Contents

Sublime Build

```
{
"cmd" : ["g++ -std=c++14 $file_name -o
    $file_base_name && timeout 4s ./
    $file_base_name<inputf.in>outputf.in"],
"selector" : "source.cpp",
"file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)
    ?:? (.*)$",
"shell": true,
"working_dir" : "$file_path"
}
```

1 All Macros

```
//#pragma GCC optimize("Ofast")
//#pragma GCC optimization ("O3")
//#pragma comment(linker, "/stack
    :200000000")
//#pragma GCC optimize("unroll-loops")
//#pragma GCC target("sse,sse2,sse3,ssse3,
    sse4,popcnt,abm,mmx,avx,tune=native")
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
   //find_by_order(k) --> returns iterator
       to the kth largest element counting
        from 0
   //order_of_key(val) --> returns the
       number of items in a set that are
       strictly smaller than our item
template <typename DT>
using ordered_set = tree <DT, null_type,
    less<DT>, rb_tree_tag,
    tree_order_statistics_node_update>;

int kx[] =
    {-2,-2,-1,+1,+2,+2,+1,-1};
int ky[] =
    {-1,+1,+2,+2,+1,-1,-2,-2};

#define fastio         ios_base::
    sync_with_stdio(0);cin.tie(0);

#define Make(x,p)     (x | (1<<p))
#define DeMake(x,p)   (x & ~(1<<p))
#define Check(x,p)    (x & (1<<p))

#define DEBUG(x)      cerr << #x << " = " <<
     x << endl
```

2 DP

2.1 Convex Hull Trick

```
 struct line {
   ll m, c;
   line() {}
   line(ll m, ll c) : m(m), c(c) {}
 };
 struct convex_hull_trick {
   vector<line>lines;
   int ptr = 0;
   convex_hull_trick() {}
   bool bad(line a, line b, line c) {
     return 1.0 * (c.c - a.c) * (a.m - b.m)
         < 1.0 * (b.c - a.c) * (a.m - c.m);
   }
   void add(line L) {
     int sz = lines.size();
     while (sz >= 2 && bad(lines[sz - 2],
         lines[sz - 1], L)) {
       lines.pop_back(); sz--;
     }
     lines.pb(L);
   }
   ll get(int idx, int x) {
     return (1ll * lines[idx].m * x + lines[
         idx].c);
   }
   ll query(int x) {
     if (lines.empty()) return 0;
     if (ptr >= lines.size()) ptr = lines.
         size() - 1;
     while (ptr < lines.size() - 1 && get(
         ptr, x) > get(ptr + 1, x)) ptr++;
     return get(ptr, x);
   }
 };
ll sum[MAX];
ll dp[MAX];
int arr[MAX];
int main() {
  fastio;
  int t;
  cin >> t;
  while (t--) {
    int n, a, b, c;
    cin >> n >> a >> b >> c;
    for (int i = 1; i <= n; i++) cin >> sum
        [i];
    for (int i = 1; i <= n; i++) dp[i] = 0,
         sum[i] += sum[i - 1];
    convex_hull_trick cht;
    cht.add( line(0, 0) );
    for (int pos = 1; pos <= n; pos++) {
      dp[pos] = cht.query(sum[pos]) - 1ll *
          a * sqr(sum[pos]) - c;
      cht.add( line(2ll * a * sum[pos], dp[
          pos] - a * sqr(sum[pos])) );
    }
    ll ans = (-1ll * dp[n]);
    ans += (1ll * sum[n] * b);
    cout << ans << "\n";
  }
}
```

2.2 Divide and Conquer DP

```
inline void compute(int cur, int L, int R,
     int best_L, int best_R) {
  if (L > R) return;
  int mid = (L + R) >> 1;
  pair<ll, int>best = {inf, -1};
  for (int k = best_L; k <= min(best_R, mid)
      ; k++) {
    best = min(best, {dp[cur ^ 1][k - 1] +
        getCost(k, mid), k});
  }
  dp[cur][mid] = best.ff;
  int best_id = best.ss;
  compute(cur, L, mid - 1, best_L, best_id);
  compute(cur, mid + 1, R, best_id, best_R);
}
// in main
int cur = 0;
for (int i = 1; i <= n; i++) dp[1][i] = inf;
for (int guard = 1; guard <= g; guard++) {
  compute(cur, 1, n, 1, n); cur ^= 1;
}
ll ans = dp[cur ^ 1][n];
```

2.3 Knuth Iterative

```
for (int i = 1; i <= n; i++) {
  path[i][i] = i;
  dp[i][i] = 0;
}

for (int len = 2; len <= n; len++) {
  for (int st = 1; st + len - 1 <= n; st++)
      {
    int ed = st + len - 1;
    int L = max(st, path[st][ed - 1]);
    int R = min(ed - 1, path[st + 1][ed]);
    dp[st][ed] = INT_MAX;
    for (int i = L; i <= R; i++) {
      int cur = dp[st][i] + dp[i + 1][ed] +
          arr[ed] - arr[st - 1];
      if (dp[st][ed] > cur) {
        dp[st][ed] = cur;
        path[st][ed] = i;
      }
    }
  }
}
cout << dp[1][n] << "\n";
```

2.4 Knuth Optimization

```
ll solve(int st, int ed) { ///recursive
  if (st == ed) {
    path[st][ed] = st;
    return 0;
  }
  ll &ret = dp[st][ed];
  if (ret != -1) return ret;
  solve(st, ed - 1); solve(st + 1, ed);
  int L = max(st, path[st][ed - 1]);
  int R = min(ed - 1, path[st + 1][ed]);
  ret = inf;
  for (int i = L; i <= R; i++) {
    ll cur = solve(st, i) + solve(i + 1, ed)
        ;
    cur += (arr[ed] - arr[st - 1]);
    if (cur < ret) ret = cur; path[st][ed] =
         i;
  }
  return ret;
}
///knuth for divide and conquer
int solve(int group, int pos) {
  if (!pos) return dp[group][pos] = 0;
  if (!group) return dp[group][pos] =
      INT_MAX;
  int &ret = dp[group][pos];
  if (ret != -1) return ret;
  int L = 1, R = pos;
  if (pos - 1 > 0) {
    solve(group, pos - 1);
    L = max(L, path[group][pos - 1]);
  }
  if (group + 1 <= m) {
    solve(group + 1, pos);
    R = min(R, path[group + 1][pos]);
  }
  ret = INT_MAX;
  for (int i = L; i <= R; i++) {
    int cur = solve(group - 1, i - 1) + 1ll
        * (arr[pos] - arr[i]) * (arr[pos] -
        arr[i]);
    if (cur < ret) {
      ret = cur;
      path[group][pos] = i;
    }
  }
  return ret;
}
```

2.5 Li Chao Tree

```
struct line {
  ll m, c;
  line(ll m = 0, ll c = 0) : m(m), c(c) {}
};
ll calc(line L, ll x) {
  return 1ll * L.m * x + L.c;
}
struct node {
  ll m, c;
  line L;
  node *lft, *rt;
  node(ll m = 0, ll c = 0, node *lft = NULL,
       node *rt = NULL) : L(line(m, c)),
      lft(lft), rt(rt) {}
};
struct LiChao {
  node *root;
  LiChao() {
    root = new node();
  }
  void update(node *now, int L, int R, line
      newline) {
    int mid = L + (R - L) / 2;
    line lo = now->L, hi = newline;
    if (calc(lo, L) > calc(hi, L)) swap(lo,
        hi);
    if (calc(lo, R) <= calc(hi, R)) {
      now->L = hi;
      return;
    }
    if (calc(lo, mid) < calc(hi, mid)) {
      now->L = hi;
```

```
      if (now->rt == NULL) now->rt = new
          node();
      update(now->rt, mid + 1, R, lo);
    } else {
      now->L = lo;
      if (now->lft == NULL) now->lft = new
          node();
      update(now->lft, L, mid, hi);
    }
  }
  ll query(node *now, int L, int R, ll x) {
    if (now == NULL) return -inf;
    int mid = L + (R - L) / 2;
    if (x <= mid) return max( calc(now->L, x
        ), query(now->lft, L, mid, x) );
    else return max( calc(now->L, x), query(
        now->rt, mid + 1, R, x) );
  }
};
```

2.6 Number Permutation

```
ll dp[2][3005]; ll sum[2][3005];
int dir[3005];
int arr[MAX];
int main() {
  fastio;
  int n;
  string s;
  cin >> n >> s;
  s = '#' + s;
  s.pb('<'); ///last element less than the
      element placed after it
  sum[1][0] = 1;
  int cur = 0;
  for (int baki = 1; baki <= n; baki++) {
    if (s[baki] == '<') dp[cur][0] = 0;
    else dp[cur][0] = sum[cur ^ 1][baki -
        1];
    for (int small = 1; small <= baki; small
        ++) {
      if (s[baki] == '<') dp[cur][small] =
          sum[cur ^ 1][small - 1];
      else {
        int big = baki - small;
        dp[cur][small] = sum[cur ^ 1][small
            + big - 1];
        dp[cur][small] -= sum[cur ^ 1][small
            - 1];
        if (dp[cur][small] < 0) dp[cur][
            small] += MOD;
      }
    }
    sum[cur][0] = dp[cur][0];
    for (int small = 1; small <= baki; small
        ++) {
      sum[cur][small] = (sum[cur][small - 1]
          + dp[cur][small]);
      if (sum[cur][small] >= MOD) sum[cur][
          small] -= MOD;
    }
    cur ^= 1;
  }
  ll ans = dp[cur ^ 1][n];
  cout << ans << "\n";
}
```

2.7 Same Color Group

```
int prv[21]; ll cost[21][21];
ll dp[1 << 21]; int m, n;
bool ok[1 << 21];
ll solve(ll mask) {
  if (mask == (1 << m) - 1) return 0ll;
  ll &ret = dp[mask];
  if (ok[mask]) return ret;
  ok[mask] = true; ret = inf;
  for (int i = 0; i < m; i++) {
    if (!(mask & (1 << i) )) {
      ll c = 0;
      for (int j = 0; j < m; j++) {
        if ((mask & (1 << j)))
          c += cost[i][j];
      }
      ret = min(ret, c + solve((mask | (1 <<
          i))));
    }
  }
  return ret;
}
int arr[MAX];
int main() {
  for (int i = 0; i < n; i++) {
    int val = arr[i];
    val--; prv[val]++;
    for (int j = 0; j < m; j++) {
      if (val == j) continue;
      cost[val][j] += prv[j];
    }
  }
  ll ans = solve(0);
}
```

2.8 Sum of Subsets

```
 //submask == all i such that mask&i == i ||
       mask&i == mask (all i such that all 0
       in mask are fixed and the 1's change)
 //sos dp memory optimized
 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)]; /// doing -= can work
          like inclusion-exclusion on unset
          bits
   }
 }
```

2.9 Triangulation DP

```
bool valid[205][205];
ll dp[205][205];
ll solve(int L, int R) {
  if (L + 1 == R) return 1;
  if (dp[L][R] != -1) return dp[L][R];
  ll ret = 0;
  for (int mid = L + 1; mid < R; mid++) {
    if (valid[L][mid] && valid[mid][R]) {
      ///selecting triangle(P[L], P[mid], P[
          R])
      ll temp = ( solve(L, mid) * solve(mid,
          R) ) % MOD;
      ret = (ret + temp) % MOD;
    }
  }
  return dp[L][R] = ret;
}
```

3 Data Structures

3.1 DSU on Tree

```
///Query: Number of distinct names among all
    the k'th son of a node.
const int N = 100005;
string name[N];
vector<int>G[N];
vector<pii>Q[N];
int L[N],ans[N];

void dfs(int v,int d){
    L[v]=d;
    for(int i:G[v]) dfs(i,d+1);
    return;
}

void dsu(int v,map<int,set<string>>&mp){
    for(int i:G[v]){
        map<int,set<string>>s;
        dsu(i,s);
        if(s.size()>mp.size()) swap(mp,s);
        for(auto it:s) mp[it.ff].insert(all(
            it.ss));
    }
    if(v!=0) mp[L[v]].insert(name[v]); //
        Here zero is not a actual node
    for(pii p:Q[v]) ans[p.ss] = mp[p.ff].
        size();
    return;
}

int main(){
    int n;
    cin >> n;
    FOR(i,1,n){
        int u;
        cin >> name[i] >> u;
        G[u].pb(i);
    }
    dfs(0,0);
    int q;
    cin >>q;
    FOR(i,1,q){
        int v,k;
        cin >> v >> k;
        Q[v].pb(pii(k+L[v],i)); //Actual
            level
    }
    map<int,set<string>>mp;
    dsu(0,mp);
    FOR(i,1,q) cout << ans[i] << '\n';
    return 0;
}
```

3.2 Dominator Tree

```
struct dominator {
  int n, d_t;
  vector<vector<int>> g, rg, tree, bucket;
  vector<int> sdom, dom, par, dsu, label,
      val, rev;
  dominator() {}
  dominator(int n) :
    n(n), d_t(0), g(n + 1), rg(n + 1),
    tree(n + 1), bucket(n + 1), sdom(n + 1),
    dom(n + 1), par(n + 1), dsu(n + 1),
    label(n + 1), val(n + 1), rev(n + 1)
  { for (int i = 1; i <= n; i++) sdom[i] =
      dom[i] = dsu[i] = label[i] = i; }

  void add_edge(int u, int v) { g[u].pb(v);
      }
  int dfs(int u) {
    d_t++;
    val[u] = d_t, rev[d_t] = u;
    label[d_t] = sdom[d_t] = dom[d_t] = d_t;
    for (int v : g[u]) {
      if (!val[v]) {
        dfs(v);
        par[val[v]] = val[u];
      }
      rg[val[v]].pb(val[u]);
    }
  }
  int findpar(int u, int x = 0) {
    if (dsu[u] == u) return x ? -1 : u;
    int v = findpar(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 join(int u, int v) { dsu[v] = u; }
  vector<vector<int>> build(int s) {
    dfs(s);
    for (int i = n; i >= 1; i--) {
      for (int j = 0; j < rg[i].size(); j++)
          {
        sdom[i] = min(sdom[i], sdom[ findpar
            (rg[i][j]) ]);
      }
      if (i > 1) bucket[sdom[i]].pb(i);
      for (int w : bucket[i]) {
        int v = findpar(w);
        if (sdom[v] == sdom[w]) dom[w] =
            sdom[w];
        else dom[w] = v;
      }
      if (i > 1) join(par[i], i);
```

```cpp
    }
    for (int i = 2; i <= n; i++) {
      if (dom[i] != sdom[i]) dom[i] = dom[
          dom[i]];
      tree[rev[i]].pb(rev[dom[i]]);
      tree[rev[dom[i]]].pb(rev[i]);
    }
    return tree;
  }
};
```

