  dn.push_back(pts[1]);
  for (int i = 2; i < pts.size(); i++) {
    if (between(dn[dn.size()-2], dn[dn.size()-1],
      pts[i])) dn.pop_back();
    dn.push_back(pts[i]);
  }
  if (dn.size() >= 3 &&
    between(dn.back(), dn[0], dn[1])) {
    dn[0] = dn.back(); dn.pop_back();
  }
  pts = dn;
#endif
}
```

9 Convex Hull Trick: Dynamic

```
// Given a set of lines in y = mx + c formatwith a
// bunch of queries to find the min y for given x
// dynamically adds line and queries in O(lg n)
struct line {
  long long m, c;
  double xleft;
  bool type;
  line(long long _m, long long _c) {
    m = _m; c = _c; type = 0;
  }
  bool operator < (const line &other) const {
    if(other.type) return xleft < other.xleft;
    return m > other.m;
  }
};
double meet(line x, line y) {
  return 1.0 * (y.c - x.c) / (x.m - y.m);
}
struct dynamic_cht { // stores the lower hull
  set < line > hull;
  typedef set < line > :: iterator iter;
  dynamic_cht() { hull.clear(); }
  bool hasleft(iter node) {
    return node != hull.begin();
  }
  bool hasright(iter node) {
    return node != prev(hull.end());
  }
  void updateborder(iter node) {
    if(hasright(node)) {
      line temp = *next(node);
      hull.erase(temp);
      temp.xleft = meet(*node, temp);
      hull.insert(temp);
    }
    if(hasleft(node)) {
      line temp = *node;
      temp.xleft = meet(*prev(node), temp);
      hull.erase(node);
      hull.insert(temp);
    }
    else {
      line temp = *node;
      hull.erase(node);
      temp.xleft = -1e18;
      hull.insert(temp);
    }
  }
  bool useless(line left,line middle,line right) {
    return meet(left, middle)>meet(middle, right);
  }
  bool useless(iter node) {
    if(hasleft(node) && hasright(node)) {
      return
        useless(*prev(node), *node, *next(node));
    }
    return 0;
  }
  void addline(long long m, long long c) {
    line temp = line(m, c);
    auto it = hull.lower_bound(temp);
    if(it != hull.end() && it -> m == m) {
      if(it -> c > c) hull.erase(it);
      else return;
    }
    hull.insert(temp);
    it = hull.find(temp);
    if(useless(it)) {
      hull.erase(it);
      return;
    }
    while(hasleft(it) && useless(prev(it)))
      hull.erase(prev(it));
    while(hasright(it) && useless(next(it)))
      hull.erase(next(it));
    updateborder(it);
  }
  long long query(long long x) {
    if(hull.empty()) return 1e18;
    line qline(0, 0);
    qline.xleft = x;
    qline.type = 1;
    auto it = hull.lower_bound(qline);
    it = prev(it);
    return it -> m * x + it -> c;
  }
};
```

10 Convex Hull Trick: Special

For lines with non-increasing slope, a bunch of sorted queries can be answered in constant time on the fly.

$$dp(i) = \min_{j<i}\{dp(j) + b(j) \cdot a(i)\}$$

Condition: $b(j) \geq b(j+1)$ and $a(i) \leq a(i+1)$
Complexity: $O(n)$

$$dp(i,j) = \min_{k<j}\{dp(i-1,k) + b(k)\cdot a(j)\}$$

Condition: $b(k) \geq b(k+1)$ and $a(j) \leq a(j+1)$
Complexity: $O(kn)$

```
// Given lines y = mx + c, finds min y for given x
// calc for minimum. for max, alter signs in //..Z
const ll inf = 10000000000000000LL;
struct line {
  ll m, c;
  line(ll _m, ll _c) : m(_m), c(_c) { }
  ll get(ll x) const { return m * x + c; }
  bool operator < (const line &p) const {
    return m < p.m;
  }
};
struct CHT {
  vector<line> hull;
  int cur;

  CHT() : cur(0) { }
  bool isBad(line &p, line &q, line &r) {
    return (p.c-r.c) * 1.0 / (r.m-p.m)
      <= (p.c-q.c) * 1.0 /  (q.m-p.m);
    // (p.c-q.c) * (r.m-p.m)
      // >= (p.c-r.c) * (q.m-p.m) if no overflow
  }
  void addLine(ll m, ll c) {
    if(!hull.empty() and hull.back().m == m) {
      if(hull.back().c > c) hull.pop_back();//..Z
      else return ;
    }
    hull.push_back(line(m, c));
    int sz = hull.size();
    while(sz > 2 and
      isBad(hull[sz-3], hull[sz-2], hull[sz-1])) {
      swap(hull[sz-2], hull[sz-1]);
      hull.pop_back();
      --sz;
    }
  }
  ll query(ll x) {
    if(hull.empty()) return -inf;
    int l = -1, r = hull.size() - 1;
    while(r-l > 1) {
      int m = (l + r) / 2;
      if(hull[m].get(x) >= hull[m+1].get(x))
        l = m;  //..Z
      else r = m;
    }
    return hull[r].get(x);
  }
  // the query x's have to be non-decreasing
  ll Cquery(ll x) {
    if(hull.empty()) return -inf;
    while(cur+1 < hull.size()) {
      if(hull[cur].get(x) > hull[cur+1].get(x))
        ++cur;  //..Z
      else break;
    }
    return hull[cur].get(x);
  }
  void clear() { hull.clear(); }
};
```

11 Discrete Log

```
unordered_map<ll, int> buck;//buckets[a^(ps)]=p
vector<ll> apow;//apow[i] = a^i;
ll s, m;//to save for future queries
void DLogPrecal(ll a, ll _m, int q){
  m = _m, s = ll(sqrtl(m/(q+1))+1;
  ll s2 = m/s+1; apow.resize(s+1); apow[0] = 1;
  for(int i = 1; i<=s; i++)apow[i]=apow[i-1]*a%m;
  ll cur = apow[s]; buck.clear();
  for(int p = 1; p<=s2; p++, cur=cur*apow[s]%m)
    if(buck.find(cur) == buck.end()) buck[cur]=p;
}
ll DLog(ll b){
  if(b == 1) return 0;
  for(ll q = s-1; q>=0; q--){
    ll tmp = b*apow[q]%mod;
    if(buck.find(tmp) != buck.end())
      return buck[tmp]*s - q;}
    return -1;
}
```

12 Divide & Conquer Optimization

$$dp(i,j) = \min_{k\leq j}\{dp(i-1,k) + C(,j)\}$$

Let $opt(i,j)$ be the value of k that minimizes the above expression. If $opt(i,j) \leq opt(i,j+1)$, we can optimize to $O(nm\log n)$ from $O(nm^2)$ where $1 \leq i \leq n$ and $1 \leq j \leq m$.

```
int n;
long long C(int i, int j);
vector<long long> dp_before(n), dp_cur(n);
// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
  if (l > r) return;
  int mid = (l + r) >> 1;
  pair<long long, int> best = {INF, -1};
  for (int k = optl; k <= min(mid, optr); k++)
    best = min(best, {dp_before[k]+C(k, mid), k});
  dp_cur[mid] = best.first;
  int opt = best.second;
  compute(l, mid - 1, optl, opt);
  compute(mid + 1, r, opt, optr);
}
```

13 Directed MST

```
// O(n*m)| Constructs a rooted tree of minimum
// total weight from root node| returns -1 if no
// solution exists| 0-indexed
const int inf = 0x3f3f3f3f;
struct edge {
  int u, v, w;
  edge() {}
  edge(int a,int b,int c) : u(a), v(b), w(c) {}
  bool operator < (const edge& o) const {
    if (u == o.u) {
      if (v == o.v) return w < o.w;
      return v < o.v;
    }
    return u < o.u;
  }
};
int dmst(vector<edge> &edges, int root, int n) {
  int ans = 0; int cur_nodes = n;
  while (true) {
    vector<int> lo(cur_nodes, inf),
      pi(cur_nodes, inf);
    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v,
        w = edges[i].w;
      if (w < lo[v] and u != v) {
        lo[v] = w; pi[v] = u;
      }
    }
    lo[root] = 0;
    for (int i = 0; i < lo.size(); ++i) {
      if (i == root) continue;
```

```
      if (lo[i] == inf) return -1;
    }
    int cur_id = 0;
    vector<int> id(cur_nodes, -1),
      mark(cur_nodes, -1);
    for (int i = 0; i < cur_nodes; ++i) {
      ans += lo[i];
      int u = i;
      while(u!=root and id[u]<0 and mark[u]!=i) {
        mark[u] = i; u = pi[u];
      }
      if (u != root and id[u] < 0) { // Cycle
        for (int v = pi[u]; v != u; v = pi[v])
          id[v] = cur_id;
        id[u] = cur_id++;
      }
    }
    if (cur_id == 0) break;
    for (int i = 0; i < cur_nodes; ++i)
      if (id[i] < 0) id[i] = cur_id++;
    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v,
        w = edges[i].w;
      edges[i].u = id[u]; edges[i].v = id[v];
      if (id[u] != id[v]) edges[i].w -= lo[v];
    }
    cur_nodes = cur_id; root = id[root];
  }
  return ans;
}
```

14 Dominator Tree

```
//Constructs a tree where all ancestors of x are
//its domintators| 1-indexed tree and consists of
//only the reachable nodes from src| for an (u, v)
//edge in the tree, (v, u) exists| Blackbox:
//Constructor w no of nodes | add(u, v) | build(s)
//Complexity: O((V+E) lg V)
typedef vector<int> VI;
typedef vector<VI> VVI;
struct ChudirBhai {
  int n, T; VVI g, tree, rg, bucket;
  VI sdom, par, dom, dsu, label, arr, rev;
  ChudirBhai(int n):
  n(n), g(n+1), tree(n+1), rg(n+1), bucket(n+1),
  sdom(n+1), par(n+1), dom(n+1), dsu(n+1),
  label(n+1), arr(n+1), rev(n+1), T(0) {
    for(int i = 1; i <= n; i++)
      sdom[i] = dom[i] = dsu[i] = label[i] = i;
  }
  void addEdge(int u, int v) { g[u].push_back(v);}
  void dfs0(int u) {
    T++; arr[u] = T, rev[T] = u;
    label[T] = T, sdom[T] = T, dsu[T] = T;
    for(int w : g[u]) {
      if(!arr[w]) dfs0(w), par[arr[w]] = arr[u];
      rg[arr[w]].push_back(arr[u]);
    }
  }
  int Find(int u, int x = 0) {
    if(u == dsu[u]) return x ? -1 : u;
    int v = Find(dsu[u], x+1);
    if(v < 0) return u;
    if(sdom[label[dsu[u]]] < sdom[label[u]])
      label[u] = label[dsu[u]];
    dsu[u] = v;
    return x ? v : label[u];
  }
  void Union(int u, int v) { dsu[v] = u; }
  VVI build(int s) {
    dfs0(s);
    for(int i = n; i >= 1; i--) {
      for(int qq : rg[i])
        sdom[i] = min(sdom[i], sdom[Find(qq)]);
      if(i > 1) bucket[sdom[i]].push_back(i);
      for(int w : bucket[i]) {
        int v = Find(w);
        if(sdom[v] == sdom[w]) dom[w] = sdom[w];
        else dom[w] = v;
      }
      if(i > 1) Union(par[i], i);
    }
    for(int i = 2; i <= n; i++) {
      if(dom[i] != sdom[i]) dom[i] = dom[dom[i]];
      tree[rev[i]].push_back(rev[dom[i]]);
      tree[rev[dom[i]]].push_back(rev[i]);
    }
    return tree;
  }
};
```

15 Euler Path, Circuit

```
struct Edge; typedef list<Edge>::iterator iter;
struct Edge {
  int next_vert; iter rev_edg;
  Edge(int next_vert) : next_vert(next_vert) { }
};
const int N = 1000000+7;
list<Edge> adj[N]; stack<int> path;
void euler_path(int v) {
  while(adj[v].size() > 0) {
    int vn = adj[v].front().next_vert;
    adj[vn].erase(adj[v].front().rev_edg);
    adj[v].pop_front(); euler_path(vn);
  }
  path.push(v);
}
void add_edge(int a, int b) {
  adj[a].push_front(Edge(b));
  iter ita = adj[a].begin();
  adj[b].push_front(Edge(a));
  iter itb = adj[b].begin();
  ita->rev_edg = itb; itb->rev_edg = ita;
} // ********** For undirected G O(V+E) ********
const int N = 26; vector<int> adj[N];
vector<int> find_path_directed(int src=1) {
  vector<int> ecnt(N, 0), ret;
  stack<int> st; st.push(src);
  while(!st.empty()) {
    int u = st.top(); st.pop();
    while(ecnt[u] < (int) adj[u].size()) {
      st.push(u); u = adj[u][ecnt[u]++];
    }
    ret.push_back(u);
  }
  reverse(ret.begin(), ret.end()); return ret;
} // ********** For directed G *****************
```

16 Extended GCD

```
ll egcd(ll a, ll b, ll &x, ll &y){
    if(a == 0) {x = 0, y = 1; return b;}
    ll x1, y1, d = egcd(b % a, a, x1, y1);
    x = y1 - (b/a)*x1;   y = x1;
    return d;
}
```

17 Fast Fourier Transform

```
/** Possible Optimizations: 1. Writing a custom
complex class reduces runtime. 2. If there are
multiple testcases of similar size, it might be
```