3.3 Hopcroft Karp

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

const int N = 3e5 + 9;

struct HopcroftKarp {
  static const int inf = 1e9;
  int n;
  vector<int> l, r, d;
  vector<vector<int>> g;
  HopcroftKarp(int _n, int _m) {
    n = _n;
    int p = _n + _m + 1;
    g.resize(p);
    l.resize(p, 0);
    r.resize(p, 0);
    d.resize(p, 0);
  }
  void add_edge(int u, int v) {
    g[u].push_back(v + n); //right id is
        increased by n, so is l[u]
  }
  bool bfs() {
    queue<int> q;
    for (int u = 1; u <= n; u++) {
      if (!l[u]) d[u] = 0, q.push(u);
      else d[u] = inf;
    }
    d[0] = inf;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto v : g[u]) {
        if (d[r[v]] == inf) {
          d[r[v]] = d[u] + 1;
          q.push(r[v]);
        }
      }
    }
    return d[0] != inf;
  }
  bool dfs(int u) {
    if (!u) return true;
    for (auto v : g[u]) {
      if(d[r[v]] == d[u] + 1 && dfs(r[v])) {
        l[u] = v;
        r[v] = u;
        return true;
      }
    }
    d[u] = inf;
    return false;
  }
  int maximum_matching() {
    int ans = 0;
    while (bfs()) {
      for(int u = 1; u <= n; u++) if (!l[u]
          && dfs(u)) ans++;
    }
    return ans;
  }
};
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m, q;
  cin >> n >> m >> q;
  HopcroftKarp M(n, m);
  while (q--) {
    int u, v;
    cin >> u >> v;
    M.add_edge(u, v);
  }
  cout << M.maximum_matching() << '\n';
  return 0;
}
```

3.4 Implicit Segment Tree

```cpp
struct node {
  int val;
  node *lft, *rt;
  node() {}
  node(int val = 0) : val(val), lft(NULL),
      rt(NULL) {}
};

struct implicit_segtree {
  node *root;
  implicit_segtree() {}
  implicit_segtree(int n) {
    root = new node(n);
  }
  void update(node *now, int L, int R, int
      idx, int val) {
    if (L == R) {
      now -> val += val;
      return;
    }
    int mid = L + (R - L) / 2;
    if (now->lft == NULL) now->lft = new
        node(mid - L + 1);
    if (now->rt == NULL) now->rt = new node(
        R - mid);
    if (idx <= mid) update(now->lft, L, mid,
         idx, val);
    else update(now->rt, mid + 1, R, idx,
        val);
    now->val = (now->lft)->val + (now->rt)->
        val;
  }

  int query(node *now, int L, int R, int k)
      {
    if (L == R) return L;
    int mid = L + (R - L) / 2;
    if (now->lft == NULL) now->lft = new
        node(mid - L + 1);
    if (now->rt == NULL) now->rt = new node(
        R - mid);
    if (k <= (now->lft)->val) return query(
        now->lft, L, mid, k);
    else return query(now->rt, mid + 1, R, k
         - (now->lft)->val);
  }
};
```

3.5 Implicit Treap

```cpp
mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());
typedef struct node* pnode;
struct node {
  int prior, sz;
  ll val, sum, lazy;
  bool rev;
  node *lft, *rt;
  node(int val = 0, node *lft = NULL, node *
      rt = NULL) : lft(lft), rt(rt), prior(
      rnd()), sz(1), val(val), rev(false),
      sum(0), lazy(0) {}
};
struct implicit_treap {
  pnode root;
  implicit_treap() {
    root = NULL;
  }
  int get_sz(pnode now) {
    return now ? now->sz : 0;
  }
  void update_sz(pnode now) {
    if (!now) return;
    now->sz = 1 + get_sz(now->lft) + get_sz(
        now->rt);
  }
  // lazy sum
  void push(pnode now) {
    if (!now || !now->lazy) return;
    now->val += now->lazy;
    now->sum += get_sz(now) * now->lazy;
    if (now->lft) now->lft->lazy += now->
        lazy;
    if (now->rt) now->rt->lazy += now->lazy;
    now->lazy = 0;
  }
  void combine(pnode now) {
    if (!now) return;
    now->sum = now->val; // reset the node
    push(now->lft), push(now->rt); // update
         lft and rt
    now->sum += (now->lft ? now->lft->sum :
        0) + (now->rt ? now->rt->sum : 0);
  }
  // reverse substring
  // void push(pnode now) {
  //   if (!now || !now->rev) return;
  //   now->rev = false;
  //   swap(now->lft, now->rt);
  //   if (now->lft) now->lft->rev ^= true;
  //   if (now->rt) now->rt->rev ^= true;
  // }
  // sort ascending or descending
  // void push(pnode now) {
  //   if (!now || !now->sort_kor) return;
  //   if (now->sort_kor == -1) swap(now->
      lft, now->rt);
  //   int cnt[26];
  //   for (int i = 0; i < 26; i++) cnt[i] =
       now->cnt[i];
  //   int idx = 0;
  //   if (now->lft) {
  //     memset(now->lft->cnt, 0, sizeof now
      ->lft->cnt);
  //     int lft_sz = get_sz(now->lft);
  //     while (idx < 26 && lft_sz) {
  //       int mn = min(cnt[idx], lft_sz);
  //       now->lft->cnt[idx] = mn;
  //       cnt[idx] -= mn; lft_sz -= mn;
  //       if (!cnt[idx]) idx++;
  //     }
  //     now->lft->sort_kor = now->sort_kor;
  //   }
  //   while (!cnt[idx]) idx++;
  //   now->val = idx, cnt[idx]--;
  //   if (!cnt[idx]) idx++;
  //   if (now->rt) {
  //     memset(now->rt->cnt, 0, sizeof now
      ->rt->cnt);
  //     int rt_sz = get_sz(now->rt);
  //     while (idx < 26 && rt_sz) {
  //       int mn = min(cnt[idx], rt_sz);
  //       now->rt->cnt[idx] = mn;
  //       cnt[idx] -= mn; rt_sz -= mn;
  //       if (!cnt[idx]) idx++;
  //     }
  //     now->rt->sort_kor = now->sort_kor;
  //   }
  //   if (now->sort_kor == -1) swap(now->
      lft, now->rt);
  //   now->sort_kor = 0;
  // }
  // void combine(pnode now) {
  //   if (!now) return;
  //   memset(now->cnt, 0, sizeof now->cnt);
  //   for (int i = 0; i < 26; i++) {
  //     now->cnt[i] = (now->lft ? now->lft
      ->cnt[i] : 0) + (now->rt ? now->rt->
      cnt[i] : 0);
  //   }
  //   now->cnt[now->val]++;
  // }
  ///first pos ta elements go to left,
      others go to right
  void split(pnode now, pnode &lft, pnode &
      rt, int pos, int add = 0) {
    if (!now) return void(lft = rt = NULL);
    push(now);
    int cur = add + get_sz(now->lft);
    if (cur < pos) split(now->rt, now->rt,
        rt, pos, cur + 1), lft = now;
```

```
    else split(now->lft, lft, now->lft, pos,
        add), rt = now;
    update_sz(now); combine(now);
  }
  void merge(pnode &now, pnode lft, pnode rt
      ) {
    push(lft);
    push(rt);
    if (!lft || !rt) now = lft ? lft : rt;
    else if (lft->prior > rt->prior) merge(
        lft->rt, lft->rt, rt), now = lft;
    else merge(rt->lft, lft, rt->lft), now =
        rt;
    update_sz(now); combine(now);
  }
  void insert(int pos, ll val) {
    if (!root) return void(root = new node(
        val));
    pnode lft, rt;
    split(root, lft, rt, pos - 1);
    pnode notun = new node(val);
    merge(root, lft, notun);
    merge(root, root, rt);
  }
  void erase(int pos) {
    pnode lft, rt, temp;
    split(root, lft, rt, pos);
    split(lft, lft, temp, pos - 1);
    merge(root, lft, rt);
    delete(temp);
  }
  void reverse(int l, int r) {
    pnode lft, rt, mid;
    split(root, lft, mid, l - 1);
    split(mid, mid, rt, r - l + 1);
    mid->rev ^= true;
    merge(root, lft, mid);
    merge(root, root, rt);
  }
  void right_shift(int l, int r) {
    pnode lft, rt, mid, last;
    split(root, lft, mid, l - 1);
    split(mid, mid, rt, r - l + 1);
    split(mid, mid, last, r - l);
    merge(mid, last, mid);
    merge(root, lft, mid);
    merge(root, root, rt);
  }
  void output(pnode now, vector<int>&v) {
    if (!now) return;
    push(now);
    output(now->lft, v);
    v.pb(now->val);
    output(now->rt, v);
  }
  vector<int>get_arr() {
    vector<int>ret;
    output(root, ret);
    return ret;
  }
};
```

3.6 LCA

```
/*Hey, What's up?*/

#include<bits/stdc++.h>
using namespace std;
#define pi acos(-1.0)
#define fastio ios_base::sync_with_stdio(
    false);cin.tie(NULL);cout.tie(NULL)
vector<long long>v[100005],vc;
long long x[200005][40],mp[100005],ml
    [100005],nd,pos[100005];
void build(long long n)
{
    long long a,i,j,k,b,c;
    a=1;
    for(i=0; i<n; i++)
    {
        x[i][0]=vc[i];
    }
    b=1;
    while(a<n)
    {
        for(i=0; i<n-a; i++)
        {
            x[i][b]=min(x[i][b-1],x[i+a][b
                -1]);
        }
        a*=2;
        b++;
    }
    return;
}
long long query(long long a, long long b)
{
    long long c,d,e,f;
    //if(a==b)return x[a][0];
    c=log2(1.0*(b-a+1));
    //cout<<c<<' ';
    f=powl(1.0*2,1.0*c);
    d=x[a][c];
    e=x[b-f+1][c];
    //cout<<b-f+1<<' ';
    return min(d,e);

}
void tour_de_euler(long long p, long long q)
{
    vc.push_back(mp[p]);
    //nd++;
    if(!pos[mp[p]])pos[mp[p]]=nd;
    nd++;
    for(int i=0;i<v[p].size();i++){
        if(v[p][i]==q)continue;
        tour_de_euler(v[p][i],p);
        vc.push_back(mp[p]); nd++;
    }
    return;

}
void dfs(long long p, long long q)
{
    mp[p]=nd;
    ml[nd]=p;
    nd++;
    for(int i=0;i<v[p].size();i++){
        if(v[p][i]==q)continue;
        dfs(v[p][i],p);
    }
    return;

}

long long lca(long long a, long long b)
{
    a=pos[mp[a]];
    b=pos[mp[b]];
    if(a>b)
        swap(a,b);
    long long c=query(a,b);
    return ml[c];
}

int main()
{
    //fastio;
    long long a=0,b=0,c,d,e,f=0,l,g,m,n,k,i,
        j,t,p,q;
    cin>>n;
    for(i=1; i<n; i++)
    {
        cin>>a>>b;
        v[a].push_back(b);
        v[b].push_back(a);
    }
    nd=1;
    dfs(1,-1);
    vc.push_back(696969696969);
    nd=1;
    tour_de_euler(1,-1);
    l=vc.size();
    build(l+2);
    cin>>q;
    while(q--){
        cin>>a>>b;
        cout<<lca(a,b)<<endl;
    }

    return 0;
}
```

3.7 Link Cut Tree

```
struct SplayTree {
  struct node {
    int ch[2] = {0, 0}, p = 0;
    ll self = 0, path = 0;
    ll sub = 0, extra = 0;
    bool rev = false;
  };
  vector<node> T;
  SplayTree(int n) : T(n + 1) {}
  void push(int x) {
    if (!x) return;
    int l = T[x].ch[0], r = T[x].ch[1];
    if (T[x].rev) {
      T[l].rev ^= true, T[r].rev ^= true;
      swap(T[x].ch[0], T[x].ch[1]);
      T[x].rev = false;
    }
  }
  void pull(int x) {
    int l = T[x].ch[0], r = T[x].ch[1];
    push(l), push(r);
    T[x].path = T[x].self + T[l].path + T[r
        ].path;
    T[x].sub = T[x].self + T[x].extra + T[l
        ].sub + T[r].sub;
  }
  void set(int parent, int child, int d) {
    T[parent].ch[d] = child;
    T[child].p = parent;
    pull(parent);
  }
  int dir(int x) {
    int parent = T[x].p;
    if (!parent) return -1;
    return (T[parent].ch[0] == x) ? 0 : (T[
        parent].ch[1] == x) ? 1 : -1;
  }
  void rotate(int x) {
    int parent = T[x].p, gparent = T[parent
        ].p;
    int dx = dir(x), dp = dir(parent);
    set(parent, T[x].ch[!dx], dx);
    set(x, parent, !dx);
    if (~dp) set(gparent, x, dp);
    T[x].p = gparent;
  }
  void splay(int x) {
    push(x);
    while (~dir(x)) {
      int parent = T[x].p;
      int gparent = T[parent].p;
      push(gparent), push(parent), push(x);
      int dx = dir(x), dp = dir(parent);
      if (~dp) rotate(dx != dp ? x : parent)
          ;
      rotate(x);
    }
  }
};
struct LinkCut : SplayTree {
  LinkCut(int n) : SplayTree(n) {}
  void cut_right(int x) {
    splay(x);
    int r = T[x].ch[1];
    T[x].extra += T[r].sub;
    T[x].ch[1] = 0, pull(x);
  }
  int access(int x) {
    int u = x, v = 0;
    for (; u; v = u, u = T[u].p) {
      cut_right(u);
      T[u].extra -= T[v].sub;
      T[u].ch[1] = v, pull(u);
    }
    return splay(x), v;
  }
  void make_root(int x) {
```