```
faster to precalculate the roots of unity.**/
typedef complex<double> CD; typedef long long LL;
const double PI = acos(-1);
struct FFT {
  int N; vector<int> perm;
  void precalculate() {
    perm.resize(N); perm[0] = 0;
    for (int k=1; k<N; k<<=1) {
      for (int i=0; i<k; i++) {
        perm[i] <<= 1; perm[i+k] = 1 + perm[i];
      }
    }
  }
  void fft(vector<CD> &v, bool invert = false) {
    if (v.size() != perm.size()) {
      N = v.size();
      assert(N && (N&(N-1)) == 0);
      precalculate();
    }
    for (int i=0; i<N; i++)
      if (i < perm[i]) swap(v[i], v[perm[i]]);
    for (int len = 2; len <= N; len <<= 1) {
      double angle = 2 * PI / len;
      if (invert) angle = -angle;
      CD factor = polar(1.0, angle);
      for (int i=0; i<N; i+=len) {
        CD w(1);
        for (int j=0; j<len/2; j++) {
          CD x = v[i+j], y = w * v[i+j+len/2];
          v[i+j] = x+y; v[i+j+len/2] = x-y;
          w *= factor;
        }
      }
    }
    if (invert) for (CD &x : v) x/=N;
  }
  vector<LL> multiply(vector<LL> a, vector<LL> b){
    vector<CD>fa(a.begin(), a.end());
    vector<CD>fb(b.begin(), b.end());
    int n=1; while (n < a.size()+ b.size()) n<<=1;
    fa.resize(n); fb.resize(n);
    fft(fa); fft(fb);
    for (int i=0; i<n; i++) fa[i] *= fb[i];
    fft(fa, true); vector<LL>ans(n);
    for (int i=0; i<n; i++)
      ans[i]=round(fa[i].real());
    return ans;
  }
};
```

18 Fast Fourier Transform: 2D

```
struct CD { ///Custom complex class
  double x, y;
  CD(double x = 0, double y = 0) : x(x), y(y) { }
  CD operator+(const CD &p) const {
    return CD(x+p.x, y+p.y);
  } CD operator - (const CD &p) const {
    return CD(x-p.x, y-p.y);
  } CD operator * (const CD &p) const {
    return CD(x*p.x-y*p.y, x*p.y+y*p.x);
  } CD operator / (double d) const {
    return CD(x/d, y/d);
  }
};
vector< VI >two2Dgun(vector<VI>a, vector<VI>b) {
  vector< vector< CD > >fa, fb; int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  a.resize(n); b.resize(n);     int m = 1;
  while (m < a[0].size() + b[0].size()) m <<= 1;
  for (int i = 0; i < n; i++) {
    a[i].resize(m); b[i].resize(m);
    fa.emplace_back(a[i].begin(), a[i].end());
    fb.emplace_back(b[i].begin(), b[i].end());
  }
  for (int i = 0; i < n; i++) {
    fft.fft(fa[i]);
    fft.fft(fb[i]);
  }
  vector<CD>tmp(n);
  for (int j = 0; j < m; j++) {
    for (int i = 0; i < n; i++) tmp[i] = fa[i][j];
    fft.fft(tmp);
    for (int i = 0; i < n; i++) fa[i][j] = tmp[i];
    for (int i = 0; i < n; i++) tmp[i] = fb[i][j];
    fft.fft(tmp);
    for (int i = 0; i < n; i++) fb[i][j] = tmp[i];
  }
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      fa[i][j] = fa[i][j]*fb[i][j];
  for (int j = 0; j < m; j++) {
    for (int i = 0; i < n; i++) tmp[i] = fa[i][j];
    fft.fft(tmp, true);
    for (int i = 0; i < n; i++) fa[i][j] = tmp[i];
  }
  for (int i = 0; i < n; i++) {
    fft.fft(fa[i], true);
    for (int j = 0; j < m; j++) {
      a[i][j] = round(fa[i][j].x);
    }
  }
  return a;
}
```

19 Fast IO

```
inline int readChar();
template <class T = int> inline T readInt();
static const int buf_size = 4096;
inline int getChar() {
  static char buf[buf_size];
  static int len = 0, pos = 0;
  if (pos == len)
    pos = 0, len = fread(buf, 1, buf_size, stdin);
  if (pos == len) return -1;
  return buf[pos++];
}
inline int readChar() {
  int c = getChar();
  while (c <= 32) c = getChar();
  return c;
}
template <class T> inline T readInt() {
  int s = 1, c = readChar();
  T x = 0;
  if (c == '-') s = -1, c = getChar();
  while ('0' <= c && c <= '9')
    x = x * 10 + c - '0', c = getChar();
  return s == 1 ? x : -x;
}
```

20 Fast Walsh Hadamard Transform

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
#define bitwiseXOR 1
///#define bitwiseAND 2
///#define bitwiseOR 3
void FWHT(vector< LL >&p, bool inverse) {
  int n = p.size(); assert((n&(n-1))==0);
  for (int len = 1; 2*len <= n; len <<= 1) {
    for (int i = 0; i < n; i += len+len) {
      for (int j = 0; j < len; j++) {
```

```
        LL u = p[i+j]; LL v = p[i+len+j];
        #ifdef bitwiseXOR
        p[i+j] = u+v; p[i+len+j] = u-v;
        #endif // bitwiseXOR
        #ifdef bitwiseAND
        if (!inverse) {
          p[i+j] = v; p[i+len+j] = u+v;
        } else {
          p[i+j] = -u+v; p[i+len+j] = u;
        }
        #endif // bitwiseAND
        #ifdef bitwiseOR
        if (!inverse) {
          p[i+j] = u+v; p[i+len+j] = u;
        } else {
          p[i+j] = v; p[i+len+j] = u-v;
        }
        #endif // bitwiseOR
      }
    }
  }
  #ifdef bitwiseXOR
  if (inverse) {
    for (int i = 0; i < n; i++) {
      assert(p[i]%n==0); p[i] /= n;
      ///divide by mmi(n) when working with mod
    }
  }
  #endif // bitwiseXOR
}
/** 140, 132, 108, 100 */
int main() {
  vector< LL >p{2, 4, 6, 8}; FWHT(p, false);
  vector< LL >q{3, 5, 7, 9}; FWHT(q, false);
  vector<LL>r(4);
  for(int i = 0; i < 4; i++) r[i]=p[i]*q[i];
  FWHT(r, true);
  for (int i = 0; i < 4; i++) {
    cout<<"r["<<bitset<2>(i)<<"] = "<<r[i]<<endl;
  }
  return 0;
}
```

21 Gaussian

```
double sys[nmax][nmax], b[nmax];
int Gaussian(int m, int n){//m = eq, n = var
  //double det = 1;
  int r = 0;
  for(int c = 0; c<n; c++){
    int p = r;
    for(int i = r+1; i<m; i++)
      if(fabs(sys[i][c]) > fabs(sys[p][c])) p=i;
    if(eq(sys[p][c], 0) continue;
    if(p != r){
      //det = -det;
      for(int j = 0; j<n; j++)
        swap(sys[p][j], sys[r][j]);
      swap(b[p], b[r]);
    }
    double scale = sys[r][c];
    for(int j = 0; j<n; j++)sys[r][j] /= scale;
    b[r] /= scale; //det *= scale;
    for(int i = 0; i<m; i++){
      if(i == r || eq(sys[i][c],0)) continue;
      scale = sys[i][c];
      for(int j = c; j<n; j++)
        sys[i][j] -= scale*sys[r][j];
      b[i] -= scale*b[r];
    }
    r++;
  }
  //if(r != n) det = 0;
  return r; //return det;
}
```

22 Geometry: 2D

```
const double INF = 1e100, EPS = 1e-12;
struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const {
    return PT(x+p.x, y+p.y);
  } PT operator - (const PT &p)  const {
    return PT(x-p.x, y-p.y);
  } PT operator * (double c) const {
    return PT(x*c,   y*c  );
  } PT operator / (double c) const {
    return PT(x/c,   y/c  );
  }
};
double dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double cross(PT p,PT q){ return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  return os << "(" << p.x << "," << p.y << ")";
}
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t),
    p.x*sin(t)+p.y*cos(t));
}  // rotate a point CCW or CW around the origin
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}  // project point c onto line through a-b | a!=b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if(fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if(r < 0) return a;
  if(r > 1) return b;
  return a + (b-a)*r;
}  // project pt c onto segment through a & b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c,
    ProjectPointSegment(a, b, c)));
}  // distance from c to segment between a & b
double DistancePointPlane(double x, double y,
  double z, double a, double b,double c,double d){
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}  // dist between (x,y,z) and plane ax+by+cz=d
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}  // line a-b, c-d parallel or collinear
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if(LinesCollinear(a, b, c, d)) {
    if(dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS)
      return true;
    if(dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 &&
      dot(c-b, d-b) > 0) return false;
    return true;
  }
  if(cross(d-a, b-a) * cross(c-a, b-a) > 0)
    return false;
  if(cross(a-c, d-c) * cross(b-c, d-c) > 0)
```

```
    return false;
  return true;
}  // segment a-b intersect with c-d
PT ComputeLineIntersection(PT a,PT b,PT c,PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}  // intersect a-b, c-d. unique. not for segments
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2; c=(a+c)/2;
  return ComputeLineIntersection(b,
    b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}  // compute center of circle given three points
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) *
      (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}  // 1/0 for strictly interior/exterior pts.
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if(dist2(ProjectPointSegment(p[i],
      p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}  // Point on boundary of polygon
vector<PT> CircleLineIntersection(PT a, PT b,
  PT c, double r) {
  vector<PT> ret;
  b = b-a; a = a-c;
  double A = dot(b, b), B = dot(a, b);
  double C = dot(a, a) - r*r, D = B*B - A*C;
  if(D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if(D > EPS) ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}  // line a-b, circle (c,r>0)
vector<PT> CircleCircleIntersection(PT a, PT b,
  double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if(d > r+R || d+min(r, R)<max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if(y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}  // circle (a,r), (b,R)
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}
double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*
      (p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}  // centroid: center of gravity/mass
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if(i == l || j == k) continue;
      if(SegmentsIntersect(p[i],p[j],p[k],p[l]))
        return false;
    }
  }
  return true;
}  // polygon (in CW or CCW order) is simple?
```

23 Geometry: 3D

```
// PRE: define PT(x,y,z) with +-*/
double dot(PT p, PT q) {
  return p.x*q.x+p.y*q.y+p.z*q.z;
} double dist2(PT p, PT q) {
  return dot(p-q,p-q);
} PT cross(PT p, PT q) {
  return PT(p.y*q.z-p.z*q.y, p.z*q.x-p.x*q.z,
                             p.x*q.y-p.y*q.x);
} double len2(PT p) { return dot(p, p);}
double len(PT p)  { return sqrt(dot(p, p));}
double triple(PT a, PT b, PT c)  {
  return dot(a, cross(b, c));
} bool isCoplanar(PT a, PT b, PT c, PT d) {
  return abs(triple(b-a, c-a, d-a)) < EPS;
}
PT ComputeCircleCenter(PT a, PT b, PT c) {
  PT x = cross(b-a, c-a);
  assert(len2(x) > EPS);
  PT num1 = cross(x, b-a) * len2(c-a);
  PT num2 = cross(c-a, x) * len2(b-a);
  return a + (num1 + num2)/(len2(x)*2);
} /// distance of Line ab from o
double PointLineDistance(PT a, PT b, PT o) {
  PT ab = b-a; PT oa = a-o;
  return len(cross(ab, oa))/len(ab);
} // dist(a-b, c-d), unstable for nearly parallel
double LineLineDistance(PT a, PT b, PT c, PT d) {
  PT ab = b-a, cd = d-c; PT dir = cross(ab, cd);
  double sz = len2(dir);
  if(sz < EPS) return PointLineDistance(a,b,c);
  dir = dir/len(dir); return abs(dot(dir, a-c));
}
struct Plane {
  PT normal; double d;    ///Ax + By + Cz = D
  Plane(double a, double b, double c, double d)
    : normal(a, b, c), d(d) {}
  Plane(PT n, double d) : normal(n), d(d) {}
}; // plane given by three Non-Collinear Points
Plane getPlane(PT a, PT b, PT c) {
  PT normal = cross(b-a, c-a);
  assert(len2(normal) > EPS);
  double d = dot(normal, a);
  return Plane(normal, d);
} // distance from point a to plane p
double PointPlaneDistance(PT a, Plane p) {
  return abs(dot(p.normal, a)-p.d)/len(p.normal);
}
```

24 Gomory Hu Tree

```
// The Gomory-Hu tree of an undirected graph with
// capacities is a weighted tree that represents
// the minimum s-t cuts for all s-t pairs in the
```

```
// graph. The minimum weighted edge in s-t path in
//  the tree is the s-t cut in original graph.
// Complexity: O(|V| maxflow)
typedef struct { int u, v, w; } edge;
// returns edges of gomory hu tree
// nodes are labeled 1..n here and also in dinic
vector<edge> gomory_hu(int n, vector<edge> &e) {
  vector<edge> edg;
  vector<int> par(n+1, 1);
  for(int i=2; i<=n; ++i) {
    reset();  // dinic
    for(auto &qq : e)
      addedge(qq.u, qq.v, qq.w, qq.w);  // dinic
    S = par[i], T = i;
    edg.push_back( { par[i], i, (int) flow() } );
    for(int j=i+1; j<=n; ++j)
      if(ds[j] >= 0 and par[j] == par[i])
        par[j] = i;
  }
  return edg;
}
```

25 Half Plane Intersection: Largest Circle in Convex Polygon

```
const double EPS = 1e-9, INF = 1e9;
typedef pair< PT, PT > line;
inline double getyval(const line& l, double x) {
  double ret = l.first.y+((l.second.y - l.first.y)
    * (x - l.first.x)) / (l.second.x - l.first.x);
  return ret;
}  // for line l. calcs y val for given x
// returns if any point in half plan intersection
// downPlanes are planes down to specified line,
// basically the lines in the upper hull
// upperPlanes are planes up to specified line,
// basically the lines in lower hull
// [fr, to] is the specified range to find point
bool halfPlaneIntersection(const vector<line>
  &downPlane, const vector<line> &upperPlane,
  double fr = -INF, double to = INF) {  // arg-end
  double lo = fr, hi = to;
  while(hi - lo > EPS) {  // fix a const iter?
    double m1 = lo + (hi - lo) / 3.0;
    double val1 = INF, val2 = -INF;
    for(auto &l : downPlane)
      val1 = min(val1, getyval(l, m1));
    for(auto &l : upperPlane)
      val2 = max(val2, getyval(l, m1));
    double diff1 = val1 - val2;
    double m2 = hi - (hi - lo) / 3.0;
    val1 = INF, val2 = -INF;
    for(auto &l : downPlane)
      val1 = min(val1, getyval(l, m2));
    for(auto &l : upperPlane)
      val2 = max(val2, getyval(l, m2));
    double diff2 = val1 - val2;
    if(max(diff1, diff2) + EPS >= 0) return true;
    if(diff1 > diff2) hi = m2;
    else lo = m1;
  }
  return false;
}  // shifts line c distances to ccw90 direction
vector<line> shiftLines(vector<line>&v, double c){
  vector<line> ret;
  for(auto &l : v) {
    PT vec = l.second - l.first;
    vec = RotateCCW90(vec);
    vec = vec / sqrt(dot(vec, vec)); vec = vec*c;
    ret.push_back(line(l.first+vec,l.second+vec));
  }
  return ret;
} // largest circle in convex polygon, points CCW
double largestFittingCircle(const vector<PT> &v) {
  int n = v.size(), min_idx = 0, max_idx = 0;
  for(int i=0; i<n; ++i) {
    if(v[i].x < v[min_idx].x) min_idx = i;
    if(v[i].x > v[max_idx].x) max_idx = i;
  }
  bool twomins = false, twomaxs = false;
  if(abs(v[(min_idx+1)%n].x-v[min_idx].x)<EPS or
    abs(v[(min_idx-1+n)%n].x-v[min_idx].x)<EPS)
    twomins = true;
  if(abs(v[(max_idx+1)%n].x-v[max_idx].x)<EPS or
    abs(v[(max_idx-1+n)%n].x-v[max_idx].x)<EPS)
    twomaxs = true;
  vector<line> downPlanes;
  for(int i=min_idx; ; i=(i-1+n)%n) {
    if(abs(v[i].x - v[max_idx].x) < EPS) break;
    if(abs(v[i].x - v[(i-1+n)%n].x)<EPS) continue;
    downPlanes.push_back(line(v[(i-1+n)%n],v[i]));
  }
  vector<line> upperPlanes;
  for(int i=min_idx; ; i=(i+1)%n) {
    if(abs(v[i].x - v[max_idx].x) < EPS) break;
    if(abs(v[i].x - v[(i+1)%n].x) < EPS) continue;
    upperPlanes.push_back(line(v[i], v[(i+1)%n]));
  }
  double lo = 0, hi = INF;
  while(hi - lo > EPS) {  // iterate const time
    double mid = (lo + hi) / 2.0;
    auto tDownPlanes=shiftLines(downPlanes,mid);
    auto tUpperPlanes=shiftLines(upperPlanes,mid);
    if(halfPlaneIntersection(tDownPlanes,
      tUpperPlanes, v[min_idx].x + mid * twomins,
      v[max_idx].x - mid * twomaxs))
      lo = mid;
    else hi = mid;
  }
  return lo;
}
```

26 Hashing: String

```
1000000007,1000000009,1000000861,1000099999 <2^30
1088888881,1111211111,1500000001,1481481481 <2^31
2147483647 (2^31-1)  // MOD List ****************
const int N = 1e5 + 7;
const ll P1=1e8+5e4+1, P2=1e9+87, MOD=1e9+7;
ll pow1[N], pow2[N], h1[N], h2[N];
void precal(int n) {
  pow1[0] = pow2[0] = 1;
  for(int i=1; i<n; ++i) {
    pow1[i] = (pow1[i-1] * P1) % MOD;
    pow2[i] = (pow2[i-1] * P2) % MOD;
  }
}
void init(string s) {
  h1[0] = h2[0] = 0;
  for(int i=0; i<(int) s.size(); ++i) {
    h1[i+1] = (h1[i] * P1 + s[i]) % MOD;
    h2[i+1] = (h2[i] * P2 + s[i]) % MOD;
  }
}
inline ll hashval(int l, int r) {
  ll a = (h1[r+1] - h1[l] * pow1[r-l+1]) % MOD;
  if(a < 0) a += MOD;
  ll b = (h2[r+1] - h2[l] * pow2[r-l+1]) % MOD;
  if(b < 0) b += MOD;
  return (a<<32) | b;
}
```

27 Hash Table

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM
      = chrono::steady_clock::now().
        time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};
unordered_map <long long, int, custom_hash > slow;
gp_hash_table <long long, int, custom_hash > fast;
```

28 Heavy Light Decomposition

```
/** flat[] (0-indexed) has the flattened array
flatIdx[] is the reverse map of flat[]. Everything
other than dfs(u, p) & HLD(u, p) are auxiliary */
const int MAXN = 500007; const int LOGN = 20;
vector<int>edg[MAXN];
int sbtr[MAXN], lvl[MAXN], pr[MAXN][LOGN];
int chainIdx[MAXN],chainHead[MAXN],flatIdx[MAXN];
int chainCnt, flatCnt, flat[MAXN];
void dfs(int u, int p) {
  lvl[u] = lvl[p] + 1; pr[u][0] = p;
  for (int k = 1; k < LOGN; k++) {
    pr[u][k] = pr[pr[u][k-1]][k-1];
  }
  sbtr[u] = 1;
  for (int v : edg[u]) {
    if (v==p) continue;
    dfs(v, u); sbtr[u] += sbtr[v];
  }
}
/// auxiliary function
int getLCA(int u, int v) {
  if (lvl[u] < lvl[v]) swap(u, v);
  for (int k = LOGN-1; k >= 0; k--) {
    if (lvl[u]-(1<<k) >= lvl[v]) u = pr[u][k];
  }
  if (u==v) return u;
  for (int k = LOGN-1; k >= 0; k--) {
    if (pr[u][k] != pr[v][k]) {
      u = pr[u][k]; v = pr[v][k];
    }
  }
  return pr[u][0];
}
void HLD(int u, int p) {
  chainIdx[u] = chainCnt; flatIdx[u] = flatCnt;
  flat[flatCnt] = u; flatCnt++;
  int biggie = -1, mx = 0;
  for (int v : edg[u]) {
    if (v==p) continue;
    if (mx < sbtr[v]) {
      mx = sbtr[v]; biggie = v;
    }
  }
  if (biggie==-1) return;
  HLD(biggie, u);
  for (int v : edg[u]) {
    if (v==p||v==biggie) continue;
    chainCnt++; chainHead[chainCnt]=v; HLD(v, u);
  }
}
/// upSeg(l,u,vp) add sgmnts for (l, u] to vp vctr
/// provided l is an ancestor of u
void upSegments(int l, int u, vector<PII>&vp) {
  while (chainIdx[l] != chainIdx[u]) {
    int uhead = chainHead[chainIdx[u]];
    vp.push_back(PII(flatIdx[uhead], flatIdx[u]));
    u = pr[uhead][0];
  }
  if (l!=u) {
    vp.push_back(PII(flatIdx[l]+1, flatIdx[u]));
  }
}
vector<PII>getChainSegments(int u, int v) {
  int l = getLCA(u, v); vector<PII>rt;
  rt.push_back(PII(flatIdx[l], flatIdx[l]));
  if (u==v) return rt;
  upSegments(l, u, rt); upSegments(l, v, rt);
  return rt;
}
PII getSubtreeSegment(int u) {
    return PII(flatIdx[u], flatIdx[u]+sbtr[u]-1);
}
void performHLD(int root) { ///CALL THIS
  dfs(root, 0);  chainCnt = 0;  flatCnt = 0;
  chainHead[0] = root;  HLD(root, 0);
}
```

29 Hungarian

```
// Given n workers, m jobs and worker i receives
// mat[i][j] for job j | MINIMIZE or MAXIMIZE job
// assignment total pay. every job is done.
// Complexity: O(n^3) | pretty faster.
// Hack: keep n <= m. if needed swap sides. fast.
#define MAX 1007
#define MAXIMIZE +1
#define MINIMIZE -1
#define clr(ar) memset(ar, 0, sizeof(ar))
namespace wm{
  bool visited[MAX];
  int U[MAX], V[MAX], P[MAX], way[MAX], minv[MAX],
    match[MAX], ar[MAX][MAX];
  // n = number of row and m = number of columns
  // in 1 based, flag = MAXIMIZE or MINIMIZE
  // match[i] is the col to which row i matches
  int hungarian(int n, int m, int mat[MAX][MAX],
    int flag){
    clr(U), clr(V), clr(P), clr(ar), clr(way);
    for (int i = 1; i <= n; i++){
      for (int j = 1; j <= m; j++){
        ar[i][j] = mat[i][j];
        if(flag == MAXIMIZE) ar[i][j] = -ar[i][j];
      }
    }
    if (n > m) m = n;
    int i, j, a, b, c, d, r, w;
    for (i = 1; i <= n; i++){
      P[0] = i, b = 0;
      for (j = 0; j <= m; j++)
        minv[j] = inf, visited[j] = false;
      do {
        visited[b] = true;
        a = P[b], d = 0, w = inf;
        for (j = 1; j <= m; j++){
          if (!visited[j]){
            r = ar[a][j] - U[a] - V[j];
            if (r < minv[j])
              minv[j] = r, way[j] = b;
            if (minv[j] < w) w = minv[j], d = j;
          }
        }
        for (j = 0; j <= m; j++){
          if (visited[j]) U[P[j]] += w, V[j] -= w;
```

```
        else minv[j] -= w;
      }
      b = d;
    } while (P[b] != 0);
    do {
      d = way[b]; P[b] = P[d], b = d;
    } while (b != 0);
  }
  for (j = 1; j <= m; j++) match[P[j]] = j;
  return (flag == MINIMIZE) ? -V[0] : V[0];
 }
}
```

30 Java IO

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.StringTokenizer;
public class Main {
  public static void main(String[] args) {
    InputStream inputStream = System.in;
    OutputStream outputStream = System.out;
    InputReader in = new InputReader(inputStream);
    PrintWriter out=new PrintWriter(outputStream);
    Task solver = new Task();
    int testCount = in.nextInt();
    for(int i=1; i<=testCount; ++i) {
      solver.solve(i, in, out);
    }
    out.close();
  }
  static class Task {
    public void solve(int testNumber,
      InputReader in, PrintWriter out) {
      // do some magic here
    }
  }
  static class InputReader {
    public BufferedReader reader;
    public StringTokenizer tokenizer;
    public InputReader(InputStream stream) {
      reader = new BufferedReader(
        new InputStreamReader(stream), 32768);
      tokenizer = null;
    }
    public String next() {
      while(tokenizer == null ||
        !tokenizer.hasMoreTokens()) {
        try {
          tokenizer = new
            StringTokenizer(reader.readLine());
        } catch (IOException e) {
          throw new RuntimeException(e);
        }
      }
      return tokenizer.nextToken();
    }
    public int nextInt() {
      return Integer.parseInt(next());
    }
    public Long nextLong() {
      return Long.parseLong(next());
    }
  }
}
```

31 KMP

```
// prefix arr, pi[q] is the len of longest proper
// prefix of P which is proper suff of P[0..q]
vector<int> pi;
void prefix_compute(string P) {
  pi.resize(P.size()); pi[0] = 0;
  int q = 0;  // number of matched characters
  for(int i=1; i<(int) P.size(); ++i) {
    while(q > 0 and P[q] != P[i]) q = pi[q-1];
    if(P[q] == P[i]) ++q;
    pi[i] = q;
  }
}
int kmp_matcher(string T, string P) {
  prefix_compute(P);
  int q = 0;  // no of matched characters
  int ocr = 0;  // no of ocr of P in T
  for(int i=0; i<(int) T.size(); ++i) {
    while(q > 0 and P[q] != T[i]) q = pi[q-1];
    if(P[q] == T[i]) ++q;
    if(q == (int) P.size()) {
      // Pattern matches at (i+1 - P.size())
      ++ocr; q = pi[q-1];
    }
  }
  return ocr;
}
```

32 Li Chao Tree

```
typedef int ftype;
typedef complex<ftype> point;
#define x real
#define y imag
ftype dot(point a, point b) {
  return (conj(a) * b).x();
}
ftype f(point a,  ftype x) {
  return dot(a, {x, 1});
}
const int maxn = 2e5;
point line[4 * maxn];
void add_line(point nw, int v = 1, int l = 0,
                                int r = maxn) {
  int m = (l + r) / 2;
  bool lef = f(nw, l) < f(line[v], l);
  bool mid = f(nw, m) < f(line[v], m);
  if(mid) swap(line[v], nw);
  if(r - l == 1) return ;
  else if(lef != mid) add_line(nw, 2 * v, l, m);
  else add_line(nw, 2 * v + 1, m, r);
}
int get(int x, int v=1, int l=0, int r = maxn) {
  int m = (l + r) / 2;
  if(r - l == 1) return f(line[v], x);
  else if(x < m)
    return min(f(line[v], x), get(x, 2*v, l, m));
  else
    return min(f(line[v], x),get(x, 2*v+1, m, r));
}
```

33 Linear Sieve

```
const int N = 1e8; int lp[N+1]; vector<int> pr;
for(int i=2; i<=N; ++i) {
  if(lp[i] == 0) { lp[i] = i; pr.push_back (i); }
  for(int j=0; j<(int)pr.size() && pr[j]<=lp[i]
        && i*pr[j]<=N; ++j) lp[i * pr[j]] = pr[j];
}
```