```
    access(x);
    T[x].rev ^= true, push(x);
  }
  void link(int u, int v) {
    make_root(v), access(u);
    T[u].extra += T[v].sub;
    T[v].p = u, pull(u);
  }
  void cut(int u) {
    access(u);
    T[u].ch[0] = T[ T[u].ch[0] ].p = 0;
    pull(u);
  }
  void cut(int u, int v) {
    make_root(u), access(v);
    T[v].ch[0] = T[u].p = 0, pull(v);
  }
  int find_root(int u) {
    access(u), push(u);
    while (T[u].ch[0]) {
      u = T[u].ch[0], push(u);
    }
    return splay(u), u;
  }
  int lca(int u, int v) {
    if (u == v) return u;
    access(u);
    int ret = access(v);
    return T[u].p ? ret : 0;
  }
  // subtree query of u if v is the root
  ll subtree(int u, int v) {
    make_root(v), access(u);
    return T[u].self + T[u].extra;
  }
  ll path(int u, int v) {
    make_root(u), access(v);
    return T[v].path;
  }
  // point update
  void update(int u, ll val) {
    access(u);
    T[u].self = val, pull(u);
  }
};
```

3.8 Merge Sort Tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<typename T> using ordered_set =
    tree<T, null_type, less<T>, rb_tree_tag
    , tree_order_statistics_node_update>;

ordered_set<pii> bst[MAX << 2];
void init(int n) {
  for (int i = 0; i <= 4 * n; i++) bst[i].
      clear();
}

void build(int now, int L, int R) {
  if (L == R) {
    bst[now].insert({arr[L], L});
    return;
  }
  for (int i = L; i <= R; i++) bst[now].
      insert({arr[i], i});

  int mid = (L + R) / 2;
  build(now << 1, L, mid);
  build((now << 1) | 1, mid + 1, R);
}

void update(int now, int L, int R, int idx,
    int ager_val, int val) {
  if (L == R) {
    bst[now].erase(bst[now].find({ager_val,
        idx}));
    bst[now].insert({val, idx});
    return;
  }
  int mid = (L + R) / 2;
  if (idx <= mid) update(now << 1, L, mid,
      idx, ager_val, val);
  else update((now << 1) | 1, mid + 1, R,
      idx, ager_val, val);
  bst[now].erase(bst[now].find({ager_val,
      idx}));
  bst[now].insert({val, idx});
}

ll query(int now, int L, int R, int i, int j
    , int val) {
  if (R < i || L > j) return 0;
  if (L >= i && R <= j) {
    int ret = bst[now].order_of_key({val,
        INT_MAX});
    return ret;
  }

  int mid = (L + R) / 2;
  return query(now << 1, L, mid, i, j, val)
      + query((now << 1) | 1, mid + 1, R, i
      , j, val);
}
```

3.9 Mo's Algorithm

```
int vis[1000005];
int y[1000005];
int main()
{
    fastio;
    long long a=0,b=0,c,d,e,f=0,l,g,m,r,n,k,
        i,j,t,p,q;
    cin>>n;
    vector<long long>v;
    d=sqrt(1.0*n);
    vector<pair<long long,pair<long long,
        long long> > >x[d+2];
    //map<pair<long long,long long> ,long
        long>mp;
    v.push_back(-37);
    for(i=0;i<n;i++){
        cin>>a;
        v.push_back(a);
    }
    cin>>q;
    for(i=0;i<q;i++){
        cin>>a>>b;
        e=a/d;
        x[e].push_back({b,{a,i}});
    }
    for(i=0;i<=d+1;i++){
        sort(x[i].begin(),x[i].end());
    }
    for(i=0;i<=d;i++){
        memset(vis,0,sizeof(vis));
        l=i*d+1;
        r=i*d;
        p=x[i].size();
        f=0;
        for(j=0;j<p;j++){
            b=x[i][j].first;
            a=x[i][j].second.first;
            while(r<b){
                r++;
                vis[v[r]]++;
                if(vis[v[r]]==1)f++;
            }
            //cout<<l<<' '<<r<<'='<<f<<endl;
            if(l<a){
            while(l<a){
                vis[v[l]]--;
                if(vis[v[l]]==0)f--;
                l++;
            }}
            else if(l>a){
                while(l>a){
                    l--;
                    vis[v[l]]++;
                    if(vis[v[l]]==1)f++;
                }
            }
            //cout<<l<<' '<<r<<'='<<f<<endl;
                y[x[i][j].second.second]=f;
                //cout<<a<<' '<<b<<'='<<f<<endl;
            }
        }
        for(i=0;i<q;i++){
            cout<<y[i]<<'\n';
        }

        return 0;
}
```

3.10 Persistent Segment Tree

```
struct node {
  int val, lft, rt;
  node(int val = 0, int lft = 0, int rt = 0)
      : val(val), lft(lft), rt(rt) {}
};
node nodes[30 * MAX]; ///take at least 2*n*
    log(n) nodes
int root[MAX], sz;
inline int update(int &now, int L, int R,
    int idx, int val) {
  if (L > idx || R < idx) return now;
  if (L == R) {
    ++sz;
    nodes[sz] = nodes[now];
    nodes[sz].val += val;
    return sz;
  }
  int mid = (L + R) >> 1;
  int ret = ++sz;
  if (idx <= mid) {
    if (!nodes[now].lft) nodes[now].lft = ++
        sz;
    nodes[ret].lft = update(nodes[now].lft,
        L, mid, idx, val);
    nodes[ret].rt = nodes[now].rt;
  } else {
    if (!nodes[now].rt) nodes[now].rt = ++sz
        ;
    nodes[ret].rt = update(nodes[now].rt,
        mid + 1, R, idx, val);
    nodes[ret].lft = nodes[now].lft;
  }
  nodes[ret].val = nodes[ nodes[ret].lft ].
      val + nodes[ nodes[ret].rt ].val;
  return ret;
}
inline int query(int &now, int L, int R, int
    i, int j) {
  if (L > j || R < i) return 0;
  if (L >= i && R <= j) return nodes[now].
      val;
  int mid = (L + R) >> 1;
  return query(nodes[now].lft, L, mid, i, j)
      + query(nodes[now].rt, mid + 1, R, i
      , j);
}
/// in main(make segtree for every prefix)
root[0] = 0;
for (int i = 1; i <= n; i++) root[i] =
    update(root[i - 1], 1, n, p[i], 1);
```

3.11 Sparse Table

```
const int maxn = (1<<20)+5 ;
int logs[maxn] = {0};

void compute_logs(){
    logs[1] = 0;
    for(int i=2;i<(1<<20);i++){
        logs[i] = logs[i/2]+1;
    }
}

class Sparse_Table
{
    public:
        vector <vector<LL>> table;
        function < LL(LL,LL) > func;
        LL identity;

    Sparse_Table(vector <LL> &v, function <
        LL(LL,LL)> _func, LL id){
```

```
    if(logs[2] != 1) compute_logs();
    int sz = v.size();
    table.assign(sz,vector <LL>(logs[sz
        ]+1) );
    func = _func, identity = id;

    for(int j=0;j<=logs[sz];j++){
        for(int i=0;i+(1<<j)<=sz;i++){
            if(j==0) table[i][j] = func(v
                [i],id); // base case,
                when only 1 element in
                range
            else table[i][j] = func(table
                [i][j-1], table[i +
                (1<<(j-1))][j-1] );
        }
    }
  }
  // when intersection of two ranges wont
     be a problem like min, gcd,max
  LL query(int l, int r){
    assert(r>=l);
    int pow = logs[r-l+1];
    return func(table[l][pow], table[r-
        (1<<pow) + 1][pow]);
  }
  // other cases like sum
  LL Query(int l,int r){
    if(l>r) return identity; // handle
        basecase
    int pow = logs[r - l + 1];
    return func(table[l][pow], Query(l
        +(1<<pow), r));
  }
};
```

3.12 SumOfDivisors

```
#include<bits/stdc++.h>
using namespace std;
#define hlw ios_base::sync_with_stdio(false)
    ;cin.tie(NULL);cout.tie(NULL)
long long sod[10000007];
bool pm[10000007];
vector<long long>prm;
int main()
{
    hlw;
    long long i,j,n,t,d,p;
    sod[1]=1;
    prm.push_back(2);
    for(i=3; i<10000007; i+=2)
    {
        if(!pm[i])
        {
            prm.push_back(i);
            for(j=i*i; j<10000007; j+=2*i)
            {
                pm[j]=1;

            }
        }
    }
    for(i=1; i<=10000007;i++)
        sod[i]=1;
    for(i=0; i<prm.size(); i++)
    {
        p=prm[i];
        for(j=p; j<10000007; j+=p)
        {
            d=p;
            while(j%d==0)
                d*=p;
            d--;
            d/=(p-1);
            sod[j]*=d;
        }
    }


    return 0;
}
```

3.13 Treap

```
mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());
typedef struct node* pnode;
struct node {
  int prior, val, sz;
  ll sum;
  node *lft, *rt;
  node(int val = 0, node *lft = NULL, node *
      rt = NULL) :
    lft(lft), rt(rt), prior(rnd()), val(val)
        , sz(1), sum(0) {}
};
struct treap {
  pnode root;
  treap() {
    root = NULL;
  }
  int get_sz(pnode now) {
    return now ? now->sz : 0;
  }
  void update_sz(pnode now) {
    if (!now) return;
    now->sz = 1 + get_sz(now->lft) + get_sz(
        now->rt);
  }
  ll get(pnode now) {
    return now ? now->sum : 0;
  }
  void push(pnode now) {}
  void combine(pnode now) {
    if (!now) return;
    now->sum = now->val + get(now->lft) +
        get(now->rt);
  }
  pnode unite(pnode lft, pnode rt) {
    if (!lft || !rt) return lft ? lft : rt;
    // push(lft), push(rt); this not tested
    if (lft->prior < rt->prior) swap(lft, rt
        );
    pnode l, r;
    split(rt, l, r, lft->val);
    lft->lft = unite(lft->lft, l), update_sz
        (lft);
    lft->rt = unite(lft->rt, r), update_sz(
        lft);
    // combine(lft); this not tested
    return lft;
  }
  ///value < val goes to left, value >= val
     goes to right
  void split(pnode now, pnode &lft, pnode &
      rt, int val, int add = 0) {
    push(now);
    if (!now) return void(lft = rt = NULL);
    if (now->val < val) split(now->rt, now->
        rt, rt, val), lft = now;
    else split(now->lft, lft, now->lft, val)
        , rt = now;
    update_sz(now), combine(now);
  }
  void merge(pnode &now, pnode lft, pnode rt
      ) {
    push(lft), push(rt);
    if (!lft || !rt) now = lft ? lft : rt;
    else if (lft->prior > rt->prior) merge(
        lft->rt, lft->rt, rt), now = lft;
    else merge(rt->lft, lft, rt->lft), now =
         rt;
    update_sz(now), combine(now);
  }
  void insert(pnode &now, pnode notun) {
    if (!now) return void(now = notun);
    push(now);
    if (notun->prior > now->prior) split(now
        , notun->lft, notun->rt, notun->val
        ), now = notun;
    else insert(notun->val < now->val ? now
        ->lft : now->rt, notun);
    update_sz(now), combine(now);
  }
  void erase(pnode &now, int val) {
    push(now);
    if (now->val == val) {
      pnode temp = now;
      merge(now, now->lft, now->rt);
      delete(temp);
    } else erase(val < now->val ? now->lft :
         now->rt, val);
    update_sz(now), combine(now);
  }
  int get_idx(pnode &now, int val) {
    if (!now) return INT_MIN;
    else if (now->val == val) return 1 +
        get_sz(now->lft);
    else if (val < now->val) return get_idx(
        now->lft, val);
    else return (1 + get_sz(now->lft) +
        get_idx(now->rt, val));
  }
  int find_kth(pnode &now, int k) {
    if (k < 1 || k > get_sz(now)) return -1;
    if (get_sz(now->lft) + 1 == k) return
        now->val;
    if (k <= get_sz(now->lft)) return
        find_kth(now->lft, k);
    return find_kth(now->rt, k - get_sz(now
        ->lft) - 1);
  }
  ll prefix_sum(pnode &now, int k) {
    if (k < 1 || k > get_sz(now)) return -
        inf;
    if (get_sz(now->lft) + 1 == k) return
        get(now->lft) + now->val;
    if (k <= get_sz(now->lft)) return
        prefix_sum(now->lft, k);
    return get(now->lft) + now->val +
        prefix_sum(now->rt, k - get_sz(now
        ->lft) - 1);
  }
  pnode get_rng(int l, int r) { ///gets all
      l <= values <= r
    pnode lft, rt, mid;
    split(root, lft, mid, l);
    split(mid, mid, rt, r + 1);
    merge(root, lft, rt);
    return mid;
  }
  void output(pnode now, vector<int>&v) {
    if (!now) return;
    output(now->lft, v);
    v.pb(now->val);
    output(now->rt, v);
  }
  vector<int>get_arr() {
    vector<int>ret;
    output(root, ret);
    return ret;
  }
};
```

3.14 Trie

```
int trie[30 * 100000 + 5][2];
int mark[30 * 100000 + 5];
int node = 1;
void add(int n) {
  int now = 1;
  for (int i = 27; i >= 0; i--) {
    int d = (bool)(n & (1 << i));
    if (!trie[now][d]) trie[now][d] = ++node
        ;
    now = trie[now][d];
    mark[now]++;
  }
}
void del(int n) {
  int now = 1;
  deque<int>v;
  for (int i = 27; i >= 0; i--) {
    int d = (bool)(n & (1 << i));
    if (trie[now][d]) {
      v.push_front(now);
      now = trie[now][d];
      mark[now]--;
    }
  }
  v.push_front(now);
  for (int i = 1; i < v.size(); i++) {
```

```
    if (!mark[v[i - 1]]) {
      if (trie[v[i]][0] == v[i - 1]) trie[v[
          i]][0] = 0;
      if (trie[v[i]][1] == v[i - 1]) trie[v[
          i]][1] = 0;
    }
  }
}
```

4 Geometry

4.1 2D Point

```
const double PI = acos(-1), EPS = 1e-10;
template <typename DT> DT sq(DT x) {return x
     * x; }
template <typename DT> int dcmp(DT x) {
     return fabs(x) < EPS ? 0 : (x<0 ? -1 :
     1);}

template <typename DT>
class point{
    public:
        DT x,y;
    point() = default;
    point(DT x, DT y): x(x), y(y) {};
    template <typename X> point(point <X> p)
        : x(p.x), y(p.y) {};
    //opeartions on complex numbers
    point operator * (point rhs) const {
         return point(x * rhs.x - y * rhs.y,
          x * rhs.y + y * rhs.x);}
    point operator / (point rhs) const {
         return *this * point(rhs.x, - rhs.y
         ) / ~(rhs);}

    bool  operator < (point rhs) const {
         return x < rhs.x or (x == rhs.x and
          y < rhs.y); }
    DT operator & (point rhs)  const {
         return x * rhs.y - y * rhs.x; } //
         cross product
    DT operator ^ (point rhs)  const {
         return x * rhs.x + y * rhs.y; } //
         dot product
    DT operator ~()   const {return sq(x) +
         sq(y); }                  //square
         of norm
    friend istream& operator >> (istream &is
         , point &p) { return is >> p.x >> p
         .y; }

    friend DT DisSq(point a, point b){
         return sq(a.x-b.x) + sq(a.y-b.y); }
    friend DT TriArea(point a, point b,
         point c) { return (b-a) & (c-a); }
    friend DT UTriArea(point a, point b,
         point c) { return abs(TriArea(a, b,
          c)); }
    friend bool Collinear(point a, point b,
         point c) { return UTriArea(a, b, c)
          < EPS; }
    friend double Angle(point u) { return
         atan2(u.y, u.x); }
    friend double Angle(point a, point b) {
        double ans = Angle(b) - Angle(a);
        return ans <= -PI ? ans + 2*PI : (
             ans > PI ? ans - 2*PI : ans);
    }
    friend point Perp(point a){
        return point(-a.y, a.x);
    }
    friend operator Orientation(point a,
         point b, point c) {return dcmp(
         TriArea(a, b, c));}
};
template <typename DT> using polygon =
     vector <point <DT>>;
template <typename DT>
class polarComp {
    point <DT> O, dir;
    bool half(point <DT> p) {
        return dcmp(dir & p) < 0 || (dcmp(
             dir & p) == 0 && dcmp(dir ^ p) >
              0);
    }
    public:
    polarComp(point <DT> O = point(0, 0),
         point <DT> dir = point(1, 0))
        : O(O), dir(dir) {}

    bool operator() (point <DT> p, point <DT
         > q) {
        return make_tuple(half(p), 0) <
             make_tuple(half(q), (p & q));
    }
}; // given a pivot point and an initial
     direction, sorts by Angle with the
     given direction
```