34 Link Cut Tree

```
/*method: decomp. of da forest into "pref. paths"
,pref. path = more recently accessed. pref. path
is stored in a splay tree, keyed by depth. oper:
- make_tree(v): create a tree of one node v
- find_root(v): root of the tree containing v
- link(v, w): connect v to w (precond: v is root
    of its tree, and v and w !in same tree)
- cut(v): cut v off from its parent
- path(v): access the path from the root of v's
    tree to v (for query on path)
LCA implemented here
Complexity: O(1) for make_tree
  O(log n) amortized for all other operations*/
const int MAXN = 100007;
struct LinkCutTree {
  struct Node {
    int L, R, P, lazyFlip, PP;
  }LCT[MAXN];
  void normalize(int u) {
    assert(u != -1);
    if (LCT[u].L != -1) LCT[LCT[u].L].P = u;
    if (LCT[u].R != -1) LCT[LCT[u].R].P = u;
    // (+ update sum of subtree elements
  }
  // set v as p's left child
  void setLeftChild(int p, int v) {
    LCT[p].L = v;   normalize(p);
  }
  // set v as p's right child
  void setRightChild(int p, int v) {
    LCT[p].R = v;  normalize(p);
  }
  void pushLazy(int u) {
    if (!LCT[u].lazyFlip) return;
    swap(LCT[u].L, LCT[u].R);
    LCT[u].lazyFlip = 0;
    if (LCT[u].L!=-1) LCT[LCT[u].L].lazyFlip ^=1;
    if (LCT[u].R!=-1) LCT[LCT[u].R].lazyFlip ^=1;
  }
  void make_tree(int v) {
    LCT[v].L=LCT[v].R=LCT[v].P=LCT[v].PP = -1;
    LCT[v].lazyFlip = 0;
  }
  void rotate(int v) {
    if (LCT[v].P == -1) return;
    int p = LCT[v].P,   g = LCT[p].P;
    if (LCT[p].L == v) {
      setLeftChild(p, LCT[v].R);
      setRightChild(v, p);
    } else {
      setRightChild(p, LCT[v].L);
      setLeftChild(v, p);
    }
    LCT[v].P = g;
    if (g != -1) {
      if (LCT[g].L == p) setLeftChild(g, v);
      else  setRightChild(g, v);
    }
    //must preserve path-pointer! EFFECT for g=-1
    LCT[v].PP = LCT[p].PP;    LCT[p].PP = -1;
  }
  void pushEmAll(int v) {
    if (LCT[v].P != -1) pushEmAll(LCT[v].P);
    pushLazy(v);
  }

  void splay(int v) {
    pushLazy(v);
    while (LCT[v].P != -1) {
      int p = LCT[v].P,    g = LCT[p].P;
      if (g != -1) pushLazy(g);
      pushLazy(p);    pushLazy(v);
      if (g == -1) rotate(v); // zig
      else if ((LCT[p].L==v) == (LCT[g].L==p)) {
        rotate(p);  rotate(v);  // zig-zig
      } else { // zig-zag
        rotate(v);  rotate(v);
      }
    }
  }
  /// returns v = root auxiliary tree  otherwise
  /// returns topmost unpreferred edge's parent
  int access(int v) {
    splay(v); // now v is root of its aux. tree
    if (LCT[v].R != -1) {
      LCT[LCT[v].R].PP = v;
      LCT[LCT[v].R].P = -1;
      setRightChild(v, -1);
    }
    int ret = v;
    while (LCT[v].PP != -1) {
      int w = LCT[v].PP;    splay(w);
      if (LCT[w].PP == -1) ret = w;
      if (LCT[w].R != -1) {
        LCT[LCT[w].R].PP = w;
        LCT[LCT[w].R].P = -1;
      }
      LCT[v].PP = -1;
      setRightChild(w, v);  splay(v);
    }
    return ret;
  }
  int find_root(int v) {
    access(v);  int ret = v;
    while (LCT[ret].L != -1) {
      ret = LCT[ret].L;   pushLazy(ret);
    }
    access(ret);   return ret;
  }
  /// make w, parent of v where v is a root
  void link(int v, int w) {//attach v's root to w
    access(w);
    // the root can only have right children in
    // its splay tree, so no need to check
    setLeftChild(v, w);    LCT[w].PP = -1;
  }
  void cut(int v) {
    access(v);
    if (LCT[v].L != -1) {
      LCT[LCT[v].L].P = -1;
      LCT[LCT[v].L].PP = -1;
      setLeftChild(v, -1);
    }
  }
  void make_root(int v) {
    access(v);    int l = LCT[v].L;
    if (l != -1) {
      setLeftChild(v, -1);   LCT[l].P = -1;
      LCT[l].PP = v;   LCT[l].lazyFlip ^= 1;
    }
  }
  bool isConnected(int p, int q) {
    return find_root(p)==find_root(q);
  }
  /// assuming p and q is in the same tree
  int LCA(int p, int q) {
    access(p);   return access(q);
  }
};
int main() {
  int n, m;    cin >> n >> m;
  LinkCutTree lct;
  for (int i = 1; i <= n; i++) lct.make_tree(i);
  while (m--) {
```

```
    string cmd;   cin >> cmd;
    if (cmd == "link") {
      int p, q; cin >> p >> q;  lct.link(p, q);
    } else if (cmd == "cut") {
      int p; cin >> p; lct.cut(p);
    } else if (cmd == "lca") {
      int p, q;  cin >> p >> q;
      cout << lct.LCA(p, q) << "\n";
    }
  }
}
```

35 Manacher

```
///p[0][i]->maxlen of half palin around half idx i
///p[1][i]->maxlen of half palin around char i
VI p[2]; ///<-vector<int> ///0-based indexing
void manacher(const string s) {
  int n = s.size(); p[0] = VI(n+1); p[1] = VI(n);
  for (int z=0; z<2; z++)
    for (int i=0, l=0, r=0; i<n; i++) {
      int t = r - i + !z;
      if (i<r) p[z][i] = min(t, p[z][l+t]);
      int L = i-p[z][i], R = i+p[z][i] - !z;
      while (L>=1 && R+1<n && s[L-1] == s[R+1])
        p[z][i]++, L--, R++;
      if (R>r) l=L, r=R;
    }
} ///lengths: p[0][i]*2; p[1][i]*2+1
bool ispalin(int l, int r) {
  int mid = (l+r+1)/2, sz = r-l+1; bool b = sz%2;
  int len=p[b][mid]; len=2*len+b; return len>=sz;
}
```

36 Matroid: Colorful ∩ Graphic

```
/** Matroid Intersection Between Graphic matroid
and Colorful matroid | VI,VB-vector<int,bool> */
struct DSU {
  vector< int >p; int componentCount;
  DSU(int n):p(n+1) {
    for (int i = 1; i <= n; i++) p[i] = i;
    componentCount = n;
  }
  int parent(int u) {
    if (u==p[u]) return u;
    return p[u]=parent(p[u]);
  }
  bool different(PII p) {
    return parent(p.first) != parent(p.second);
  }
  void unite(int u, int v) {
    u=parent(u); v=parent(v); if (u==v) return;
    p[v]=u; componentCount--;
  }
  void processThemAll(const vector< PII >&edges,
                      const vector< bool >&take){
    for (int i = 0; i < edges.size(); i++) {
      if (take[i]) {
        unite(edges[i].first, edges[i].second);
      }
    }
  }
};
struct Graph {
  vector< vector< int > >edg;
  Graph(int n) : edg(n) { }
  void addEdge(int u, int v) {
    edg[u].push_back(v);
  }
  Graph reversedGraph() {
    Graph r(edg.size());
    for (int i = 0; i < edg.size(); i++)
      for (int v : edg[i]) r.addEdge(v, i);
    return r;
  }
  void unite(const Graph &g) {
    for (int u = 0; u < g.edg.size(); u++)
      for (int v : g.edg[u]) addEdge(u, v);
  }
};
Graph exchangeGraph4Graphic(int n, int m,
    const vector< PII >&edges, vector<bool>take){
  Graph g(m);
  for (int i = 0; i < m; i++) {
    if (!take[i]) continue; take[i] = false;
    DSU dsu(n); dsu.processThemAll(edges, take);
    if (dsu.componentCount==1) {
      for (int j = 0; j < m; j++)
        if (i!=j && !take[j]) g.addEdge(i, j);
    } else {
      for (int j = 0; j < m; j++)
        if (i != j && !take[j])
          if (dsu.different(edges[j]))
            g.addEdge(i, j);
    }
    take[i] = true;
  }
  return g;
}
Graph exchangeGraph4Colorful(int n, int m,
    VI &colors, VB &take, VB &ctaken) {
  Graph g(m);
  for (int i = 0; i < m; i++) {
    if (!take[i]) continue;
    for (int j = 0; j < m; j++) {
      if(j==i) continue;
      if(colors[j]==colors[i]) g.addEdge(i, j);
      else if(!ctaken[colors[j]]) g.addEdge(i, j);
    }
  }
  return g;
}
bool augment(int m, const Graph &g, VB &etaking,
  VI &colors, VB &ctaking, VB &source, VB &sink) {
  vector< int >parent(m, -2); queue< int >q;
  for (int i = 0; i < m; i++) {
    if (source[i]) { q.push(i); parent[i] = -1; }
  }
  int uu = -1;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    if (sink[u]) { uu = u; break; }
    for (int v : g.edg[u]) {
      if (parent[v] != -2) continue;
      parent[v] = u; q.push(v);
    }
  }
  if (uu == -1) return false;
  bool flag = true;
  while (uu != -1) {
    etaking[uu]=flag;  ctaking[colors[uu]]=flag;
    flag = !flag;  uu = parent[uu];
  }
  return true;
}
bool solve(int n, int k,
    const vector<PII>&edges, vector<int>&colors) {
  int m = edges.size();
  VB etaken(m, false), ctaken(k, false);
  while (true) {
    DSU dsu(n); dsu.processThemAll(edges, etaken);
    if (dsu.componentCount == 1) return true;
    VB source(m, false), sink(m, false);
    for (int i = 0; i < m; i++) {
```

```
      if (etaken[i]) continue;
      if (dsu.different(edges[i])) source[i]=true;
    }
    for (int i = 0; i < m; i++) {
      if (etaken[i]) continue;
      if (!ctaken[colors[i]]) sink[i] = true;
    }
    bool trivial = false;
    for (int i = 0; i < m; i++) {
      if (sink[i] && source[i]) {
        trivial = etaken[i] = true;
        ctaken[colors[i]] = true; break;
      }
    }
    if (trivial) continue;
    Graph exchangeGraph =
      exchangeGraph4Graphic(n, m, edges, etaken);
    exchangeGraph.unite(exchangeGraph4Colorful
                        (n,m,colors,etaken,ctaken));
    //^^ WARNING: output of exchangeGraph4Colorful
    ↪  must be reversed before united with
    ↪  exchangeGraph
    if (!augment(m, exchangeGraph, etaken, colors,
        ctaken, source, sink)) break;
  }
  return false;
}
int main() {
  int n, m, k; int ti = 0;
  while (cin >> n >> m >> k) {
    vector< PII >edge(m); vector< int >col(m);
    for (int i = 0; i < m; i++) {
      cin>>edge[i].first>>edge[i].second>>col[i];
      col[i]--;
    } ++ti;
    cout << "Instancia " << ti << "\n";
    cout << (solve(n, k, edge, col)?"sim":"nao");
    cout << "\n\n";
  }
  return 0;
}
```

37 Matroid: Colorful ∩ Linear

```
/** Matroid Intersection Between Linear matroid
and Colorful matroid | VI,VB-vector<int,bool> */
const int MAXK = 60;
int gauss(vector< LL >&v) {
  int n = v.size(); int i = 0; int ans = 0;
  for (int j = 0; j < MAXK && i < n; j++) {
    if ((v[i]&(1LL<<j))==0) {
      int k = i+1;
      for ( ; k < n && (v[k]&(1LL<<j))==0; k++);
      if (k >= n) continue; swap(v[i], v[k]);
    }
    ans++;
    for (int k = i+1; k < n; k++) {
      if (v[k]&(1LL<<j)) v[k] ^= v[i];
    }
    i++;
  }
  return ans;
}
// INSERT Graph from ``MatroidColorGraph.cc"
Graph exchangeGraph4Linear(const VL &alice,
    const VL &bob, VB &taken) {
  int m = bob.size(); Graph g(m);
  for (int i = 0; i < m; i++) {
    if (!taken[i]) continue; taken[i] = false;
    vector< LL >tmp(alice);
    for (int j = 0; j < m; j++)
      if (taken[j]) tmp.push_back(bob[j]);
    gauss(tmp);
    for (int j = 0; j < m; j++) {
      if (taken[j]) continue; if (i==j) continue;
      LL x = bob[j];
      for (LL y : tmp) {
        if (__builtin_ctzll(y)==
            __builtin_ctzll(x)) x ^= y;
      }
      if (x > 0) g.addEdge(i, j);
    }
    taken[i] = true;
  }
  return g;
}
Graph buildExchangeGraphForColor(const VI &color,
    const VB &ctaken, VB &taken) {
  int m = color.size(); Graph g(m);
  for (int i = 0; i < m; i++) {
    if (!taken[i]) continue; taken[i] = false;
    for (int j = 0; j < m; j++) {
      if (taken[j]) continue; if (i==j) continue;
      if (color[i]==color[j]) g.addEdge(i, j);
      else if (!ctaken[color[j]]) g.addEdge(i, j);
    }
    taken[i] = true;
  }
  return g;
}
bool augment(int m, const Graph &g, VB &taken,
    VI &color, VB &ctaken, VB &source, VB &sink) {
  vector< int >parent(m, -2); queue< int >q;
  for (int i = 0; i < m; i++)
    if (source[i]) { q.push(i); parent[i] = -1; }
  int uu = -1;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    if (sink[u]) { uu = u; break; }
    for (int v : g.edg[u]) {
      if (parent[v] != -2) continue;
      parent[v] = u; q.push(v);
    }
  }
  if (uu == -1) return false; bool flag = true;
  while (uu != -1) {
    taken[uu] = flag; ctaken[color[uu]] = flag;
    flag = !flag; uu = parent[uu];
  }
  return true;
}

int main() {
  int n; cin >> n; vector< LL >alice(n);
  for (int i = 0; i < n; i++) cin >> alice[i];
  if (n > 0 && gauss(alice) < n) {
    cout << -1 << endl; return 0;
  }
  vector< LL >bob; vector< int >color;
  int m; cin >> m;
  for (int i = 0; i < m; i++) {
    int sz; cin >> sz;
    while (sz--) {
      LL x; cin >> x;
      bob.push_back(x); color.push_back(i);
    }
  }
  VB taken(bob.size(), false), ctaken(m, false);
  for (int step = 1; step <= m; step++) {
    VB source(bob.size(), false);
    VB sink(bob.size(), false);
    vector< int >current;
    for (int j = 0; j < bob.size(); j++)
      if (taken[j]) current.push_back(j);
    for (int i = 0; i < bob.size(); i++) {
      if (taken[i]) continue;
```

```
      vector< LL >tmp(alice);
      for (int j : current) tmp.push_back(bob[j]);
      tmp.push_back(bob[i]);
      if (gauss(tmp) == n+step) source[i] = true;
    }
    for (int i = 0; i < bob.size(); i++) {
      if (taken[i]) continue;
      if (!ctaken[color[i]]) sink[i] = true;
    }
    bool trivial = false;
    for (int i = 0; i < bob.size(); i++) {
      if (sink[i] && source[i]) {
        trivial = true;
        taken[i] = true;
        ctaken[color[i]] = true;
        break;
      }
    }
    if (trivial) continue;
    Graph exchangeGraph =
      exchangeGraph4Linear(alice, bob, taken);
    exchangeGraph.unite(buildExchangeGraphForColor
      (color, ctaken, taken).reversedGraph());
    if (!augment(bob.size(), exchangeGraph, taken,
          color, ctaken, source, sink)) {
      cout << -1 << endl;
      return 0;
    }
  }
  for (int i = 0; i < bob.size(); i++) {
    if (taken[i]) cout << bob[i] << "\n";
  }
  return 0;
}
```

38 Max Flow: Dinic

In a network flow $G = (V, E)$, every edge now has a demand function $d(e)$ which tells us that at least this much flow should be passed through this edge. Obviously $d(e) \leq f(e) \leq c(e)$; $f(e)$ is the flow and $c(e)$ is the capacity of edge e. To check whether such a flow can be found, we add a new source s' and a new sink t' besides the old s, t. Our capacity function:

- $c'((s', v)) = \sum_{u \in V} d((u, v))$
- $c'((v, t')) = \sum_{w \in V} d((v, w))$
- $c'((u, v)) = c((u, v)) - d((u, v)) \ \forall \ (u, v) \in E$
- $c'((t, s)) = \infty$

There's a feasible flow if maximum flow from s' to t' saturates every outgoing edge from s' and every incoming edge of t'.
Finding Maximal Flow: Find any feasible flow. Run a max flow from old s to t in the augmented graph. This flow is the maximal flow.
Finding Minimal Flow: Notice that the edge (t, s) with capacity ∞ carries the entire flow in the old graph. Binary search on it's capacity and check for the minimal capacity that still makes the network feasible. That capacity is the minimal flow.

```
#define INF 2000000000
const int MAX_E=60003;
const int MAX_V=5003;
int ver[MAX_E],cap[MAX_E],nx[MAX_E],last[MAX_V],
 ds[MAX_V],st[MAX_V],now[MAX_V],edg_cnt,S,T;

inline void reset() {
  memset(nx,-1,sizeof(nx));
  memset(last,-1,sizeof(last));
  edg_cnt=0;
}
inline void addedge(const int v,const int w,
  const int capacity,const int reverse_capacity) {
  ver[edg_cnt]=w; cap[edg_cnt]=capacity;
  nx[edg_cnt]=last[v]; last[v]=edg_cnt++;
  ver[edg_cnt]=v; cap[edg_cnt]=reverse_capacity;
  nx[edg_cnt]=last[w]; last[w]=edg_cnt++;
}
inline bool bfs() {
  memset(ds,-1,sizeof(ds));
  int a,b;
  a=b=0;
  st[0]=T;
  ds[T]=0;
  while (a<=b) {
    int v=st[a++];
    for (int w=last[v];w>=0;w=nx[w]) {
      if (cap[w^1]>0 && ds[ver[w]]==-1) {
        st[++b]=ver[w];
        ds[ver[w]]=ds[v]+1;
      }
    }
  }
  return ds[S]>=0;
}
int dfs(int v,int cur) {
  if (v==T) return cur;
  for (int &w=now[v];w>=0;w=nx[w]) {
    if (cap[w]>0 && ds[ver[w]]==ds[v]-1) {
      int d=dfs(ver[w],min(cur,cap[w]));
      if (d) {
        cap[w]-=d;
        cap[w^1]+=d;
        return d;
      }
    }
  }
  return 0;
}
inline long long flow() {
  long long res=0;
  while (bfs()) {
    for (int i=0;i<MAX_V;i++) now[i]=last[i];
    while (1) {
      int tf=dfs(S,INF);
      res+=tf;
      if (!tf) break;
    }
  }
  return res;
}
```

39 Min Cost Max Flow

```
// 0 Based indexed for directed weighted graphs
// for undirected graphs, add two directed edges
namespace mcmf {
  const int MAX = 1000010;
  const ll INF = 1ll << 60;
  ll cap[MAX], flow[MAX], cost[MAX], dis[MAX];
  int n, m, s, t, Q[10000010], adj[MAX],
    link[MAX], last[MAX], from[MAX], visited[MAX];

  void init(int nodes, int source, int sink){
    m = 0, n = nodes, s = source, t = sink;
    for (int i = 0; i <= n; i++) last[i] = -1;
  }
  void addEdge(int u, int v, ll c, ll w){
    adj[m]=v, cap[m]=c, flow[m]=0, cost[m]=+w,
      link[m]=last[u], last[u]=m++;
    adj[m]=u, cap[m]=0, flow[m]=0, cost[m]=-w,
```

```
    link[m]=last[v], last[v]=m++;
  }
  bool spfa(){
    int i, j, x, f = 0, l = 0;
    for (i = 0; i <= n; i++)
      visited[i] = 0, dis[i] = INF;
    dis[s] = 0, Q[l++] = s;
    while (f < l){
      i = Q[f++];
      for (j = last[i]; j != -1; j = link[j]){
        if (flow[j] < cap[j]){
          x = adj[j];
          if (dis[x] > dis[i] + cost[j]){
            dis[x] = dis[i] + cost[j],from[x] = j;
            if (!visited[x]){
              visited[x] = 1;
              if (f && rand() & 7) Q[--f] = x;
              else Q[l++] = x;
            }
          }
        }
      }
      visited[i] = 0;
    }
    return (dis[t] != INF);
  }
  pair <ll, ll> maxFlow(){
    int i, j;
    ll mincost = 0, maxflow = 0;
    while (spfa()){
      ll aug = INF;
      for (i = t, j = from[i]; i != s;
        i = adj[j ^ 1], j = from[i]){
        aug = min(aug, cap[j] - flow[j]);
      }
      for (i = t, j = from[i]; i != s;
        i = adj[j ^ 1], j = from[i]){
        flow[j] += aug, flow[j ^ 1] -= aug;
      }
      maxflow += aug, mincost += aug * dis[t];
    }
    return make_pair(maxflow, mincost);
  }
}
```

40 Minimum Enclosing Circle

```
inline circle trivial(vector<point> v){
  switch(v.size()){
    case 0:  return circle{point(), 0.0};
    case 1:  return circle{v[0], 0.0};
    case 2:  return circumcircle(v[0], v[1]);
    case 3:
      sort(v.begin(), v.end(), comp);
      if(eq(orient(v[0], v[1], v[2]), 0))
        return circumcircle(v[0], v[2]);
      else  return circumcircle(v[0], v[1], v[2]);
    default:  assert(false);
  }
}
vector<point> arr;  mt19937 rng(time(0));
circle MEC(int id, vector<point> &bound){
  if(id == arr.size() || bound.size() == 3)
    return trivial(boundary);
  if(id==0) shuffle(arr.begin(), arr.end(), rng);
  circle c = boundingCircle(id + 1, bound);
  if(inside(arr[id], c))  return c;
  else {
    bound.push_back(arr[id]);
    c = boundingCircle(id + 1, bound);
    bound.pop_back();   return c;
  }
}
```

41 MO

```
struct query {
  int l, r, idx;
  bool operator < (const query &p) const {
    if(l/block_sz != p.l/block_sz) return l < p.l;
    return ((l/block_sz) & 1) ? r > p.r : r < p.r;
  }  // use with care
};
vector<int> mo(vector<query> &q) {
  sort(q.begin(), q.end());
  vector<int> ret(q.size());
  // l = 1, r = 0 if 1-indexed array
  int l = 0, r = -1;
  for(auto &qq : q) {
    while(qq.l < l) add(--l);
    while(qq.r > r) add(++r);
    while(qq.l > l) erase(l++);
    while(qq.r < r) erase(r--);
    ret[qq.idx] = get_ans();
  }
  return ret;
}
```

42 MO with Dancing Links

```
/** Given a sequence a[]. In each query given l
and r, find min{|a_s - a_t|} where l s, t r, s t
Solution: MO with Dancing Links. First do
buildList(), then do delete and insert in the
same order, if delete A->B->C, insert C->B->A */
#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int>PII;
const int MAXN = 300007, INF = 1e9+7, magic = 158;
int n; int a[MAXN]; int dp[MAXN][magic+7];
void preCal() {
  for (int i = 0; i < n; i++) dp[i][0] = INF;
  for (int j = 1; j < magic; j++)
    for (int i = 0; i+j < n; i++)
      dp[i][j] = min(abs(a[i]-a[i+j]),
                     min(dp[i][j-1], dp[i+1][j-1]));
}
int l[MAXN], r[MAXN]; int ans[MAXN]; int listSize;
vector<int>qr[MAXN];  int ajib[MAXN];
int pred[MAXN],succ[MAXN],anew[MAXN],idnew[MAXN];
void buildList(const set<PII> &st) {
  int i = 0; listSize = st.size();
  for (PII p : st) {
    anew[++i] = p.first; idnew[p.second] = i;
  }
  for (int i = 1; i <= listSize; i++) {
    pred[i] = i-1; succ[i] = i+1;
  }
}
void del(int idx) {
  pred[succ[idx]] = pred[idx];
  succ[pred[idx]] = succ[idx];
}
void insert(int idx) {
  succ[pred[idx]] = pred[succ[idx]] = idx;
}
int getMin(int idx) {
  int rt = INF;
  if (pred[idx] > 0)
    rt = min(rt, anew[idx]-anew[pred[idx]]);
  if (succ[idx] <= listSize)
    rt = min(rt, anew[succ[idx]]-anew[idx]);
  return rt;
}
int cmp(int i, int j) { return r[i] > r[j]; }
```

```
int main() {
  ios::sync_with_stdio(false); cin.tie(0);
  cin >> n;
  for (int i = 0; i < n; i++) cin >> a[i];
  int m;  cin >> m;  preCal();
  for (int i = 0; i < m; i++) {
    cin >> l[i] >> r[i];  l[i]--;  r[i]--;
    if (r[i]-l[i] < magic)
      ans[i] = dp[l[i]][r[i]-l[i]];
    else qr[l[i]/magic].push_back(i);
  }
  set<PII>st;
  for (int i = n/magic+3; i >= 0; i--) {
    int x = i*magic; if (x >= n) continue;
    int y = min(i*magic+magic-1, n-1);
    st.emplace(a[y], y); buildList(st);
    for (int k = n-1; k > y; k--) del(idnew[k]);
    ajib[y] = INF;
    for (int k = y+1; k < n; k++) {
      insert(idnew[k]);
      ajib[k] = min(ajib[k-1], getMin(idnew[k]));
    }
    for(int k = x; k < y; k++) st.emplace(a[k],k);
    buildList(st);
    sort(qr[i].begin(), qr[i].end(), cmp);
    int last = n-1;
    for (int qi : qr[i]) {
      assert(r[qi] > y);
      while(last > r[qi]) {
        del(idnew[last]); last--;
      }
      for (int k = x; k < y; k++) del(idnew[k]);
      ans[qi] = ajib[r[qi]];
      for (int k = y-1; k >= x; k--) {
        insert(idnew[k]);
        if (k >= l[qi])
          ans[qi] = min(ans[qi],getMin(idnew[k]));
      }
    }
  }
  for(int i = 0; i<m; i++) cout << ans[i] << "\n";
  return 0;
}
```

43 Mod Multiplication

```
ll modmult(ll a, ll b, ll m){
  if(a >= m) a %= m; if(b >= m) b %= m;
  ull q = ull(ld(a)*b/m);
  ll r = ll(a*b-q*m) % ll(m);
  return (r < 0)? r+m : r;
}
```

44 NTT

```
/** Possible Optimizations:
1. Remove leading zeroes. 2. You may use ints
everywhere, but be careful with overflow
998244353, 15311432, 469870224, 1<<23
If required mod is not above,
use nttdata function OFFLINE **/
LL power(LL a, LL p, LL mod) {
    if (p==0)   return 1;
    LL ans = power(a, p/2, mod);
    ans = (ans * ans)%mod;
    if (p%2)    ans = (ans * a)%mod;
    return ans;
}
/** Find primitive root of p assuming p is prime.
if not, we must add calculation of phi(p)
Returns -1 if not found */
int primitive_root(int p) {
  vector<int> factor;
  int phi = p-1, n = phi;
  for (int i=2; i*i<=n; ++i)
    if (n%i == 0) {
      factor.push_back (i);  while (n%i==0) n/=i;
    }
  if (n>1)  factor.push_back(n);
  for (int res=2; res<=p; ++res) {
    bool ok = true;
    for (int i=0; i<factor.size() && ok; ++i)
      ok &= power(res, phi/factor[i], p) != 1;
    if (ok)  return res;
  }
  return -1;
}
/** returns maximum k such that 2^k divides mod
ntt can only be applied for arrays not larger than
this size. mod MUST BE PRIME!!!!! */
int nttdata(int mod,int &root,int &inv,int &pw) {
  int c=0, n=mod-1;  while (n%2 == 0) c++, n/=2;
  pw = (mod-1)/n; int g = primitive_root(mod);
  root=power(g,n,mod); inv=power(root,mod-2,mod);
  return c;
}
struct NTT {
  int N; vector<int> perm;
  int mod, root, inv, pw;
  NTT(int mod, int root, int inv, int pw) :
    mod(mod), root(root), inv(inv), pw(pw) {}
  void precalculate() {
    perm.resize(N); perm[0] = 0;
    for (int k=1; k<N; k<<=1) {
      for (int i=0; i<k; i++) {
        perm[i] <<= 1; perm[i+k] = 1 + perm[i];
      }
    }
  }
  void fft(vector<LL> &v, bool invert = false) {
    if (v.size() != perm.size()) {
      N = v.size(); assert(N && (N&(N-1)) == 0);
      precalculate();
    }
    for (int i=0; i<N; i++)
      if (i < perm[i]) swap(v[i], v[perm[i]]);
    for (int len = 2; len <= N; len <<= 1) {
      LL factor = invert ? inv : root;
      for (int i = len; i < pw; i <<= 1)
        factor = (factor * factor) % mod;
      for (int i=0; i<N; i+=len) {
        LL w = 1;
        for (int j=0; j<len/2; j++) {
          LL x=v[i+j], y = (w * v[i+j+len/2])%mod;
          v[i+j] = (x+y)%mod;
          v[i+j+len/2] = (x-y+mod)%mod;
          w = (w * factor)%mod;
        }
      }
    }
    if (invert) {
      LL n1 = power(N, mod-2, mod);
      for (LL &x : v) x=(x*n1)%mod;
    }
  }
  vector<LL> multiply(vector<LL> a, vector<LL> b){
    int n=1; while (n < a.size()+ b.size()) n<<=1;
    a.resize(n); b.resize(n);
    fft(a); fft(b);
    for(int i=0; i<n; i++) a[i] = (a[i]*b[i])%mod;
    fft(a, true);
    return a;
  }
};
int main() {
```

```
  int mod = 7340033; int root, inv, pw;
  int k = nttdata(mod, root, inv, pw);
  vector<LL> left = {1,4,6,4,1};
  NTT ntt(mod, root, inv, pw);
  vector<LL> ans = ntt.multiply(left, left);
  for (LL x: ans)    cout<<x<<" ";
}
```