4.2 3D Geo Templates

```
point get_perp(point p){ // returns a random
      perpendicular line to the vector p
    assert(sgn(norm(p)));

    point ret = point(-p.y, p.x, 0);
    if(sgn(norm(ret))) return ret;

    ret = point(0, -p.z, p.y);
    if(sgn(norm(ret))) return ret;

    assert(false)
}

struct plane{ // Caution: directed plane,
     directed on the direction of (p2 x p3)

 point n; // {a, b, c}
 double d; //ax + by + cz = d
 // d = n . p [ where p is any point on the
      plane ]

 plane(){;}
 plane(point _n, double _d){
        n = _n;
        d = _d;
 }
 plane(point p1, point p2, point p3){
  n = crsp(p2 - p1, p3 - p1);
  if(norm(n) < eps) {assert(false);} //
       doesn't define a plance
  d = dotp(p1, n);
 }

   //Preserves the direction
 point get_p1(){ return univ(n) * d / norm(n
      );}
 point get_p2(){ return get_p1() + get_perp(
      n);}
 point get_p3(){ return crsp(n, get_p2() -
      get_p1()) + get_p1();}

 int get_side(point p){ return sgn(dotp(n, p
      ) - d);} ///OK
 double sgn_dist(point p) {return (dotp(n, p
      ) - d) / norm(n);}
 double dist(point p) {return fabs(sgn_dist(
      p));}

 point project(point p){ return p - sgn_dist
      (p) * univ(n);}
 point reflect(point p){ return p - 2 *
      sgn_dist(p) * univ(n);}

   ///OK
   point get_coords(point p){ // "2-D"-fies
        the plane. All points on this
        plane have z = 0
       point O = get_p1();
       point ox = univ(get_p2() - O);
       point oy = univ(get_p3() - O);
       point oz = univ(n);
       p = p - O;
       return {dotp(p, ox), dotp(p, oy),
            dotp(p, oz)};
   }
};

plane translate(plane p, point t) {return {p
     .n, p.d + dotp(p.n, t)};}
plane shiftUp(plane p, double d) {return {p.
     n, p.d + d * norm(p.n)};}


point projection(point p, point st, point ed
     ) { return dotp(ed - st, p - st) / norm
     (ed - st) * univ(ed - st) + st;} //OK
point extend(point st, point ed, double len)
      { return ed + univ(ed-st) * len;} //OK

point rtt(point axis, point p, double theta)
     {
    axis = univ(axis);
    return p * cos(theta) + sin(theta) *
         crsp(axis, p) + axis * (1-cos(theta
         )) * dotp(axis, p);
} //OK

point segmentProjection(point p, point st,
     point ed)
{
    double d = dotp(p - st, ed - st) / norm(
         ed - st);
    if(d < 0) return st;
    if(d > norm(ed - st) + eps) return ed;
    return st + univ(ed - st) * d;
} //OK

double distPointSegment(point p, point st,
     point ed) {return norm(p -
     segmentProjection(p, st, ed)); } //OK
double distPointLine( point P, point st,
     point ed) { return norm( projection(P,
     st, ed) - P ); } //OK

double pointPlanedist(plane P, point q){
     return fabs(dotp(P.n, q) - P.d) / norm(
     P.n);}
double pointPlanedist(point p1, point p2,
     point p3, point q){ return
     pointPlanedist(plane(p1,p2,p3), q); }
     //OK

point reflection(point p, point st, point ed
     ){
    point proj = projection(p, st, ed);
    if(p != proj) return extend(p, proj,
         norm(p - proj));
    return proj;
} //OK

bool coplanar(point p1, point p2, point p3,
     point q)
{
    p2 = p2-p1, p3 = p3-p1, q = q-p1;
    if( fabs( dotp(q, crsp(p2, p3)) ) < eps
         ) return true;
    return false;
}

int linePlaneIntersection(point u, point v,
     point l, point m, point r, point &x){
    /*
        -> l, m, r defines the plane
        -> u, v defines the line
        -> returns 0 when does not intersect
        -> returns 1 when there exists one
             unique common point
        -> returns -1 when there exists
             infinite number of common point
    */

    assert(l != m && m != r && l != r && u
         != v);
    if(coplanar(l, m, r, u) && coplanar(l, m
         , r, v)) return -1;
    l = l - m;
    r = r - m;
    u = u - m;
    v = v - m;
```

```
    point C = crsp(l, r);
    double denom = dotp(v - u, C);
    if(fabs(denom) < eps) return 0;

    double alpha = -dotp(C, u) / denom;
    x = u + (v - u) * alpha + m;

    return 1;
}

double angle(point u, point v) { return acos
     ( max(-1.0, min(1.0, dotp(u, v) / (norm
     (u) * norm(v)))) );}

struct line3d{ //directed
    point d, o; // dir = direction, o =
         online point

    line3d(point p, point q){
        d = q - p;
        o = p;
        assert(sgn(norm(d)));
    }

    line3d(plane p1, plane p2){
        d = crsp(p1.n, p2.n);
        o = (crsp(p2.n*p1.d - p1.n*p2.d, d))
            /sq(d);
    }

    point get_p1(){return o;}
    point get_p2(){return o + d;}

    double dist(point p){ return norm(crsp(d
         , p - o)) / norm(d);};
    point project(point p){ return
         projection(p, o, o + d); }
    point reflect(point p) {return
         reflection(p, o, o + d);}
};

line3d perpThrough(plane p, point o){return
     line3d(o, o + p.n);}
plane perpThrough(line3d l, point o){return
     plane(l.d, dotp(l.d, o));}

double dist(line3d l1, line3d l2) {
    point n = crsp(l1.d, l2.d);
    if (!sgn(norm(n))) return l1.dist(l2.o);
    return abs(dotp(l2.o-l1.o, n))/norm(n);
}

point closestOnL1(line3d l1, line3d l2) {
    point n2 = crsp(l2.d, crsp(l1.d, l2.d));
    return l1.o + (l1.d * (dotp(l2.o-l1.o,
         n2))) / dotp(l1.d,n2);
}

double angle(plane p1, plane p2){return
     angle(p1.n, p2.n);}
bool isparallel(plane p1, plane p2){return !
     sgn(norm(crsp(p1.n, p2.n)));}
bool isperp(plane p1, plane p2) {return !sgn
     (dotp(p1.n, p2.n));}


double angle(line3d l1, line3d l2){return
     angle(l1.d, l2.d);}
bool isparallel(line3d l1, line3d l2){return
      !sgn(norm(crsp(l1.d, l2.d)));}
bool isperp(line3d l1, line3d l2) {return !
     sgn(dotp(l1.d, l2.d));}


double angle(plane p, line3d l) {return pi/2
      - angle(p.n, l.d);}
bool isParallel(plane p, line3d l) {return !
     sgn(dotp(p.n, l.d));}
bool isPerpendicular(plane p, line3d l) {
     return !sgn(norm(crsp(p.n, l.d)));}

point vector_area2(vector <point> &poly){
    point S = {0, 0, 0};
    for(int i = 0; i < (int) poly.size(); i
         ++)
        S = S + crsp(poly[i], poly[ (i + 1)
             % poly.size()]);
    return S;
}

double area(vector < point > &poly){ // All
     points must be co-planer
    return norm(vector_area2(poly)) * 0.5;
}

/*
    Polyhedrons
*/

bool operator <(point p, point q) { ///OK
    return tie(p.x, p.y, p.z) < tie(q.x, q.y
         , q.z);
}

struct edge {
    int v;
    bool same; // = is the common edge in
         the same order?
};

// Given a series of faces (lists of points)
     , reverse some of them
// so that their orientations are consistent
      [ every face then will point in the
     same direction, inside / outside ]
void reorient(vector< vector<point> > &fs) {
    int n = fs.size();
    // Find the common edges and create the
         resulting graph
    vector< vector<edge>  > g(n);
    map<pair<point,point>, int> es;
    for (int u = 0; u < n; u++) {
        for (int i = 0, m = fs[u].size(); i
             < m; i++) {
            point a = fs[u][i], b = fs[u][(i
                 +1)%m];
            // Let look at edge [AB]
            if (es.count({a,b})) { // seen in
                 same order
                int v = es[{a,b}];
                g[u].push_back({v,true});
                g[v].push_back({u,true});
            }
            else if (es.count({b,a})) { //
                 seen in different order
                int v = es[{b,a}];
                g[u].push_back({v,false});
                g[v].push_back({u,false});
            }
            else es[{a,b}] = u;
        }
    }

    vector<bool> vis(n,false), flip(n);
    flip[0] = false;
    queue<int> q;
    q.push(0);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (edge e : g[u]) {
            if (!vis[e.v]) {
                vis[e.v] = true;
                // If the edge was in the
                     same order,
                // exactly one of the two
                     should be flipped
                flip[e.v] = (flip[u] ^ e.same
                     );
                q.push(e.v);
            }
        }
    }

    for (int u = 0; u < n; u++)
        if (flip[u])
            reverse(fs[u].begin(), fs[u].end
                 ());
}

double volume(vector< vector<point> > fs) {
    double vol6 = 0.0;
    for (vector<point> f : fs)
        vol6 += dotp(vector_area2(f), f[0]);
    return abs(vol6) / 6.0;
}


/*
    Spherical Co-ordinate System
*/

point sph(double r, double lat, double lon)
     { // lat, lon in degrees
    lat *= pi/180, lon *= pi/180;
    return {r*cos(lat)*cos(lon), r*cos(lat)*
         sin(lon), r*sin(lat)};
}

double greatCircleDist(point o, double r,
     point a, point b) {
    return r * angle(a-o, b-o);
}

int main()
{

    plane p = {point(0, 0, 10), point(0, 1,
         10), point(1, 0, 10)};
//   plane q = {p.get_p1(), p.get_p2(), p.
     get_p3()};
    double d = dotp(p.get_p2() - p.get_p1(),
          p.get_p3() - p.get_p1());

    D(eq(d, 0))
    return 0;
}
```

4.3 Closest Pair of Points

```
LL ClosestPair(vector<pii> pts) {
    int n = pts.size();
    sort(all(pts));
    set<pii> s;

    LL best_dist = 1e18;
    int j = 0;
    for (int i = 0; i < n; ++i) {
        int d = ceil(sqrt(best_dist));
        while (pts[i].ff - pts[j].ff >=
             best_dist) {
            s.erase({pts[j].ss, pts[j].ff});
            j += 1;
        }

        auto it1 = s.lower_bound({pts[i].ss
             - d, pts[i].ff});
        auto it2 = s.upper_bound({pts[i].ss
             + d, pts[i].ff});

        for (auto it = it1; it != it2; ++it)
             {
            int dx = pts[i].ff - it->ss;
            int dy = pts[i].ss - it->ff;
            best_dist = min(best_dist, 1LL *
                 dx * dx + 1LL * dy * dy);
        }
        s.insert({pts[i].ss, pts[i].ff});
    }
    return best_dist;
}
```

4.4 Line and Circles

```
template <typename DT>
class line{
    public:
        point <DT> dir, O; // direction of
             vector and starting point
    line(point <DT> p,point <DT> q): dir(q -
         p), O(p) {};
```

```cpp
    bool Contains(const point <double> &p){
        return fabs(p - O & dir ) < EPS;
    } // checks whether the line Contains a certain point
    template <typename XT> point <XT> At(XT t){
        return point <XT> (dir) * t + O;
    } // inserts value of t in the vector representation, finds the point which is O + Dir*t
    double AtInv(const point <double> &p){
        return abs(dir.x) > 0 ? (p - O).x / dir.x : (p - O).y / dir.y;
    } // if the line Contains a point, gives the value t such that, p = O+Dir*t
    line Perp(point <DT> p){
        return line(p, p + (-dir.y,dir.x));
    }
    point <DT> ProjOfPoint(const point <DT> &P) {
        return O + dir * ((P - O) ^ dir) / (~dir);
    }
    double DisOfPoint(const point <DT> &P) {
        return fabs(dir & (P - O))/sqrt(~(dir));
    }
    friend bool Parallel(line& L, line& R){
        return fabs(R.dir & L.dir) < EPS;
    }
    friend int Intersects(line& L, line& R){
        return Parallel(L, R) ? R.Contains(L.O) ? -1 : 0 : 1;
    }
    friend pair <double, double> IntersectionAt(line &L, line &R){
        double r = double((L.O - R.O) & L.dir)/(R.dir & L.dir);
        double l = double((R.O - L.O) & R.dir)/(L.dir & R.dir);
        return {l, r};
    }
    friend pair <int, point<double>> IntersectionPoint(line L, line R, int _L = 0, int _R = 0){
        // _L and _R can be 0 to 3, 0 is a normal line, 3 is a segment, 1 and 2 are rays (considered bitwise)
        int ok = Intersects(L, R);
        if(ok == 0)
            return {0, {0, 0}};
        if(ok == 1){
            auto [l,r] = IntersectionAt(L, R);
            if(l < (0-EPS) and _L & 2 )
                return {0, {0, 0}};
            if(l > (1+EPS) and _L & 1)
                return {0, {0, 0}};
            if(r < (0-EPS) and _R & 2 )
                return {0, {0, 0}};
            if(r > (1+EPS) and _R & 1)
                return {0, {0, 0}};
            return {1, L.At(l)};
        }
        return {-1, {0,0}}; // they are the same line
    }
};
template <typename DT>
class circle {
public:
    point <DT> O;
    DT R;
    circle(const point <DT> &O = {0, 0}, DT R = 0) : O(O), R(R) {}
    // the next two make sense only on circle <double>
    circle(const point <DT> &A, const point <DT> &B, const point <DT> &C){
        point <DT> X = (A + B) / 2, Y = (B + C) / 2, d1 = Perp(A - B), d2 = Perp(B - C);
        O = IntersectionPoint(line(X, d1), line(Y, d2)).second;
        R = sqrt(~(O - A));
    }
    circle(const point <DT> &A, const point <DT> &B, DT R){
        point <DT> X = (A + B) / 2, d = Perp(A - B);
        d = d * (R / sqrt(~(d)));
        O = X + d;
        R = sqrt(~(O - A));
    }
    double SectorArea(double ang) {
        // Area of a sector of cicle
        return ang* R * R * .5;
    }
    double SectorArea(const point <DT> &a, const point <DT> &b) {
        return SectorArea(Angle(a - O, b - O));
    }
    double ChordArea(const point <DT> &a, const point <DT> &b) {
        // Area between sector and its chord
        return SectorArea(a, b) - 0.5 * TriArea(O, a, b);
    }
    int Contains(const point <DT> &p){
        // 0 for outside, 1 for inside, -1 for on the circle
        DT d = DisSq(O, p);
        return d > R * R ? 0 : (d == R * R ? -1 : 1);
    }
    friend tuple <int, point <DT>, point <DT>> IntersectionPoint(const circle &a,const circle &b) {
        if(a.R == b.R and a.O == b.O)
            return {-1, {0, 0}, {0, 0}};
        double d = sqrt(DisSq(a.O, b.O));
        if(d > a.R + b.R or d < fabs(a.R - b.R))
            return {0, {0, 0}, {0, 0}};
        double z = (sq(a.R) + sq(d) - sq(b.R)) / (2 * d);
        double y = sqrt(sq(a.R) - sq(z));
        point <DT> O = b.O - a.O, h = Perp(O) * (y / sqrt(~O));
        O = a.O + O * (z / sqrt(~O));
        return make_tuple(1 + (~(h) > EPS), O - h, O + h);
    }
    friend tuple <int, point <DT>, point <DT>> IntersectionPoint(const circle &C, line <DT> L) {
        point <DT> P = L.ProjOfPoint(C.O);
        double D = DisSq(C.O, P);
        if(D > C.R * C.R)
            return {0, {0, 0}, {0, 0}};
        double x = sqrt(C.R * C.R - D);
        point <DT> h = L.dir * (x / sqrt(~L.dir));
        return {1 + (x > EPS), P - h, P + h};
    }
    double SegmentedArea(point <DT> &a, point <DT> &b) {
        // signed area of the intersection between the circle and triangle OAB
        double ans = SectorArea(a, b);
        line <DT> L(a, b);
        auto [cnt, p1, p2] = IntersectionPoint(*this, L);
        if(cnt < 2)
            return ans;
        double t1 = L.AtInv(p1), t2 = L.AtInv(p2);
        if(t2 < 0 or t1 > 1)
            return ans;
        if(t1 < 0)
            p1 = a;
        if(t2 > 1)
            p2 = b;
        return ans - ChordArea(p1, p2);
    }
};
```