45 NTT: Any Mod

```
/** Can Handle any mod that fits in int
Possible Optimizations: 1. Refer to FFT and NTT
optimizations. 2. If mod is < 2^26 use RT = 13 and
comment out long double */
#define double long double
typedef complex<double> CD; const int RT = 16;
struct FFT {
  int N;  vector<int> perm;
  void precalculate() {
    perm.resize(N);  perm[0] = 0;
    for (int k=1; k<N; k<<=1) {
      for (int i=0; i<k; i++) {
        perm[i] <<= 1;  perm[i+k] = 1 + perm[i];
      }
    }
  }
  void fft(vector<CD> &v, bool invert = false) {
    if (v.size() != perm.size()) {
      N = v.size(); assert(N && (N&(N-1)) == 0);
      precalculate();
    }
    for (int i=0; i<N; i++)
      if (i < perm[i]) swap(v[i], v[perm[i]]);
    for (int len = 2; len <= N; len <<= 1) {
      double angle = 2 * PI / len;
      if (invert) angle = -angle;
      CD factor = polar(1.0L, angle);
      for (int i=0; i<N; i+=len) {
        CD w(1);
        for (int j = 0; j < len/2; j++) {
          CD x = v[i+j], y = w * v[i+j+len/2];
          v[i+j]=x+y; v[i+j+len/2]=x-y; w*=factor;
        }
      }
    }
    if (invert) for (CD &x : v) x/=N;
  }
  vector<LL>multiply(  const vector<LL> &a,
                       const vector<LL> &b, int M) {
    int n = 1; while(n < a.size()+b.size()) n<<=1;
    vector<CD> al(n), ar(n), bl(n), br(n);
    for (int i=0; i<a.size(); i++) {
      LL k = a[i]%M;   al[i] = k >> RT;
      ar[i] = k & ((1<<RT)-1);
    }
    for (int i=0; i<b.size(); i++) {
      LL k = b[i]%M;  bl[i] = k >>RT;
      br[i] = k & ((1<<RT)-1);
    }
    fft(al); fft(ar); fft(bl); fft(br);
    for (int i=0; i<n; i++) {
      CD ll = (al[i]*bl[i]); CD lr= (al[i]*br[i]);
      CD rl = (ar[i]*bl[i]); CD rr= (ar[i]*br[i]);
      al[i]=ll;  ar[i]=lr;  bl[i]=rl;  br[i]=rr;
    }
    fft(al,true); fft(ar,true); fft(bl,true);
    fft(br,true); vector<LL> ans(n);
    for (int i=0; i<n; i++) {
      LL right = round(br[i].real()); right %= M;
      LL mid = round(  round(bl[i].real())
                     +  round(ar[i].real()) );
      mid = ((mid%M)<<RT)%M;
      LL left = round(al[i].real());
      left = ((left%M)<<(2*RT))%M;
      ans[i] = (left+mid+right)%M;
    }
    return ans;
  }
};
```

46 Order Statistics Tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
// find_by_order(k): iter to k'th largest 0-idx
// order_of_key(val): ele strict smaller than val
typedef tree<int,null_type,less<int>,rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
```

47 Palindromic Tree

```
const int MAXN = 105000;
struct node { int len, sufflink, num, next[26]; };
string s; int len; node tree[MAXN];
int num;///1: root with len -1; 2: root with len 0
int suff; /// max suffix palindrome
long long ans;
bool addLetter(int pos) {
  int cur=suff, curlen=0, let=s[pos] - 'a';
  while (true) {
    curlen = tree[cur].len;
    if (pos-1-curlen>=0&&s[pos-1-curlen]==s[pos])
      break;
    cur = tree[cur].sufflink;
  }
  if (tree[cur].next[let]) {
    suff = tree[cur].next[let];
    return false;
  }
  num++; suff = num; tree[cur].next[let]=num;
  tree[num].len=tree[cur].len+2;
  if (tree[num].len==1) {
      tree[num].sufflink = 2, tree[num].num = 1;
      return true;
  }
  while (true) {
    cur=tree[cur].sufflink;curlen=tree[cur].len;
    if (pos-1-curlen>=0&&s[pos-1-curlen]==s[pos]){
      tree[num].sufflink=tree[cur].next[let];
      break;
    }
  }
  tree[num].num=1+tree[tree[num].sufflink].num;
  return true;
}
void initTree() {
    memset(tree, 0, sizeof tree); num=2; suff=2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}
///number of palindromic sub's banana->10, abba->6
int main() {
  cin >> s; len = s.size(); initTree();
  for (int i = 0; i < len; i++) {
    addLetter(i); ans += tree[suff].num;
  }
  cout << ans << endl;
  return 0;
}
```

48 Persistent Segment Tree: Lazy

```
//PSegment Tree with Lazy. Arithmatic range update
struct Node{
```

```
  LL sum;LL lazyA=0,lazyB=0; bool hasLazy=false;
  Node* bam, *dan;
  Node(LL s=0, Node *b=NULL, Node *d=NULL):
    sum(s), bam(b), dan(d) { }
  void init(int l, int r) {
    if (l==r) return;   int mid = (l+r)/2;
    bam = new Node();   dan = new Node();
    bam->init(l, mid);  dan->init(mid+1, r);
  }
  void propagate(int l, int r) {
    if (!hasLazy) return;    LL n = r-l+1;
    sum += n*lazyA;   sum += ((n*(n-1))/2)*lazyB;
    if (l!=r) {
      bam = new Node(*bam); dan = new Node(*dan);
      int mid = (l+r)/2;    dan->lazyB += lazyB;
      dan->lazyA += lazyA+lazyB*(mid-l+1);
      bam->lazyA += lazyA; bam->lazyB += lazyB;
      bam->hasLazy = true; dan->hasLazy = true;
    }
    lazyA = 0;   lazyB = 0;  hasLazy = false;
  }
  Node* update(int l, int r, int x, int y, LL a,
  ↪  LL b) {
    propagate(l, r);
    if (r < x || y < l) return this;
    if (x <= l && r <= y) {
      Node* rt = new Node(*this);
      rt->lazyA += a+(l-x)*b; rt->lazyB += b;
      rt->hasLazy = true; rt->propagate(l, r);
      return rt;
    }
    int mid = (l+r)/2; Node* rt = new Node();
    rt->bam = bam->update(l, mid, x, y, a, b);
    rt->dan = dan->update(mid+1, r, x, y, a, b);
    rt->sum = rt->bam->sum + rt->dan->sum;
    return rt;
  }
  LL query(int l, int r, int x, int y) {
    propagate(l, r);
    if (x <= l && r <= y) return sum;
    LL ans = 0; int mid = (l+r)/2;
    if (x<=mid) ans += bam->query(l, mid, x, y);
    if (mid<y)  ans += dan->query(mid+1, r, x, y);
    return ans;
  }
};
```

49 Polar Sort

```
bool half(pt p) { // true if in blue half
  assert(p.x != 0 || p.y != 0); // (0,0) is undef
  return p.y > 0 || (p.y == 0 && p.x < 0);
}
void polarSort(vector<pt> &v) {
  sort(v.begin(), v.end(), [](pt v, pt w) {
    return make_tuple(half(v), 0, sq(v)) <
      make_tuple(half(w), cross(v,w), sq(w));
  });   // uses magnitude as tie breaker
} // sort around origin **********************
pt v = {/* whatever you want except 0,0 */};
bool half(pt p) {  // v is the first angle
  return cross(v,p) < 0 ||
    (cross(v,p) == 0 && dot(v,p) < 0);
}
```

50 Pollard Rho

```
LL powMod(LL x, LL k, LL m) {
  if (k == 0) return 1;
  if(k & 1) return mulMod(x,powMod(x, k-1, m), m);
  else return powMod(mulMod(x,x,m), k/2, m);
}
bool suspect(LL a, LL s, LL d, LL n) {
  LL x = powMod(a, d, n);
  if (x == 1) return true;
  for (int r = 0; r < s; ++r) {
    if (x == n - 1) return true;
    x = mulMod(x, x, n);
  }
  return false;
}
// {2,7,61,-1} is for n < 4759123141 (= 2^32)
// {2,3,5,7,11,13,17,19,23,-1} for n < 10^15
// LL SPRP = {2LL, 325LL, 9375LL, 28178LL,
//   450775LL, 9780504LL, 1795265022LL};
bool isPrime(LL n) {
  if (n <= 1 || (n > 2 && n % 2 == 0))
    return false;
  LL test[] = {2,3,5,7,11,13,17,19,23,-1};
  LL d = n - 1, s = 0;
  while (d % 2 == 0) ++s, d /= 2;
  for(int i=0; test[i] < n && test[i] != -1; ++i)
    if (!suspect(test[i], s, d, n)) return false;
  return true;
}
// KillerPr: 5555555557LL (fail when !used mulMod)
```

```
LL pollard_rho(LL n, LL seed) { //call factorize
  LL x, y;
  x = y = rand() % (n - 1) + 1;
  int head = 1, tail = 2;
  while (true) {
    x = mulMod(x, x, n); x = (x + seed) % n;
    if (x == y) return n;
    LL d = gcd(max(x - y, y - x), n);
    if (1 < d && d < n) return d;
    if (++head == tail) y = x, tail <<= 1;
  }
}
void factorize(LL n, vector<LL> &divisor) {
  if (n == 1) return;
  if (isPrime(n)) divisor.push_back(n);
  else {
    LL d = n;
    while(d >= n)
      d = pollard_rho(n, rand() % (n - 1) + 1);
    factorize(n / d, divisor);
    factorize(d, divisor);
  }
}
vector<LL> divisors(vector<LL> d) {
  vector<LL> ret = {1};
  sort(d.begin(), d.end());
  for (int i = 0, count = 1; i < d.size(); ++i) {
    if(i + 1 == d.size() || d[i] != d[i + 1]) {
      int c = ret.size();
      ret.resize(ret.size() * (count+1));
      LL n = 1;
      for (int j = 1; j <= count + 1; ++j) {
        for (int k = 0; k < c; ++k)
          ret[(j-1)*c+k] = ret[k]*n;
        n *= d[i];
      }
      count = 1;
    }
    else count += 1;
  }
  sort(ret.begin(), ret.end());
  return ret;
}
int main() {
  LL n; cin>>n;
  VL v; factorize(n,v);
  VL v1; v1=divisors(v);
  REP(i,0,v1.size()-1)cout<<v1[i]<<' ';
  return 0;
```

```
}
```

51 Polygon Stabbing: Extreme PTs

```
// Extreme Point for a direction is the farthest
// point in that dir| poly is convex, CCW, no
// redundant points| u direction for extremeness
int extremePoint(const vector<PT> &poly,
                                 PT u=PT(0, 1)) {
  int n = (int) poly.size(); int a = 0, b = n;
  while(b - a > 1) {
    int c = (a + b) / 2;
    if(dot(poly[c] - poly[(c+1)%n], u) > 0 and
          dot(poly[c] - poly[(c-1+n)%n], u) > 0)
      return c;
    long double a_dot = dot(poly[(a+1)%n] -
                                    poly[a], u);
    long double c_dot = dot(poly[(c+1)%n] -
                                    poly[c], u);
    // bool a_zer = fabs(a_dot) < EPS;
    // bool c_zer = fabs(c_dot) < EPS;
    bool a_up = !(a_dot < 0);
    bool c_up = !(c_dot < 0);
    bool a_above_c = dot(poly[a]-poly[c], u) > 0;
    // if(a_zer or c_zer) {
    //   if(a_zer and c_zer) {
    //     if(a_above_c) return a;
    //     else return c;
    //   }
    //   else if(a_zer) {
    //     if(c_up) a = c;
    //     else b = c;
    //   }
    //   else {
    //     if(a_up) b = c;
    //     else a = c;
    //   }
    //   continue;
    // }
    if(a_up and !c_up) b = c;
    else if(!a_up and c_up) a = c;
    else if(a_up and c_up) {
      if(a_above_c) b = c;
      else a = c;
    }
    else {
      if(!a_above_c) b = c;
      else a = c;
    }
  }
  if(dot(poly[a] - poly[(a+1)%n], u) > 0 and
    dot(poly[a]-poly[(a-1+n)%n], u) > 0) return a;
  return b % n;
}
//true if LINE x-y intersex poly, false otherwise
bool stabConvexPolygon(const vector<PT> &poly,
                                      PT x, PT y) {
  PT vec = y - x;
  int p1 = extremePoint(poly, RotateCCW90(vec));
  int p2 = extremePoint(poly, RotateCW90(vec));
  if(p1 == p2)
    return LinesCollinear(x, y, x, poly[p1]);
  if(LinesParallel(x, y, poly[p1], poly[p2]))
    return false;
  PT c = ComputeLineIntersection(x, y, poly[p1],
                                       poly[p2]);
  if(DistancePointSegment(poly[p1], poly[p2], c)
                                 > EPS) return false;
  return true;
}
```

52 Polynomial Interpolation

Given a set of $k+1$ data points $(x_0, y_0), ..(x_j, y_j), .., (x_k, y_k)$ where no two x_j are the same, the interpolation polynomial in the Lagrange form is a linear combination

$$L(x) := \sum_{j=0}^{k} y_j l_j(x)$$

of Lagrange basis polynomials

$$l_j(x) := \prod_{0 \le m \le k, m \ne j} \frac{x - x_m}{x_j - x_m}$$

where $0 \le j \le k$.