4.5 Pair of Intersecting segments using Line Sweep

```cpp
/*
Checking for the intersection of two segments is carried out by the intersect () function, using an algorithm based on the oriented area of the triangle.

The queue of segments is the global variable s, a set<event>. Iterators that specify the position of each segment in the queue (for convenient removal of segments from the queue) are stored in the global array where.

Two auxiliary functions prev() and next() are also introduced, which return iterators to the previous and next elements (or end(), if one does not exist).
*/
set<seg> s;
vector<set<seg>::iterator> where;

set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}

set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}

pair<int, int> solve(const vector<seg>& a) {
    int n = (int)a.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(e.begin(), e.end());

    s.clear();
    where.resize(a.size());
    for (size_t i = 0; i < e.size(); ++i) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev(nxt);
            if (nxt != s.end() && intersect(*nxt, a[id]))
                return make_pair(nxt->id, id);
            if (prv != s.end() && intersect(*prv, a[id]))
                return make_pair(prv->id, id);
            where[id] = s.insert(nxt, a[id]);
        } else {
            set<seg>::iterator nxt = next(where[id]), prv = prev(where[id]);
            if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv))
                return make_pair(prv->id, nxt->id);
            s.erase(where[id]);
        }
    }

    return make_pair(-1, -1);
}
```

4.6 Point in Polygon

```
template <typename DT> DT
    FarthestPairOfPoints(polygon <DT> p){
  p = ConvexHull(p);
  int n = p.size();
  DT ans = -1e9;
  for(int i = 0, j = 1; i < n; i++)
  {
    for( ; UTriArea(p[i], p[(i + 1) % n
        ], p[(j + 1) % n]) > UTriArea(p[
        i], p[(i + 1) % n], p[j]) ; j =
        (j + 1) % n ) ;
    ans = max(ans, DisSq(p[i], p[j]));
    ans = max(ans, DisSq(p[(i + 1) % n],
        p[j]));
  }
  return ans; // will return square of the
      answer.
}
template <typename DT> int
    PointInConvexPolygon(polygon <int> ::
    iterator b, polygon <int> :: iterator e
    , const point <DT> &O){
  polygon <int> :: iterator lo = b + 2, hi
      = e - 1, ans = e;
  while(lo <= hi) {
    auto mid = lo + (hi - lo) / 2;
    if(TriArea(*b, O, *mid) >= 0) ans =
        mid, hi = mid - 1;
    else lo = mid + 1;
  }
  if (ans == e or abs(UTriArea(*b, *(ans -
      1), *ans) - UTriArea(*b, *(ans -
      1), O) - UTriArea(*b, *ans, O) -
      UTriArea(*(ans - 1), *ans, O)) >
      EPS)
    return 0;
  else return (Collinear(*b, *(b + 1), O)
      or Collinear(*(e - 1), *b, O) or
      Collinear(*(ans), *(ans - 1), O)) ?
      -1 : 1;
} // 0 for outside, -1 for on border, 1 for
    inside

template <typename DT> int PointInPolygon(
    polygon <DT> &P, point <DT> pt) {
  int n = P.size();
  int cnt = 0;
  for(int i = 0, j = 1; i < n; i++, j = (j
      + 1) % n) {
    if(TriArea(pt, P[i], P[j]) == 0 and
        min(P[i], P[j]) <= pt and pt <=
        max(P[i], P[j]))
      return -1;
    cnt += ((P[j].y >= pt.y) - (P[i].y
        >= pt.y)) * TriArea(pt, P[i], P[
        j]) > 0;
  }
  return cnt & 1;
}

// returns 1e9 if the point is on the
    polygon
int winding_number(vector<PT> &p, const PT&
    z) { // O(n)
  if (is_point_on_polygon(p, z)) return 1
      e9;
  int n = p.size(), ans = 0;
  for (int i = 0; i < n; ++i) {
    int j = (i + 1) % n;
    bool below = p[i].y < z.y;
    if (below != (p[j].y < z.y)) {
      auto orient = orientation(z, p[j
          ], p[i]);
      if (orient == 0) return 0;
      if (below == (orient > 0)) ans +=
          below ? 1 : -1;
    }
  }
  return ans;
}
// -1 if strictly inside, 0 if on the
    polygon, 1 if strictly outside
int is_point_in_polygon(vector<PT> &p, const
    PT& z) { // O(n)
  int k = winding_number(p, z);
  return k == 1e9 ? 0 : k == 0 ? 1 : -1;
}
```

4.7 Rotating Calipers

```
template <typename DT> polygon <DT>
    ConvexHull(polygon <DT> &PT){
  sort(PT.begin(), PT.end());
  int m = 0, n = PT.size();

  polygon <DT> hull(n + n + 2);
  for(int i = 0; i < n; i++){
    for( ; m > 1 and TriArea(hull[m-2],
        hull[m-1], PT[i]) <= 0; m-- );
    hull[m++] = PT[i];
  }
  for(int i = n - 2, k = m; i >= 0; i--){
    for( ; m > k and TriArea(hull[m -
        2], hull[m - 1], PT[i]) <= 0; m
        --);
    hull[m++] = PT[i];
  }
  if(n > 1)
    m--;
  while(hull.size() > m)
    hull.pop_back();
  return hull;
}

template <typename DT> double
    MinimumBoundingBox(polygon <DT> P){
  auto p = ConvexHull(P);
  int n = p.size();
  double area = 1e20 + 5;
  for(int i = 0, l = 1, r = 1, u = 1 ; i <
      n ; i++){
    point <DT> edge = (p[(i+1)%n]- p[i])
        /sqrt(DisSq(p[i], p[(i+1)%n]));

    for( ; (edge ^ p[r%n]-p[i]) < (edge
        ^ p[(r+1)%n] - p[i]); r++);
    for( ; u<r || (edge & p[u%n] - p[i])
        < (edge & p[(u+1)%n] - p[i]); u
        ++) ;
    for( ; l<u || (edge ^ p[l%n] - p[i])
        > (edge ^ p[(l+1)%n] - p[i]); l
        ++);

    double w = (edge ^ p[r%n]-p[i]) - (
        edge ^ p[l%n] - p[i]);
    double h = UTriArea(p[u%n], p[i], p
        [(i+1)%n])/sqrt(DisSq(p[i], p[(i
        +1)%n]));
    area = min(area,w*h);
  }
  if(area>1e19)
    area = 0;
  return area;
}
```

4.8 Some More 2D Geo

```
Tf distancePointLine(Point p, Line l) {
  return abs(cross(l.b - l.a, p - l.a) /
      length(l.b - l.a));
}

// returns the shortest distance from point
    a to segment s
Tf distancePointSegment(Point p, Segment s)
    {
  if (s.a == s.b) return length(p - s.a);
  Point v1 = s.b - s.a, v2 = p - s.a, v3 =
      p - s.b;
  if (dcmp(dot(v1, v2)) < 0)
    return length(v2);
  else if (dcmp(dot(v1, v3)) > 0)
    return length(v3);
  else
    return abs(cross(v1, v2) / length(v1
        ));
}

// returns the shortest distance from
    segment p to segment q
Tf distanceSegmentSegment(Segment p, Segment
    q) {
  if (segmentsIntersect(p, q)) return 0;
  Tf ans = distancePointSegment(p.a, q);
  ans = min(ans, distancePointSegment(p.b,
      q));
  ans = min(ans, distancePointSegment(q.a,
      p));
  ans = min(ans, distancePointSegment(q.b,
      p));
  return ans;
}

// returns the projection of point p on line
    l
Point projectPointLine(Point p, Line l) {
  static_assert(is_same<Tf, Ti>::value);
  Point v = l.b - l.a;
  return l.a + v * ((Tf)dot(v, p - l.a) /
      dot(v, v));
}
// returns the left side of polygon u after
    cutting it by ray a->b
Polygon cutPolygon(Polygon u, Point a, Point
    b) {
  using Linear::lineLineIntersection,
      Linear::onSegment;

  Polygon ret;
  int n = u.size();
  for (int i = 0; i < n; i++) {
    Point c = u[i], d = u[(i + 1) % n];
    if (dcmp(cross(b - a, c - a)) >= 0)
        ret.push_back(c);
    if (dcmp(cross(b - a, d - c)) != 0)
        {
      Point t;
      lineLineIntersection(a, b - a, c,
          d - c, t);
      if (onSegment(t, Segment(c, d)))
          ret.push_back(t);
    }
  }
  return ret;
}
// returns false if points are collinear,
    true otherwise
// circle p touch each arm of triangle abc
bool inscribedCircle(Point a, Point b, Point
    c, Circle &p) {
  using Linear::distancePointLine;
  static_assert(is_same<Tf, Ti>::value);
  if (orient(a, b, c) == 0) return false;
  Tf u = length(b - c);
  Tf v = length(c - a);
  Tf w = length(a - b);
  p.o = (a * u + b * v + c * w) / (u + v +
      w);
  p.r = distancePointLine(p.o, Line(a, b))
      ;
  return true;
}

// set of points A(x, y) such that PA : QA =
    rp : rq
Circle apolloniusCircle(Point P, Point Q, Tf
    rp, Tf rq) {
  static_assert(is_same<Tf, Ti>::value);
  rq *= rq;
  rp *= rp;
  Tf a = rq - rp;
  assert(dcmp(a));
  Tf g = (rq * P.x - rp * Q.x) / a, h = (
      rq * P.y - rp * Q.y) / a;
  Tf c =
    (rq * P.x * P.x - rp * Q.x * Q.x +
        rq * P.y * P.y - rp * Q.y * Q.y)
        / a;
```

```
    Point o(g, h);
    Tf R = sqrt(g * g + h * h - c);
    return Circle(o, R);
}
// returns false if points are collinear
bool circumscribedCircle(Point a, Point b,
     Point c, Circle &p) {
    using Linear::lineLineIntersection;
    if (orient(a, b, c) == 0) return false;
    Point d = (a + b) / 2, e = (a + c) / 2;
    Point vd = rotate90(b - a), ve =
         rotate90(a - c);
    bool f = lineLineIntersection(d, vd, e,
         ve, p.o);
    if (f) p.r = length(a - p.o);
    return f;
}

// Following three methods implement Welzl's
      algorithm for
// the smallest enclosing circle problem:
     Given a set of
// points, find out the minimal circle that
     covers them all.
// boundary(p) determines (if possible) a
     circle that goes
// through the points in p. Ideally |p| <=
     3.
// welzl() is a recursive helper function
     doing the most part
// of Welzl's algorithm. Call minidisk with
     the set of points
// Randomized Complexity: O(CN) with C~10 (
     practically lesser)
// TESTED: ICPC Dhaka 2019 - I (CodeMarshal)

Circle boundary(const vector<Point> &p) {
    Circle ret;
    int sz = p.size();
    if (sz == 0)
        return Circle({0, 0}, 0);
    else if (sz == 1)
        ;
    else if (sz == 2)
        ret.o = (p[0] + p[1]) / 2, ret.r =
             length(p[0] - p[1]) / 2;
    else if (!circumscribedCircle(p[0], p
         [1], p[2], ret))
        ret.r = 0;
    return ret;
}
Circle welzl(const vector<Point> &p, int fr,
      vector<Point> &b) {
    if (fr >= (int)p.size() || b.size() ==
         3) return boundary(b);

    Circle c = welzl(p, fr + 1, b);
    if (!c.contains(p[fr])) {
        b.push_back(p[fr]);
        c = welzl(p, fr + 1, b);
        b.pop_back();
    }
    return c;
}
Circle minidisk(vector<Point> p) {
    random_shuffle(p.begin(), p.end());
    vector<Point> q;
    return welzl(p, 0, q);
}
```

5 Graph

5.1 Block Cut Tree

```
bool ap[MAX];
int id[MAX], koyta[MAX];
int d[MAX], low[MAX];
bool vis[MAX];
vii g[MAX], tree[MAX];
int d_t;
stack<int>st;
vector<vector<int>>comp;
void articulation(int u, int p) {
  vis[u] = true;
  d[u] = low[u] = ++d_t;
  int child = 0; st.push(u);
  for (int v : g[u]) {
    if (v == p) continue;
    if (!vis[v]) {
      child++;
      articulation(v, u);
      low[u] = min(low[u], low[v]);
      if (p == -1 && child > 1) ap[u] = true
          ;
      if (low[v] >= d[u]) {
        if (p != -1) ap[u] = true;
        comp.pb({u}); int top;
        do {
          top = st.top(); st.pop();
          comp.back().pb(top);
        } while (top != v);
      }
    } else low[u] = min(low[u], d[v]);
  }
}
int node = 0;
void make_tree(int n) {
  for (int i = 1; i <= n; i++) {
    if (ap[i]) id[i] = ++node;
  }
  for (int i = 0; i < comp.size(); i++) {
    ++node;
    int cnt = 0;
    for (int u : comp[i]) {
      if (ap[u]) tree[node].pb(id[u]), tree[
          id[u]].pb(node), koyta[id[u]] =
          1;
      else id[u] = node, cnt++;
    }
    koyta[node] = cnt;
  }
}
```

5.2 Bridge Tree

```
vector<int> tree[MAX];
bool vis[MAX];
int d[MAX], low[MAX];
int id[MAX];
int d_t;

struct edge {
  int v, rev;
  edge() {}
  edge(int v, int rev) : v(v), rev(rev) {}
};
vector<edge>g[MAX];
vector<bool>is_bridge[MAX];
queue<int>q[MAX];
int comp = 1;

void add_edge(int u, int v) {
  edge _u = edge(v, g[v].size());
  edge _v = edge(u, g[u].size());
  g[u].pb(_u);
  g[v].pb(_v);
  is_bridge[u].pb(false);
  is_bridge[v].pb(false);
}

void bridge(int u, int p) {
  vis[u] = true;
  d[u] = low[u] = ++d_t;

  for (int i = 0; i < g[u].size(); i++) {
    edge e = g[u][i]; int v = e.v;
    if (v == p) continue;
    if (!vis[v]) {
      bridge(v, u);
      low[u] = min(low[v], low[u]);
      if (low[v] > d[u]) {
        is_bridge[u][i] = true;
        is_bridge[v][e.rev] = true;
      }
    } else low[u] = min(low[u], d[v]);
  }
}
void make_tree(int node) {
  int cur = comp; q[cur].push(node);
  vis[node] = true; id[node] = cur;
  while (!q[cur].empty()) {
    int u = q[cur].front(); q[cur].pop();
    for (int i = 0; i < g[u].size(); i++) {
      edge e = g[u][i]; int v = e.v;
      if (vis[v]) continue;
      if (is_bridge[u][i]) {
        comp++;
        tree[cur].pb(comp);
        tree[comp].pb(cur);
        make_tree(v);
      } else {
        q[cur].push(v);
        vis[v] = true; id[v] = cur;
      }
    }
  }
}
```

5.3 Centroid Decomposition

```
// problem: calculate the sum of number of
     distinct colors in the path between any
      two pair of nodes
//centroid decomposition (res[i] contains
     the sum of numbers of distinct colors
     in all paths from i)
set<int>g[MAX];
int col[MAX], child[MAX], used[18][MAX];
ll ans[MAX], res[MAX];
int sz = 0, uniq = 0, n;
bool vis[MAX];
void dfs(int u, int p) {
  sz++; child[u] = 1;
  for (auto v : g[u]) {
    if (v != p) {
      dfs(v, u);
      child[u] += child[v];
    }
  }
}
int get_centroid(int u, int p) {
  for (auto v : g[u]) {
    if (v != p && child[v] > sz / 2) return
         get_centroid(v, u);
  }
  return u;
}
void add(int u, int p, int depth, int
     centroid) {
  bool check = false; child[u] = 1;
  if (!vis[col[u]]) {
    uniq++; check = true;
    vis[col[u]] = true;
  }
  ans[centroid] += uniq;
  for (auto v : g[u]) {
    if (v != p) {
      add(v, u, depth, centroid);
      child[u] += child[v];
    }
  }
  if (check) {
    uniq--;
    used[depth][col[u]] += child[u];
    vis[col[u]] = false;
  }
}
void del(int u, int p, int depth, int
     centroid) {
  bool check = false;
  if (!vis[col[u]]) {
    uniq++;
    used[depth][col[u]] -= child[u];
    vis[col[u]] = true; check = true;
  }
  ans[centroid] -= uniq;
  for (auto v : g[u]) {
    if (v != p) del(v, u, depth, centroid);
  }
  child[u] = 0;
  if (check) uniq--; vis[col[u]] = false;
```

```cpp
}
void solve(int u, int p, int depth, int
    centroid) {
  ans[u] += (ans[p] + child[centroid] - used
      [depth][col[u]]);
  res[u] += ans[u];
  int temp = used[depth][col[u]];
  used[depth][col[u]] = child[centroid];
  for (auto v : g[u]) {
    if (v != p) solve(v, u, depth, centroid)
        ;
  }
  ans[u] = 0;
  used[depth][col[u]] = temp;
}
void reset_col(int u, int p, int depth) {
  used[depth][col[u]] = 0;
  for (auto v : g[u]) {
    if (v != p) reset_col(v, u, depth);
  }
}
void decompose(int u, int depth) {
  sz = 0;
  uniq = 0;
  dfs(u, -1);
  int centroid = get_centroid(u, -1);
  reset_col(centroid, -1, depth);
  add(centroid, -1, depth, centroid); ///get
      ans for centroid and get the number
      of paths where each color is used
  res[centroid] += ans[centroid];
  uniq++;
  vis[col[centroid]] = true;
  for (auto v : g[centroid]) {
    child[centroid] -= child[v];
    ///remove all contribution of the
        subtree of v
    del(v, centroid, depth, centroid);
    used[depth][col[centroid]] = child[
        centroid];
    solve(v, centroid, depth, centroid);
    ///add back the contribution of the
        subtree of v
    add(v, centroid, depth, centroid);
    child[centroid] += child[v];
  }
  uniq--;
  vis[col[centroid]] = false;
  for (auto it = g[centroid].begin(); it !=
      g[centroid].end(); it++) {
    g[*it].erase(centroid);
    decompose(*it, depth + 1);
  }
}
int arr[MAX];
int main() {
  fastio;
  cin >> n;
  for (int i = 1; i <= n; i++) cin >> col[i
      ];
  for (int i = 0; i < n - 1; i++) {
    int u, v;
    cin >> u >> v;
    g[u].insert(v); g[v].insert(u);
  }
  decompose(1, 0);
  for (int i = 1; i <= n; i++) cout << res[i
      ] << "\n";
}
```

5.4 Dinic Max-Flow

```cpp
int src, sink;
int dis[MAX], q[MAX], work[MAX];
int n, m, nodes;

struct edge {
  int v, rev, cap, flow;
  edge() {}
  edge(int v, int rev, int cap) : v(v), rev(
      rev), cap(cap), flow(0) {}
};
vector<edge>g[MAX];
void add_edge(int u, int v, int cap, int rev
    = 0) {
  edge _u = edge(v, g[v].size(), cap);
  edge _v = edge(u, g[u].size(), rev);
  g[u].pb(_u);
  g[v].pb(_v);
}

bool dinic_bfs(int s) {
  fill(dis, dis + nodes, -1);
  dis[s] = 0;
  int id = 0;
  q[id++] = s;

  for (int i = 0; i < id; i++) {
    int u = q[i];
    for (int j = 0; j < g[u].size(); j++) {
      edge &e = g[u][j];
      if (dis[e.v] == -1 && e.flow < e.cap)
          {
        dis[e.v] = dis[u] + 1;
        q[id++] = e.v;
      }
    }
  }
  return dis[sink] >= 0;
}

int dinic_dfs(int u, int f) {
  if (u == sink) return f;
  for (int &i = work[u]; i < g[u].size(); i
      ++) {
    edge &e = g[u][i];
    if (e.cap <= e.flow) continue;
    if (dis[e.v] == dis[u] + 1) {
      int flow = dinic_dfs(e.v, min(f, e.cap
          - e.flow));
      if (flow) {
        e.flow += flow;
        g[e.v][e.rev].flow -= flow;
        return flow;
      }
    }
  }
  return 0;
}

int max_flow(int _src, int _sink) {
  src = _src;
  sink = _sink;
  int ret = 0;
  while (dinic_bfs(src)) {
    fill(work, work + nodes, 0);
    while (int flow = dinic_dfs(src, INT_MAX
        )) {
      ret += flow;
    }
  }
  return ret;
}
```

5.5 Flow with Lower Bound

```cpp
///flow with demand(lower bound) only for
    DAG
//create new src and sink
//add_edge(new src, u, sum(in_demand[u]))
//add_edge(u, new sink, sum(out_demand[u]))
//add_edge(old sink, old src, inf)
// if (sum of lower bound == flow) then
    demand satisfied
//flow in every edge i = demand[i] + e.flow
```

5.6 Heavy Light Decomposition

```cpp
int arr[MAX], n;
vector<int> parent, depth, heavy, head, pos;
int cur_pos, sub[MAX];
int tree[4 * MAX];
vii g[MAX];
void update(int now, int L, int R, int idx,
    int val) {
  if (L == R) {
    tree[now] = val;
    return;
  }
  int mid = (L + R) / 2;
  if (idx <= mid) update(now << 1, L, mid,
      idx, val);
  else update( (now << 1) | 1, mid + 1, R,
      idx, val);
  tree[now] = tree[now << 1] + tree[(now <<
      1) | 1];
}
ll segtree_query(int now, int L, int R, int
    i, int j) {
  if (R < i || L > j) return 0;
  if (L >= i && R <= j) return tree[now];
  int mid = (L + R) / 2;
  return segtree_query(now << 1, L, mid, i,
      j) + segtree_query((now << 1) | 1,
      mid + 1, R, i, j);
}
int dfs(int u) {
  sub[u] = 1;
  int mx_size = 0;
  for (int v : g[u]) {
    if (v != parent[u]) {
      parent[v] = u, depth[v] = depth[u] +
          1;
      int v_size = dfs(v);
      sub[u] += v_size;
      if (v_size > mx_size) {
        mx_size = v_size;
        heavy[u] = v;
      }
    }
  }
  return sub[u];
}
void decompose(int u, int h) {
  head[u] = h, pos[u] = cur_pos++;
  if (heavy[u] != -1) decompose(heavy[u], h)
      ;
  for (int v : g[u]) {
    if (v != parent[u] && v != heavy[u])
        decompose(v, v);
  }
}
void init(int n) {
  parent = vector<int>(n, -1);
  depth = vector<int>(n);
  heavy = vector<int>(n, -1);
  head = vector<int>(n);
  pos = vector<int>(n);
  cur_pos = 1;
  dfs(1); decompose(1, 1);
}
ll query(int a, int b) {
  ll res = 0;
  for (; head[a] != head[b]; b = parent[head
      [b]]) {
    if (depth[head[a]] > depth[head[b]])
        swap(a, b);
    ll cur_heavy_path_res = segtree_query(1,
         1, n, pos[head[b]], pos[b]);
    res += cur_heavy_path_res;
  }
  if (depth[a] > depth[b]) swap(a, b);
  ll last_heavy_path_res = segtree_query(1,
      1, n, pos[a], pos[b]);
  res += last_heavy_path_res;
  return res;
}
```

5.7 K-th Root of a Permutation

```cpp
vector<vector<int>> decompose(vector<int> &p
    ) {
  int n = p.size();
  vector<vector<int>> cycles;
  vector<bool> vis(n, 0);
  for (int i = 0; i < n; i++) {
    if (!vis[i]) {
      vector<int> v;
      while (!vis[i]) {
```

```
        v.push_back(i);
        vis[i] = 1;
        i = p[i];
      }
      cycles.push_back(v);
    }
  }
  return cycles;
}
vector<int> restore(int n, vector<vector<int>> &cycles) {
  vector<int> p(n);
  for (auto v : cycles) {
    int m = v.size();
    for (int i = 0; i < m; i++) p[v[i]] = v[(i + 1) % m];
  }
  return p;
}
//cycle decomposition of the k-th power of p
vector<vector<int>> power(vector<int> &p, int k) {
  int n = p.size();
  auto cycles = decompose(p);
  vector<vector<int>> ans;
  for (auto v : cycles) {
    int len = v.size(), g = __gcd(k, len); //g cycles of len / g length
    for (int i = 0; i < g; i++) {
      vector<int> w;
      for (int j = i, cnt = 0; cnt < len / g; cnt++, j = (j + k) % len) {
        w.push_back(v[j]);
      }
      ans.push_back(w);
    }
  }
  return ans;
}
//cycle decomposition of the k-th root of p with minimum disjoint cycles
//if toggle = 1, then the parity of number of cycles will be different from the other(toggle = 0) if possible
//returns empty vector if no solution exists
vector<vector<int>> root(vector<int> &p, int k, int toggle = 0) {
  int n = p.size();
  vector<vector<int>> cycles[n + 1];
  auto d = decompose(p);
  for (auto v : d) cycles[(int)v.size()].push_back(v);
  vector<vector<int>> ans;
  for (int len = 1; len <= n; len++) {
    if (cycles[len].empty()) continue;
    int tmp = k, t = 1, x = __gcd(len, tmp);
    while (x != 1) {
      t *= x;
      tmp /= x;
      x = __gcd(len, tmp);
    }
    if ((int)cycles[len].size() % t != 0) {
      ans.clear();
      return ans; //no solution exists
    }
    int id = 0;
    //we can merge t * z cycles iff tmp % z === 0
    if (toggle && tmp % 2 == 0 && (int)cycles[len].size() >= 2 * t) {
      int m = 2 * t * len;
      vector<int> cycle(m);
      for (int x = 0; x < 2 * t; x++) { // merging 2t cycles to perform the toggle
        for (int y = 0; y < len; y++) {
          cycle[(x + 1LL * y * k) % m] = cycles[len][x][y];
        }
      }
      ans.push_back(cycle);
      id = 2 * t;
      toggle = 0;
    }
    for (int i = id; i < (int)cycles[len].size(); i += t) {
      int m = t * len;
      vector<int> cycle(m);
      for (int x = 0; x < t; x++) { // merging t cycles
        for (int y = 0; y < len; y++) {
          cycle[(x + 1LL * y * k) % m] = cycles[len][i + x][y];
        }
      }
      ans.push_back(cycle);
    }
  }
  return ans;
}
//minimum swaps to obtain this perm from unit perm
vector<pair<int, int>> transpositions(vector<vector<int>> &cycles) {
  vector<pair<int, int>> ans;
  for (auto v : cycles) {
    int m = v.size();
    for (int i = m - 1; i > 0; i--) ans.push_back({v[0], v[i]});
  }
  return ans;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, l, k;
  cin >> n >> l >> k;
  vector<int> p(n);
  for (auto &x : p) cin >> x, --x;
  auto a = root(p, k);
  if (a.empty()) return cout << "no solution\n", 0;
  auto t = transpositions(a);
  if (t.size() % 2 != l % 2) {
    a = root(p, k, 1);
    t = transpositions(a);
  }
  if (t.size() % 2 != l % 2 || t.size() > l) return cout << "no solution\n", 0;
  auto z = restore(n, a);
  auto w = power(z, k);
  auto x = restore(n, w);
  assert(p == x);
  for (auto x : t) cout << x.first + 1 << ' ' << x.second + 1 << '\n';
  for (int i = t.size(); i < l; i++) cout << 1 << ' ' << 2 << '\n';
  return 0;
}
```