```
// Given n points (x[i], y[i]), computes an n-1
// degree polynomial p that passes through them:
// p(x) = a[0]*x^0 + ... + a[n-1]*x^{n-1}. O(n^2)
// To work mod a prime, use modinverse to divide
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n,0), temp(n,0);
  for(int k=0;k<n;k++) for(int i=k+1;i<n;i++)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  for(int k=0;k<n;k++) for(int i=0;i<n;i++) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

53 Random

```
#include <chrono>
mt19937 rng(chrono::steady_clock::now().
  time_since_epoch().count());  // together!
// uniform_int_distribution<int>(0, i)(rng)
// shuffle(arr.begin(), arr.end(), rng);
```

54 Rotating Calipers

```
double area(const PT &a,const PT &b,const PT &c) {
  return abs((b.x-a.x)*(c.y-a.y)
    - (b.y-a.y)*(c.x-a.x));
}
double dist(const PT &a, const PT &b) {
  return hypot(a.x - b.x, a.y - b.y);
}
// max distance of any two vertices
double diameter(const vector<PT> &p) {
  vector<PT> h = convexHull(p);  // ERR
  int m = h.size();
  if(m == 1) return 0;
  if(m == 2) return dist(h[0], h[1]);
  int k = 1;
  while(area(h[m - 1], h[0], h[(k + 1) % m])
    > area(h[m - 1], h[0], h[k])) ++k;
  double res = 0;
  for (int i = 0, j = k; i <= k && j < m; i++) {
    res = max(res, dist(h[i], h[j]));
    while(j<m && area(h[i],h[(i+1)%m],h[(j+1)%m])
      > area(h[i], h[(i + 1) % m], h[j])) {
      res = max(res, dist(h[i], h[(j + 1) % m]));
      ++j;
    }
  }
  return res;
}
```

55 Script

```
# For linux: terminal$ bash script.sh
g++ -std=c++14 solution.cpp -o mine
g++ -std=c++14 brute.cpp -o brute
g++ -std=c++14 testgen.cpp -o gen
echo 'Compilations Done!'
for((i=0; ; ++i)); do
  ./gen $i'' >in
  diff -w <(./mine <in) <(./brute <in) || break
  echo 'Test' $i 'checks out'
done
echo 'Okay'
# For Windows: cmd> checker.bat
# req: TestGenerator.exe solution.exe brute.exe
@echo off
for /l %%x in (1, 1, 100) do (
  TestGenerator > input.in
  solution < input.in > output.out
  brute < input.in > output2.out
  fc output.out output2.out >diagnostics|| exit /b
  echo %%x
)
echo all tests passed
```

56 Segment Tree: Lazy

```
const int MAXN = 1e5+7; /// change this
struct nd {
  int mn, cnt;
}tr[4*MAXN];
int lazy[4*MAXN];
//1. Merge left and right
nd combine(const nd &a, const nd &b) {
  if (a.mn < b.mn) return a;
  else if (a.mn > b.mn) return b;
  nd rt;   rt.mn = a.mn;
  rt.cnt = a.cnt+b.cnt;   return rt;
}
//2. Push lazy down and merge lazy
void propagate(int u, int l, int r) {
  if (lazy[u]) {
    tr[u].mn += lazy[u];
    if (l != r)
      lazy[u*2]+=lazy[u], lazy[u*2+1]+=lazy[u];
    lazy[u] = 0;
  }
}
int a[MAXN];
void build(int u, int l, int r) {
  lazy[u] = 0;     ///3. Initialize
  if (l==r) {
    tr[u].mn = 0; tr[u].cnt = a[l]; return;
  }
  int mid = (l+r)/2;
  build(u*2, l, mid);   build(u*2+1, mid+1, r);
  tr[u] = combine(tr[u*2], tr[u*2+1]);
}
void update(int u,int l,int r,int x,int y,int v){
  propagate(u, l, r);
  if (r < x || y < l) return;
  if (x <= l && r <= y) {
    lazy[u]+=v;     ///4. Merge lazy*
    propagate(u, l, r);   return;
  }
  int mid = (l+r)/2;
  update(u*2, l, mid, x, y, v);
  update(u*2+1, mid+1, r, x, y, v);
  tr[u] = combine(tr[u*2], tr[u*2+1]);
}
nd query(int u, int l, int r, int x, int y) {
  propagate(u, l, r);
  if (x <= l && r <= y) return tr[u];
  int mid = (l+r)/2;
  if(y <= mid)return query(u*2, l, mid, x, y);
  if(mid < x)return query(u*2+1, mid+1, r, x, y);
  return combine(query(u*2, l, mid, x, y),
                 query(u*2+1, mid+1, r, x, y));
}
```

57 SOS DP

```
// F(x) = sum A(i) such that i is subset of x
for(int i = 0; i<(1<<N); ++i)
  F[i] = A[i];
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)];
  }  // memory optimized ************************
for(int mask=0; mask<(1<<n); ++mask) {
  sub[mask][0] = 0;
  for(int i=0; i<n; ++i) {
    if(i) sub[mask][i] = sub[mask][i-1];
    if(mask & (1<<i)) {
      sub[mask][i] += sub[mask ^ (1<<i)][i]
                    + dp[mask ^ (1<<i)];
      while(sub[mask][i] >= MOD)
        sub[mask][i] -= MOD;
    }
  }
  dp[mask] -= sub[mask][n-1];
  if(dp[mask] < 0) dp[mask] += MOD;
}  // reverse problem ***************************
```

58 Suffix Array, Tree

```
const int maxn = 3e5+5; ///NOTICE
int wa[maxn],wb[maxn],wv[maxn],wc[maxn],r[maxn];
int sa[maxn],rak[maxn],height[maxn],SIGMA=0;
int cmp(int *r,int a,int b,int l)
  {return r[a]==r[b]&&r[a+l]==r[b+l];}
void da(int *r,int *sa,int n,int m) {
  int i,j,p,*x=wa,*y=wb,*t;
  for( i=0;i<m;i++) wc[i]=0;
  for( i=0;i<n;i++) wc[x[i]=r[i]] ++;
  for( i=1;i<m;i++) wc[i] += wc[i-1];
  for( i= n-1;i>=0;i--)sa[--wc[x[i]]] = i;
  for( j= 1,p=1;p<n;j*=2,m=p){
    for(p=0,i=n-j;i<n;i++)y[p++] = i;
    for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
    for(i=0;i<n;i++) wv[i] = x[y[i]];
    for(i=0;i<m;i++) wc[i] = 0;
    for(i=0;i<n;i++) wc[wv[i]] ++;
    for(i=1;i<m;i++) wc[i] += wc[i-1];
    for(i=n-1;i>=0;i--) sa[--wc[wv[i]]] = y[i];
    for(t=x,x=y,y=t,p=1,x[sa[0]] = 0,i=1;i<n;i++)
      x[sa[i]]= cmp(y,sa[i-1],sa[i],j) ? p-1:p++;
  }
}
void calheight(int *r,int *sa,int n){
  int i,j,k=0;
  for(i=1;i<=n;i++) rak[sa[i]] = i;
  for(i=0;i<n;height[rak[i++]] = k) {
   for(k?k--:0,j=sa[rak[i]-1];r[i+k]==r[j+k];k++);
  }
}
typedef vector< int >VI;
const int MAXX = maxn, K = 20; int lg[MAXX];
///CALL ME PLS
void pre() {
  lg[1]=0; for(int i=2;i<MAXX;i++)lg[i]=lg[i/2]+1;
}
struct RMQ{
  int N; VI v[K];
```

```
  RMQ(const VI &a) {
    N = a.size(); v[0] = a;
    for (int k = 0; (1<<(k+1)) <= N; k++) {
      v[k+1].resize(N);
      for (int i = 0; i-1+(1<<(k+1)) < N; i++) {
        v[k+1][i] = min(v[k][i], v[k][i+(1<<k)]);
      }
    }
  }
  int findMin(int i, int j) const {
    assert(i <= j); int k = lg[j-i+1];
    return min(v[k][i], v[k][j+1-(1<<k)]);
  }
};
void suffixArray(const string &s,
    vector< int >&suffArray, vector< int >&lcp) {
  int n = s.size(); SIGMA = 0;
  for(int i = 0; i < n; i ++) {
      if ('a'<=s[i]&&s[i]<='z')r[i]=s[i]-'a'+2;
      else r[i] = 1;  ///separators
      SIGMA = max(SIGMA, r[i]);
  }
  r[n] = 0; /// don't forget SIGMA+1.
  da(r, sa, n+1, SIGMA + 1); suffArray.resize(n);
  for (int i=0; i < n; i++) suffArray[i]=sa[i+1];
  calheight(r,sa,n); lcp.resize(n-1); ///optional
  for (int i=0; i+1 < n; i++) lcp[i]=height[i+2];
}
/** Given a length and a pos in suffix array, find
the maximal range [L, R] such that lcp between pos
and L and R is >= len */
PII extend(RMQ &rmq, int saSize, int ps, int len){
  int L = ps, R = ps;
  for (int k = K-1; k >= 0; k--) {
    int r = R+(1<<k); if (r >= saSize) continue;
    if (rmq.findMin(ps, r-1) >= len) R = r;
  }
  for (int k = K-1; k >= 0; k--) {
    int l = L-(1<<k); if (l < 0) continue;
    if (rmq.findMin(l, ps-1) >= len) L = l;
  }
  return PII(L, R);
}
///lngth rtrnd by backstep() mst b 'min'nd frm out
struct BackStepper {
  vector< int >startsWith[26];
  BackStepper(const string &s,
              const vector<int>&sa){
    for (int i = 0; i < sa.size(); i++) {
      if (sa[i] > 0)
        startsWith[s[sa[i]-1]-'a'].push_back(i);
    }
    startsWith[s.back()-'a'].push_back(s.size());
  }
  /**Return <len, j> s.t. s[j] = c and suffix[j+1]
     shares the longest prefix with suffix[i]
     Returns <0, 0> if no such index exists. */
  PII backstep(int i,int c,const vector< int >&sa,
          const vector< int >&ra, const RMQ &rmq){
    if (startsWith[c].empty()) return PII(0, 0);
    int ri = ra[i];
    int idx = lower_bound(startsWith[c].begin(),
    startsWith[c].end(),ri)-startsWith[c].begin();
    if (idx < startsWith[c].size() &&
      startsWith[c][idx]==ri) { ///same pos again
      return PII(ra.size()-i+1, i-1);
    }
    PII rt(-1, -1);
    if (idx > 0) {
      int ci = startsWith[c][idx-1];
      rt = PII(rmq.findMin(ci, ri-1)+1, sa[ci]-1);
    }
    if (idx < startsWith[c].size()) {
      int ci = startsWith[c][idx];
      if (ci==sa.size())
        rt=max(rt,PII(1, sa.size()-1));
      else rt = max(rt,
        PII(rmq.findMin(ri, ci-1)+1, sa[ci]-1));
    }
    return rt;
  }
};
/**SZ -> number of nodes in suffix tree. 0 is root
(the empty string). length of node u (starting
from root) is length[u]. node u belongs to all of
[L[u], R[u]] suffixes in Suffix Array */
///node number must be twice the size of string
struct SuffixTree {
  vector<int>edg[MAXX];
  int length[MAXX], L[MAXX], R[MAXX], SZ, n;
  void buildGraph(const vector< int >&sa,
      const vector<int>&lcp, const RMQ &rmq) {
    n = sa.size(); SZ = 1;
    length[0] = 0; L[0] = 0; R[0] = n-1;
    stack< int >st; st.push(0);
    for (int i = 0; i < sa.size(); i++) {
      int last = i==0?0:lcp[i-1];
      while (length[st.top()] > last) st.pop();
      ///IF strng has '#' inside dis need to chnge
      int len = n-sa[i];
      while (last < len) {
        int r = expandRight(i, last, rmq);
        if (i < r) last = rmq.findMin(i, r-1);
        else last = len;
        length[SZ] = last; L[SZ] = i; R[SZ] = r;
        edg[st.top()].push_back(SZ);
        st.push(SZ); SZ++;
      }
    }
  }
  int expandRight(int i,int last,const RMQ &rmq){
    int r = i;
    for (int k = K-1; k >= 0; k--) {
      int j = r+(1<<k); if (j >= n) continue;
      if (rmq.findMin(i, j-1) > last) r = j;
    }
    return r;
  }
} st;
void dfs(const string &s,vector<int>&sa,int u){
  for (int v : st.edg[u]) {
    cout << u << " -:";
    cout << s.substr(sa[st.L[v]]+st.length[u],
                     st.length[v]-st.length[u]);
    cout << ":- " << v << endl; dfs(s, sa, v);
  }
}
//If strng is a cnctntion of mr dan one strngs
//glud wth '#',deir lcp must b modfid b4 cllng RMQ
int main() {
  pre();    ///MUST for RMQ to be working
  string s; cin >> s;
  vector< int >sa, lcp;
  suffixArray(s, sa, lcp); RMQ rmq(lcp);
  for (int i : sa) cout << s.substr(i) << endl;
  for (int i : lcp) cout << i << " ";
  cout << endl; cout << rmq.N << endl;
  st.buildGraph(sa, lcp, rmq); dfs(s, sa, 0);
  return 0;
}
/** Input: banana
Output: a | ana | anana | banana | na | nana
1 3 0 0 2 |5| 0 -:a:- 1 | 1 -:na:- 2 | 2 -:na:- 3
0 -:banana:- 4 | 0 -:na:- 5 | 5 -:na:- 6 */
```