5.8 Lowest Common Ancestor

```
int lvl[MAX], P[MAX][25];
void dfs(int u, int par, int d) {
  lvl[u] = d;
  P[u][0] = par;
  for (int v : g[u]) {
    if (v == par) continue;
    dfs(v, u, d + 1);
  }
}

void init() {
  for (int j = 0; j < 25; j++) {
    for (int i = 0; i <= n; i++) P[i][j] = -1;
  }

  dfs(1, -1, 0);

  for (j = 1; j < 25; j++) {
    for (int i = 1; i <= n; i++) {
      if (P[i][j - 1] != -1) {
        P[i][j] = P[ P[i][j - 1] ][j - 1];
        ///to find max weight between two edges
        // weight[i][j] = max(weight[i][j-1], weight[p[i][j-1]][j-1]);
      }
    }
  }
}

int lca(int u, int v) {
  int i, lg;
  if (lvl[u] < lvl[v]) swap(u, v);

  for (lg = 0; (1 << lg) <= lvl[u]; lg++);
  lg--;

  for (i = lg; i >= 0; i--) {
    if (lvl[u] - (1 << i) >= lvl[v]) {
      u = P[u][i];
    }
  }

  if (u == v) return u;

  for (i = lg; i >= 0; i--) {
    if (P[u][i] != -1 && P[u][i] != P[v][i]) {
      u = P[u][i], v = P[v][i];
      // ret = max(ret, weight[u][i]);
      // ret = max(ret, weight[v][i]);
    }
  }
  // ret = max(ret, weight[u][0]);
  return P[u][0];
}

//Get the ancestor of node "u"
//which is "dis" distance above.
int getAncestor(int u, int dis) {
  dis = lvl[u] - dis;
  int i, lg = 0;
  for (; (1 << lg) <= lvl[u]; lg++) continue;
  lg--;

  for (i = lg; i >= 0; i--) {
    if (lvl[u] - (1 << i) >= dis) {
      u = P[u][i];
    }
  }
  return u;
}

//returns the distance between
//two nodes "u" and "v".
int dis(int u, int v) {
  if (lvl[u] < lvl[v]) swap(u, v);
  int p = lca(u, v);
  return lvl[u] + lvl[v] - 2 * lvl[p];
}
```

5.9 Max Clique

```
const int N = 42;

int g[N][N];
int res;
long long edges[N];
//3 ^ (n / 3)
void BronKerbosch(int n, long long R, long long P, long long X) {
  if (P == 0LL && X == 0LL) { //here we will find all possible maximal cliques (not maximum) i.e. there is no node which can be included in this set
    int t = __builtin_popcountll(R);
    res = max(res, t);
    return;
  }
  int u = 0;
  while (!((1LL << u) & (P | X))) u ++;
  for (int v = 0; v < n; v++) {
    if (((1LL << v) & P) && !((1LL << v) & edges[u])) {
      BronKerbosch(n, R | (1LL << v), P & edges[v], X & edges[v]);
      P -= (1LL << v);
```

```
      X |= (1LL << v);
    }
  }
}

int max_clique (int n) {
  res = 0;
  for (int i = 1; i <= n; i++) {
    edges[i - 1] = 0;
    for (int j = 1; j <= n; j++) if (g[i][j
        ]) edges[i - 1] |= ( 1LL << (j - 1)
        );
  }
  BronKerbosch(n, 0, (1LL << n) - 1, 0);
  return res;
}
```

5.10 Min Cost Max Flow

```
mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());
struct edge {
  int v, rev;
  ll cap, cost, flow;
  edge() {}
  edge(int v, int rev, ll cap, ll cost) : v(
      v), rev(rev), cap(cap), cost(cost),
      flow(0) {}
};
struct mcmf {
  int src, sink, nodes;
  vector<int> par, idx, Q;
  vector<bool> inq;
  vector<ll> dis;
  vector<vector<edge>> g;
  mcmf() {}
  mcmf(int src, int sink, int nodes) : src(
      src), sink(sink), nodes(nodes),
    par(nodes), idx(nodes), inq(nodes),
    dis(nodes), g(nodes), Q(10000005) {} //
        use Q(nodes) if not using random
  void add_edge(int u, int v, ll cap, ll
      cost, bool directed = true) {
    edge _u = edge(v, g[v].size(), cap, cost
        );
    edge _v = edge(u, g[u].size(), 0, -cost)
        ;
    g[u].pb(_u);
    g[v].pb(_v);
    if (!directed) add_edge(v, u, cap, cost,
        true);
  }
  bool spfa() {
    for (int i = 0; i < nodes; i++) {
      dis[i] = inf, inq[i] = false;
    }
    int f = 0, l = 0;
    dis[src] = 0, par[src] = -1, Q[l++] =
        src, inq[src] = true;
    while (f < l) {
      int u = Q[f++];
      for (int i = 0; i < g[u].size(); i++)
          {
        edge &e = g[u][i];
        if (e.cap <= e.flow) continue;
        if (dis[e.v] > dis[u] + e.cost) {
          dis[e.v] = dis[u] + e.cost;
          par[e.v] = u, idx[e.v] = i;
          if(!inq[e.v]) inq[e.v] = true, Q[l
              ++] = e.v;
          // if (!inq[e.v]) {
          //   inq[e.v] = true;
          //   if (f && rnd() & 7) Q[--f] = e
              .v;
          //   else Q[l++] = e.v;
          // }
        }
      }
      inq[u] = false;
    }
    return (dis[sink] != inf);
  }
  pair<ll, ll> solve() {
    ll mincost = 0, maxflow = 0;
    while (spfa()) {
      ll bottleneck = inf;
      for (int u = par[sink], v = idx[sink];
          u != -1; v = idx[u], u = par[u])
          {
        edge &e = g[u][v];
        bottleneck = min(bottleneck, e.cap -
            e.flow);
      }
      for (int u = par[sink], v = idx[sink];
          u != -1; v = idx[u], u = par[u])
          {
        edge &e = g[u][v];
        e.flow += bottleneck;
        g[e.v][e.rev].flow -= bottleneck;
      }
      mincost += bottleneck * dis[sink],
          maxflow += bottleneck;
    }
    return make_pair(mincost, maxflow);
  }
};
// want to minimize cost and don't care
    about flow
// add edge from sink to dummy sink (cap =
    inf, cost = 0)
// add edge from source to sink (cap = inf,
    cost = 0)
// run mcmf, cost returned is the minimum
    cost
```

5.11 Steiner Tree

```
/*
Find the minimum cost connected tree where
    at least the important nodes are
    connected
dp(x,i) = minimum cost of a tree rooted at i
     connecting the important node in
    bitmask x.
Complexity: O(3^k * n + 2^k * m log m)
*/

int n, k, m;
vector<int> imp;//k important nodes
vector<pair<int, long long>> g[N];
long long d[32][N]; //[2^k][edge count]
const long long inf = LLONG_MAX / 3;
bool vis[N];
long long MST() {
  for(int i = 0; i < (1 << k); i++) fill(d[i
      ], d[i] + N, inf);
  for(int i = 0; i < k; ++i) {
    d[1 << i][imp[i]] = 0;
  }
  priority_queue<pair<long long, int>> q;
  for(int mask = 1; mask < (1 << k); ++mask)
      {
    for(int a = 0; a < mask; ++a) { //you
        can still fasten this loop to get
        exact 3^k complexity
      if((a | mask) != mask) continue; //we
          only need the subsets
      int b = mask ^ a;
      if(b > a) continue;
      for(int v = 0; v < n; ++v) {
        d[mask][v] = min(d[mask][v], d[a][v]
            + d[b][v]);
      }
    }
    memset(vis, 0, sizeof vis);
    for(int v = 0; v < n; ++v) {
      if(d[mask][v] == inf) continue;
      q.emplace(-d[mask][v], v);
    }

    while(!q.empty()) {
      long long cost = -q.top().first;
      int v = q.top().second;
      q.pop();
      if(vis[v]) continue;
      vis[v] = true;
      for(auto edge : g[v]) {
        long long ec = cost + edge.second;
        if(ec < d[mask][edge.first]) {
          d[mask][edge.first] = ec;
          q.emplace(-ec, edge.first);
        }
      }
    }
  }

  long long res = inf;
  for(int v = 0; v < n; ++v) {
    res = min(res, d[(1 << k) - 1][v]);
  }
  return res;
}
```

5.12 Tree Isomorphism

```
mp["01"] = 1;
ind = 1;
int dfs(int u, int p) {
  int cnt = 0;
  vector<int>vs;
  for (auto v : g1[u]) {
    if (v != p) {
      int got = dfs(v, u);
      vs.pb(got);
      cnt++;
    }
  }
  if (!cnt) return 1;

  sort(vs.begin(), vs.end());
  string s = "0";
  for (auto i : vs) s += to_string(i);
  vs.clear();
  s.pb('1');
  if (mp.find(s) == mp.end()) mp[s] = ++ind;
  int ret = mp[s];
  return ret;
}
```

6 Math

6.1 Berlekamp Massey

```
struct berlekamp_massey { // for linear
    recursion
  typedef long long LL;
  static const int SZ = 2e5 + 5;
  static const int MOD = 1e9 + 7; /// mod
      must be a prime
  LL m , a[SZ] , h[SZ] , t_[SZ] , s[SZ] , t[
      SZ];
  // bigmod goes here
  inline vector <LL> BM( vector <LL> &x ) {
    LL lf , ld;
    vector <LL> ls , cur;
    for ( int i = 0; i < int(x.size()); ++i
        ) {
      LL t = 0;
      for ( int j = 0; j < int(cur.size());
          ++j ) t = (t + x[i - j - 1] * cur
          [j]) % MOD;
      if ( (t - x[i]) % MOD == 0 ) continue;
      if ( !cur.size() ) {
        cur.resize( i + 1 );
        lf = i; ld = (t - x[i]) % MOD;
        continue;
      }
      LL k = -(x[i] - t) * bigmod( ld , MOD
          - 2 , MOD ) % MOD;
      vector <LL> c(i - lf - 1);
      c.push_back( k );
      for ( int j = 0; j < int(ls.size());
          ++j ) c.push_back(-ls[j] * k %
          MOD);
      if ( c.size() < cur.size() ) c.resize(
           cur.size() );
      for ( int j = 0; j < int(cur.size());
          ++j ) c[j] = (c[j] + cur[j]) %
          MOD;
      if (i - lf + (int)ls.size() >= (int)
          cur.size() ) ls = cur, lf = i, ld
           = (t - x[i]) % MOD;
      cur = c;
```

```
    }
    for ( int i = 0; i < int(cur.size()); ++
        i ) cur[i] = (cur[i] % MOD + MOD) %
         MOD;
    return cur;
  }
  inline void mull( LL *p , LL *q ) {
    for ( int i = 0; i < m + m; ++i ) t_[i]
        = 0;
    for ( int i = 0; i < m; ++i ) if ( p[i]
        )
      for ( int j = 0; j < m; ++j ) t_[i +
          j] = (t_[i + j] + p[i] * q[j])
          % MOD;
    for ( int i = m + m - 1; i >= m; --i )
      if ( t_[i] )
      for ( int j = m - 1; ~j; --j ) t_[i
          - j - 1] = (t_[i - j - 1] + t_[i
          ] * h[j]) % MOD;
    for ( int i = 0; i < m; ++i ) p[i] = t_[
        i];
  }
  inline LL calc( LL K ) {
    for ( int i = m; ~i; --i ) s[i] = t[i] =
         0;
    s[0] = 1; if ( m != 1 ) t[1] = 1; else t
        [0] = h[0];
    while ( K ) {
      if ( K & 1 ) mull( s , t );
      mull( t , t ); K >>= 1;
    }
    LL su = 0;
    for ( int i = 0; i < m; ++i ) su = (su +
         s[i] * a[i]) % MOD;
    return (su % MOD + MOD) % MOD;
  }
  /// already calculated upto k , now
       calculate upto n.
  inline vector <LL> process( vector <LL> &x
      , int n , int k ) {
    auto re = BM( x );
    x.resize( n + 1 );
    for ( int i = k + 1; i <= n; i++ ) {
      for ( int j = 0; j < re.size(); j++ )
          {
        x[i] += 1LL * x[i - j - 1] % MOD *
            re[j] % MOD; x[i] %= MOD;
      }
    }
    return x;
  }
  inline LL work( vector <LL> &x , LL n ) {
    if ( n < int(x.size()) ) return x[n] %
        MOD;
    vector <LL> v = BM( x ); m = v.size();
        if ( !m ) return 0;
    for ( int i = 0; i < m; ++i ) h[i] = v[i
        ], a[i] = x[i];
    return calc( n ) % MOD;
  }
} rec;
vector <LL> v;
void solve() {
  int n;
  cin >> n;
  cout << rec.work(v, n - 1) << endl;
}
```

6.2 Chinese Remainder Theorem

```
// given a, b will find solutions for
// ax + by = 1
tuple <LL,LL,LL> EGCD(LL a, LL b){
    if(b == 0) return {1, 0, a};
    else{
        auto [x,y,g] = EGCD(b, a%b);
        return {y, x - a/b*y,g};
    }
}
// given modulo equations, will apply CRT
PLL CRT(vector <PLL> &v){
    LL V = 0, M = 1;
    for(auto &[v, m]:v){
        auto [x, y, g] = EGCD(M, m);
        if((v - V) % g != 0)
            return {-1, 0};
        V += x * (v - V) / g % (m / g) * M,
            M *= m / g;
        V = (V % M + M) % M;
    }
    return make_pair(V, M);
}
```

6.3 Derangements

```
array <int, N + 1> Drng;
void init(){
    Drng[0] = 1, Drng[1] = 0;
    for(int i = 2; i <= N; i++)
        Drng[i] = (LL) (i - 1) * (Drng[i -
            1] + Drng[i - 2]) % mod;
}
int D(int n) {
    return n < 0 ? 0 : Drng[n];
}
```