59 Suffix Automaton

```
const int alpha = 26;
//vector<int> n_list[nmax];  //len sorted list
struct state{
  int len, link, cnt, nxt[alpha];
  void init(int le = 0, int li = -1){
    len = le; link = li; cnt = 0;
    memset(nxt, -1, sizeof(nxt)); //mp
  }
};
state SA[2*nmax];
int sz, last;
void SA_init(){
  for(int i = 0; i<sz; i++) n_list[i].clear();
  SA[0].init(); sz = 1; last = 0;
  //n_list[0].push_back(0);
}
int SA_add(char c){
  c -= 'a'; int cur = sz++;
  SA[cur].init(SA[last].len+1);
  //n_list[SA[last].len+1].push_back(cur);
  int p = last;
  while(p != -1 && SA[p].nxt[c] == -1){ //mp
    SA[p].nxt[c] = cur; p = SA[p].link;
  }
  if(p == -1) SA[cur].link = 0;
  else{
    int q = SA[p].nxt[c];
    if(SA[p].len+1 == SA[q].len)SA[cur].link = q;
    else{
      int clone = sz++;
      SA[clone].init(SA[p].len+1, SA[q].link);
      memcpy(SA[clone].nxt,
             SA[q].nxt,sizeof(SA[q].nxt)); //mp
      //n_list[SA[p].len+1].push_back(clone);
      while(p != -1 && SA[p].nxt[c] == q){ //mp
        SA[p].nxt[c] = clone; p = SA[p].link;
      }
      SA[q].link = SA[cur].link = clone;
    }
  }
  return last = cur;
}
//pos is list of nodes to be counted
void cnt_calc(int n, vector<int> &pos){
  for(int x: pos) SA[x].cnt = 1;
  for(int i = n; i>=0; i--){
    for(int x: n_list[i])
      SA[SA[x].link].cnt += SA[x].cnt;
  }
}
```

60 Treap: Implicit

```
typedef struct item * pitem;
struct item {
  int prior, value, cnt; bool rev;
  item(int value):prior(rand()), value(value) {
    cnt = 0; rev = 0; l = r = nullptr;
  }
  pitem l, r;
}; ///Implicit Treap. Use 0-based indexing
namespace Treap {
  int cnt (pitem it) {
    return it != nullptr? it->cnt : 0;
  }
  void upd_cnt (pitem it) {
    if (it != nullptr)
      it->cnt = cnt(it->l)+cnt(it->r)+1;
  }
  void push (pitem it) {
    if (it != nullptr && it->rev == true) {
      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==nullptr || r==nullptr)
      t = (l!=nullptr) ? 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==nullptr) {
      l = r = nullptr;  return;
    }
    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 insert (pitem & t, int key, int value) {
    pitem x = new item(value);
    pitem L, R; split(t, L, R, key);
    merge(L, L, x); merge(t, L, R); upd_cnt(t);
  }
  void erase (pitem & t, int key) {
    assert(cnt(t) > key);   pitem L, MID, R;
    split(t, L, MID, key); split(MID, MID, R, 1);
    merge(t, L, R); delete MID; upd_cnt(t);
  }
  void cyclicShift(pitem &t, int l, int r) {
    assert(0 <= l && r < cnt(t));
    pitem L, MID, R;  split(t, L, MID, r);
    split(MID, MID, R, 1);  merge(t, L, R);
    assert(MID!=nullptr);insert(t, l, MID->value);
    delete MID;  upd_cnt(t);
  }
  void output (pitem t, vector< int >&v) {
    if (t==nullptr) return;
    push (t);               output(t->l,v);
    v.push_back(t->value);  output(t->r,v);
  }
}
///declare root : pitem tr = nullptr;
///then use like this: Treap::insert(tr, i, x);
///or like this: Treap::reverse(tr, l, r);
```

61 Trie

```
const int M_NODE = 1e5 + 7, M_KIDS = 26, root = 0;
int kids[M_NODE][M_KIDS];  // child, -1 first
bool ended[M_NODE];    // string ended here?
int size;  // struct Trie, static const int - ln 1
void init() {
  memset(kids[root], -1, sizeof kids[root]);
  ended[root] = false; size = 1;
}
int ctoi(char c) { return c-'a'; }
void insert(string s) {
  int cur = root;
  for(int i=0; i<(int) s.size(); ++i) {
```

```
    int c = ctoi(s[i]);
    if(kids[cur][c] == -1) {
      memset(kids[size], -1, sizeof kids[size]);
      ended[size] = false;kids[cur][c] = size++;
    }
    cur = kids[cur][c];
  }
  ended[cur] = true;
}
bool query(string s) {
  int cur = root;
  for(int i=0; i<(int) s.size(); ++i) {
    int c = ctoi(s[i]);
    if(kids[cur][c] == -1) return false;
    cur = kids[cur][c];
  }
  return ended[cur];
}  // true if s exists in trie
```

62 Two-SAT, SCC

```
//1 based Index. call init(n) b4 each test case
//col -> sccNo, ok -> value assignment
//For SCC: G-orgnl grph,R-Rvrs grph,top-toplogical
struct TwoSAT{
  int n,nn; static const int N = 1e5+7; // O(N)
  vector<int> G[2*N] , R[2*N], top;
  int col[2*N],vis[2*N],ok[2*N];
  void init(int n) {
    this->n=n; this->nn = n+n; top.clear();
    for(int i=0;i<=nn;i++) G[i].clear(),
      R[i].clear(), col[i]=0, vis[i]=0, ok[i]=0;
  }
  int inv(int no) { return (no <= n) ? no+n:no-n;}
  void add(int u,int v) {
    G[u].push_back(v);   R[v].push_back(u);
  }
  void OR(int u,int v){
    add(inv(u), v);    add(inv(v), u);
  }
  void AND(int u,int v){
    add(u,v);          add(v,u);
  }
  void XOR(int u,int v){
    add(inv(v), u);    add(u, inv(v));
    add(inv(u), v);    add(v, inv(u));
  }
  void XNOR(int u,int v){
    add(u,v);  add(v,u);
    add(inv(u), inv(v));  add(inv(v), inv(u));
  }
  void force_true(int x)   { add(inv(x),x); }
  void force_false(int x)  { add(x,inv(x)); }
  void dfs(int u) {
    vis[u] = 1;
    for(int i = 0; i < G[u].size(); i++) {
      int v = G[u][i];   if(vis[v] == 0 ) dfs(v);
    }
    top.push_back(u);
  }
  void dfs1(int u,int color) {
    col[u] = color;
    if(u <= n) ok[u] = 1;
    else ok[u - n] = 0;
    for(int i = 0 ;i < R[u].size(); i ++ ) {
      if(col[R[u][i]] == 0) dfs1(R[u][i], color);
    }
  }
  void FindScc() {
    for(int i = 1; i <= nn ; i ++ ) {
      if(vis[i] == 0) dfs(i);
    }
    int color = 0; reverse(top.begin(),top.end());
    for(auto u: top) {
      if(col[u] == 0) dfs1(u, ++color);
    }
  }
  void solve() {
    FindScc();
    for(int i=1;i<=n;i++){
      if(col[i] == col[n + i]) {
        printf("Impossible\n");   return;
      }
    }
    vector<int>v;
    for(int i = 1; i <= n; i ++ ) {
        if(ok[i]) v.push_back(i); // All 1'
    }
    cout << v.size() << endl;
    for(auto x: v) printf("%d ",x); puts("");
  }
};
```

63 Voronoi Diagram

```
typedef long double T;  // O(n^2 lg n)
const T EPS = 1e-9, INF = 1e7;
int dcmp(T x) {
  if(fabs(x) < EPS) return 0;
  return x < 0 ? -1 : 1;
} // INSERT PT CLASS WITH +-*/<==,dot,cross,dist2
PT RotateCCW90(PT p) { return PT(-p.y, p.x); }
T ccw(PT p, PT q, PT r) {
  return cross((q-p), (r-q));
}
struct line{  // ax + by = c
  T a, b, c; PT u, d;
  line(T a, T b,T c):a(a), b(b), c(c) { }
  // careful that u, d is not updated here.
  line(PT u_, PT d_) {
    u = u_, d = d_; //CCW direction is the region
    a = d.y, b = -d.x, c = -u.y*d.x + u.x*d.y;
  }  // ax + by <= c --- up ^
  bool operator < (const line &l)const{
    bool flag1 = make_pair(a, b) >
                          make_pair((T) 0, (T) 0);
    bool flag2 = make_pair(l.a, l.b) >
                          make_pair((T) 0, (T) 0);
    if(flag1 != flag2) return flag1 > flag2;
    T t = ccw(PT(0, 0), PT(a, b), PT(l.a, l.b));
    return dcmp(t) == 0 ? c*hypot(l.a, l.b) <
                           l.c * hypot(a, b):t>0;
  }
  PT slope() { return PT(a, b); }
};
PT cross(line a, line b){
  T det = a.a * b.b - b.a * a.b;
  return PT((a.c * b.b - a.b * b.c) / det,
                 (a.a * b.c - a.c * b.a) / det);
}
bool bad(line a, line b, line c){
  if(ccw(PT(0, 0), a.slope(), b.slope()) <= 0)
    return false;
  PT crs = cross(a, b);
  return crs.x * c.a + crs.y * c.b >= c.c;
} // ax + by <= c; -- down VVV
bool hpi(vector<line> v, vector<PT> &solution){
  sort(v.begin(), v.end());
  deque<line> dq;
  for(auto &i : v) {
    if(!dq.empty() && !dcmp(ccw(PT(0, 0),
      dq.back().slope(), i.slope()))) continue;
    while(dq.size() >= 2 && bad(dq[dq.size()-2],
      dq.back(), i)) dq.pop_back();
    while(dq.size() >= 2 && bad(i,dq[0],dq[1]))
```

```
      dq.pop_front();
    dq.push_back(i);
  }
  while(dq.size() > 2 && bad(dq[dq.size()-2],
    dq.back(), dq[0])) dq.pop_back();
  while(dq.size() > 2 && bad(dq.back(), dq[0],
    dq[1])) dq.pop_front();
  vector<PT> tmp;
  for(int i=0; i< dq.size(); i++){
    line cur = dq[i], nxt = dq[(i+1)%dq.size()];
    if(ccw(PT(0, 0), cur.slope(), nxt.slope()) <=
      EPS) return false;
    tmp.push_back(cross(cur, nxt));
  }
  solution = tmp; return true;
}
// returns set of of polygons for each site in P
// bounded by (-inf, -inf) - (inf, inf) rectangle
vector< vector<PT> > VoronoiDiagram(
  const vector<PT> &P) {
  int n = (int) P.size();
  vector<vector<PT>> voronoi_diagram;
  for(int i = 0; i < n; i++) {
    vector<line> lines;
    lines.push_back(line(1,0,INF)); // x <= INF
    lines.push_back(line(-1,0,INF)); // x >= -INF
    lines.push_back(line(0,1,INF)); // y <= INF
    lines.push_back(line(0,-1,INF)); // y >= -INF
    for(int j = 0; j < n; j++) {
      if(P[i] == P[j]) continue;
      PT u = (P[i]+P[j])*0.5;
      PT dir = P[j]-P[i];
      PT dir_90 = RotateCCW90(dir);
      PT v = u + dir_90;
      T a = dir_90.y, b = -dir_90.x;
      T c = -u.y*dir_90.x + u.x*dir_90.y;
      lines.push_back(line(a,b,c));
    }
    vector<PT> polygon; hpi(lines, polygon);
    voronoi_diagram.push_back(polygon);
  }
  return voronoi_diagram;
} // Timus: Empire Strikes Back
```

64 Z Algorithm

```
// z[i] is the length of the longest common prefix
// between s and the suffix of s starting at i.
vector<int> z_function(string s) {
  int n=s.size(); vector<int> z(n); int l=0, r=0;
  for (int i=1; i<n; i++) {
    if (i<=r) z[i] = min(r-i+1, z[i-l]);
    while (i+z[i]<n && s[i+z[i]]==s[z[i]]) z[i]++;
    if (i+z[i]-1>r) l = i, r = i+z[i]-1;
  }
  return z;
}
```

65 Cheat Sheet

65.1 Fibonacci

$$\sum_{i=1}^{n} F_i = F_{n+2} - 1$$

$$\sum_{i=1}^{n} F_i^2 = F_n F_{n+1}$$

$$F_{n+m} = F_n F_{m+1} + F_{n-1} F_m$$

$$F_{2n} = F_n(F_{n+1} + F_{n-1})$$

$$F_{2n+1} = F_n^2 + F_{n+1}^2$$

65.2 Binomial

$$\binom{n}{r} + \binom{n}{r+1} = \binom{n+1}{r+1}$$

$$\binom{n}{r}\binom{r}{k} = \binom{n}{k}\binom{n-k}{r-k}$$

$$\binom{n}{r} = (-1)^r \binom{r-1-n}{r}, (n < 0)$$

65.3 Stirling

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n$$

$$\left\{ {n+1 \atop k} \right\} = k \left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$$

$$\sum_{i=0}^{n} \left\{ {n \atop i} \right\} = B_n$$

65.4 Catalan Number

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

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

65.5 Generating Function

$$\frac{1}{1-ax} = 1 + ax + a^2x^2 + a^3x^3 + \cdots$$

$$\frac{x}{(1-x)^2} = 0 + x + 2x^2 + 3x^3 + \cdots$$

65.6 Euler Number

$$\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle$$

$$x^n = \sum_{k=0}^{n} \left\langle {n \atop k} \right\rangle \binom{x+k}{n}$$

$$\left\langle {n \atop m} \right\rangle = \sum_{k=0}^{m} \binom{n+1}{k} (m+1-k)^n (-1)^k$$