6.4 FFT in Mod

```
// need to add modulo to res[i] in Mul
vector<ll> Mul_mod(vector<ll>& a, vector<ll
    >& b, ll mod) {
  ll sqrt_mod = (ll)sqrtl(mod);
  vector<ll> a0(a.size()), a1(a.size());
  vector<ll> b0(b.size()), b1(b.size());
  for (int i = 0; i < a.size(); i++) {
    a0[i] = a[i] % sqrt_mod;
    a1[i] = a[i] / sqrt_mod;
  }
  for (int i = 0; i < b.size(); i++) {
    b0[i] = b[i] % sqrt_mod;
    b1[i] = b[i] / sqrt_mod;
  }
  vector<ll> a01(a.size()), b01(b.size());
  for (int i = 0; i < a.size(); i++) {
    a01[i] = a0[i] + a1[i];
    if (a01[i] >= mod) a01[i] -= mod;
  }
  for (int i = 0; i < b.size(); i++) {
    b01[i] = b0[i] + b1[i];
    if (b01[i] >= mod) b01[i] -= mod;
  }
  vector<ll> mid = Mul(a01, b01);
  vector<ll> a0b0 = Mul(a0, b0);
  vector<ll> a1b1 = Mul(a1, b1);
  for (int i = 0; i < mid.size(); i++) {
    mid[i] = (mid[i] - a0b0[i] + mod) % mod;
    mid[i] = (mid[i] - a1b1[i] + mod) % mod;
  }
  vector<ll> res = a0b0;
  for (int i = 0; i < res.size(); i++) {
    res[i] += (sqrt_mod * mid[i]) % mod;
    if (res[i] >= mod) res[i] -= mod;
  }
  sqrt_mod = (sqrt_mod * sqrt_mod) % mod;
  for (int i = 0; i < res.size(); i++) {
    res[i] += (sqrt_mod * a1b1[i]) % mod;
    if (res[i] >= mod) res[i] -= mod;
  }
  return res;
}
```

6.5 Fast Fourier Transformation

```
typedef complex<double> base;

void fft(vector<base> & a, bool invert) {
    int n = (int)a.size();

    for (int i = 1, j = 0; i<n; ++i) {
        int bit = n >> 1;
        for (; j >= bit; bit >>= 1)j -= bit;
        j += bit;
        if (i < j)swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert
            ? -1 : 1);
        base wlen(cos(ang), sin(ang));
        for (int i = 0; i<n; i += len) {
            base w(1);
            for (int j = 0; j<len / 2; ++j) {
                base u = a[i + j], v = a[i +
                    j + len / 2] * w;
                a[i + j] = u + v;
                a[i + j + len / 2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert)for (int i = 0; i<n; ++i)a[i]
        /= n;
}

vector<LL> Mul(vector<LL>& a, vector<LL>& b)
{
    vector<base> fa(a.begin(), a.end()), fb(
        b.begin(), b.end());
    int n = 1;
    while (n < max(a.size(), b.size())) n
        <<= 1;
    n <<= 1;
    fa.resize(n), fb.resize(n);

    fft(fa, false), fft(fb, false);
    for (int i = 0; i<n; ++i)fa[i] *= fb[i];
    fft(fa, true);

    vector<LL> res;
    res.resize(n);
    for (int i = 0; i<n; ++i)res[i] = round(
        fa[i].real());
    return res;
}
/*
n degree Polynomial division: (one of the
    vector)
coefficient of x^i is replaced with
    coefficient of x^(n-i).
x^(i-j) starts from n+1 ends at 2*n-1
*/
/*
All possible sum of 3 different index.
vector<ll>v=Mul(v1,v2);
v=Mul(v,v3);
vector<ll>dbl(v.size()),tri(v.size());
for(ll i=0;i<v1.size();i++){
    dbl[i+i]=v1[i]*v2[i]; // All (i,i) Pairs
    tri[i+i+i]=v1[i]*v2[i]*v3[i]; // All (i,
        i,i) Triplets
}
dbl=Mul(dbl,v3); // All (i,i,j) Triplets
for(ll i=0;i<v.size();i++){
    v[i] = v[i] - (3 * dbl[i] - 2 * tri[i]);
         // 3 (i,i,j) Triplets have 3 (i,i,
        i) triplets. So Remove 2 (i,i,i)
        triplets.
    v[i]/=6; // (i,j,k) can be oriented in
        3! way
    if(v[i])cout << i - 60000 << " : " << v[
        i] << "\n"; // Handle negative
        value
}
*/
int main()
{
    int t;
    cin >> t;
    while(t--){
        string a,b;
        cin >> a >> b;
        vector<LL>v1,v2;

        int sign = 0;
        if(a[0] == '-'){
            sign = 1 - sign;
            a.erase(a.begin());
        }
        if(b[0] == '-'){
            sign = 1 - sign;
```

```
        b.erase(b.begin());
    }

    for(int i = 0;i < a.size();i++){
        int d = a[i] - '0';
        v1.push_back(d);
    }
    for(int i = 0;i < b.size();i++){
        int d = b[i] - '0';
        v2.push_back(d);
    }

    reverse(all(v1)),reverse(all(v2));
     //Reverse needed if v1 is in x^n
     +x^n-1+.....+x^1+1 form
    vector<LL>v = Mul(v1,v2);

    int carry = 0;
    vector<int>answer;
    for(int i = 0;i < v.size();i++){
        int temp = v[i];
        temp += carry;
        answer.push_back(temp % 10);
        carry = temp/10;
    }
    while(answer.size() > 1 and answer.
     back() == 0)answer.pop_back();
    reverse(all(answer));

    for(int i : answer)cout << i;
    cout << "\n";

  }
}
```

6.6 Fast Walsh Hadamord Transformation

```
#define bitwiseXOR 1
//#define bitwiseAND 2
//#define bitwiseOR 3

void FWHT(vector< LL >&p, bool inverse){
    LL n = p.size();
    assert((n&(n-1))==0);

    for (LL len = 1; 2*len <= n; len <<= 1)
     {
        for (LL i = 0; i < n; i += len+len)
         {
            for (LL 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 % MOD;
                    p[i+len+j] = (u+v) % MOD;
                } else {
                    p[i+j] = (-u+v) % MOD;
                    p[i+len+j] = u % MOD;
                }
                #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) {
        //LL val=BigMod(n,MOD-2); //Option
         2: Exclude
        for (LL i = 0; i < n; i++) {
            //assert(p[i]%n==0); //Option 2:
             Include
            //p[i] = (p[i]*val)%MOD; //Option
             2: p[i]/=n;
            p[i]/=n;
        }
    }
    #endif // bitwiseXOR
}

vector<pair<int,int> >V[100005];
int dis[100005];

void dfs(int s,int pr){
    for(auto p:V[s]){
        if(p.first==pr) continue;
        dis[p.first]=dis[s]^p.second;
        dfs(p.first,s);
    }
}

int main(){
    int t;
    cin >> t;
    const int len=(1<<16);
    for(int tc=1;tc<=t;tc++){
        LL n;
        cin >> n;
        for(int i=1;i<=n-1;i++){
            int u,v,w;
            cin >> u >> v >> w;
            V[u].push_back({v,w});
            V[v].push_back({u,w});
        }
        dfs(1,0);
        vector<LL>a(len,0);
        for(int i=1;i<=n;i++) a[dis[i]]++;
        FWHT(a,false);
        for(int i=0;i<len;i++) a[i]*=a[i];
        FWHT(a,true);
        a[0]-=n;
        cout << "Case " << tc << ":\n";
        for(int i=0;i<len;i++) cout << a[i
         ]/2 << '\n';
        memset(dis,0,sizeof(dis));
        for(int i=1;i<=n;i++) V[i].clear();
    }
}
```

6.7 Gaussian Elimination

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually
     have to be infinity or a big number

template <typename DT> int gauss (vector <
     vector<DT> > a, vector<DT> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n;
     ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel
             ][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                DT c = a[i][col] / a[row][col
                 ];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where
             [i]][i];
    for (int i=0; i<n; ++i) {
        DT sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}

int compute_rank(vector<vector<double>> A) {
    int n = A.size();
    int m = A[0].size();

    int rank = 0;
    vector<bool> row_selected(n, false);
    for (int i = 0; i < m; ++i) {
        int j;
        for (j = 0; j < n; ++j) {
            if (!row_selected[j] && abs(A[j][
             i]) > EPS)
                break;
        }

        if (j != n) {
            ++rank;
            row_selected[j] = true;
            for (int p = i + 1; p < m; ++p)
                A[j][p] /= A[j][i];
            for (int k = 0; k < n; ++k) {
                if (k != j && abs(A[k][i]) >
                 EPS) {
                    for (int p = i + 1; p < m;
                     ++p)
                        A[k][p] -= A[j][p] * A
                         [k][i];
                }
            }
        }
    }
    return rank;
}
```

6.8 Gradient Descent

```
/*
Given n 3D point. Find a point from where
     distance to farthest of those n point
     is minimum.
*/
int main(){
    int n;
    cin >> n;
    double x[n],y[n],z[n];
    REP(i,n) cin >> x[i] >> y[i] >> z[i];
    double px=0.0,py=0.0,pz=0.0,alpha=1.0;
    REP(loop,100000){
        int idx=0;
        double dis=-1.0;
        REP(i,n){
            double tmp=SQ(x[i]-px)+SQ(y[i]-py
             )+SQ(z[i]-pz);
            if(tmp>dis) {
                dis=tmp;
                idx=i;
            }
        }
        px+=alpha*(x[idx]-px);
        py+=alpha*(y[idx]-py);
        pz+=alpha*(z[idx]-pz);
        alpha*=0.999;
    }
```

```
    cout << px << ' ' << py << ' ' << pz;
}
```

6.9 Green Hackenbush on Trie

```
int trie[40 * MAX][26];
int XOR[40 * MAX][26];
int valu[40 * MAX];
int node = 1;

int add(string s) {
  int now = 1;
  stack<int>st;
  for (int i = 0; i < s.size(); i++) {
    int c = s[i] - 'a';
    if (!trie[now][c]) trie[now][c] = ++node
        ;
    st.push(now);
    now = trie[now][c];
  }

  int nxt = now;
  int nxt_val = 0;
  for (int i = 0; i < 26; i++) nxt_val ^=
      XOR[now][i];
  while (!st.empty()) {
    now = st.top();
    st.pop();
    int val = 0;
    for (int i = 0; i < 26; i++) {
      if (trie[now][i] == nxt) {
        XOR[now][i] = nxt_val + 1;
      }
      val ^= XOR[now][i];
    }
    nxt_val = val;
    nxt = now;
  }
  return nxt_val;
}
```

6.10 Grid Nim

```
int r,c;
scanf("%d %d",&r,&c);
int nim=0;
FOR(i,1,r){
    FOR(j,1,c){
        int tmp;
        scanf("%d",&tmp);
        if(((r-i)+(c-j))%2){
            nim^=tmp;
        }
    }
}
if(nim) printf("Case %d: win\n",tc);
else printf("Case %d: lose\n",tc);
```

6.11 Linear Seive with Multiplicative Functions

```
const int maxn = 1e7;
vector <int> primes;
int spf[maxn+5], phi[maxn+5], NOD[maxn+5],
     cnt[maxn+5], POW[maxn+5], SOD[maxn+5];
bool prime[maxn+5];

void sieve(){
    fill(prime+2, prime+maxn+1, 1);
    SOD[1] = NOD[1] = phi[1] = spf[1] = 1;
    for(ll i=2;i<=maxn;i++){
        if(prime[i]) {
            primes.push_back(i), spf[i] = i;
            phi[i] = i-1;
            NOD[i] = 2, cnt[i] = 1;
            SOD[i] = i+1, POW[i] = i;
        }
        for(auto p:primes){
            if(p*i>maxn or p > spf[i]) break;
            prime[p*i] = false, spf[p*i] = p;
            if(i%p == 0){
                phi[p*i]=p*phi[i];
                NOD[p*i]=NOD[i]/(cnt[i]+1)*(
                     cnt[i]+2), cnt[p*i]=cnt[
                     i]+1;
                SOD[p*i]=SOD[i]/SOD[POW[i]]*(
                     SOD[POW[i]]+p*POW[i]),
                     POW[p*i]=p*POW[i];
                break;
            } else {
                phi[p*i]=phi[p]*phi[i];
                NOD[p*i]=NOD[p]*NOD[i], cnt[p
                     *i]=1;
                SOD[p*i]=SOD[p]*SOD[i], POW[p
                     *i]=p;
            }

        }
    }
}
```

6.12 Matrix Exponentiation

```
struct matrix {
  vector<vector<ll>> mat;
  int n, m;
  matrix() {}
  matrix(int n, int m) : n(n), m(m), mat(n)
      {
    for (int i = 0; i < n; i++) mat[i] =
        vector<ll>(m);
  }

  void identity() { for (int i = 0; i < n; i
      ++) mat[i][i] = 1; }
  void print() {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) cout <<
          mat[i][j] << " ";
      cout << "\n";
    }
  }
  vector<ll> &operator[](int i) {
    return mat[i];
  }
};

// make sure a.m == b.n
matrix operator * (matrix &a, matrix &b) {
  int n = a.n, m = b.m;
  matrix ret(n, m);
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      for (int k = 0; k < a.m; k++) {
        ll val = (1ll * a[i][k] * b[k][j]) %
            MOD;
        ret[i][j] = (ret[i][j] + val) % MOD;
      }
    }
  }
  return ret;
}

matrix mat_exp(matrix &mat, ll p) {
  int n = mat.n, m = mat.m;
  matrix ret(n, m);
  ret.identity();
  matrix x = mat;

  while (p) {
    if (p & 1) ret = ret * x;
    x = x * x;
    p = p >> 1;
  }
  return ret;
}
```

6.13 Mobius Function

```
const int N=1000001;
int mu[N];
void mobius(){
    MEM(mu,-1);
    mu[1]=1;
    for(int i = 2; i<N; i++){
        if(mu[i]){
            for(int j = i+i; j<N; j += i) mu[
                j] -= mu[i];
        }
    }
}
```

6.14 Modular Binomial Coefficients

```
const int N = 2e5+5;
const int mod = 1e9+7;

array <int, N+1> fact, inv, inv_fact;
void init(){
    fact[0] = inv_fact[0] = 1;
    for(int i=1; i<=N; i++){
        inv[i] = i == 1 ? 1 : (LL) inv[i -
            mod%i] * (mod/i + 1) % mod;
        fact[i] = (LL) fact[i-1] * i % mod;
        inv_fact[i] = (LL) inv_fact[i-1] *
            inv[i] % mod;
    }
}
int C(int n,int r){
    if(fact[0] != 1) init();
    return (r < 0 or r > n) ? 0 : (LL) fact[
        n]*inv_fact[r] % mod * inv_fact[n-r
        ] % mod;
}
```

6.15 Number Theoretic Transformation

```
#define pii   pair<LL,LL>
const LL N= 1<<18;
const LL MOD=786433;

vector<LL>P[N];

LL rev[N],w[N|1],a[N],b[N],inv_n,g;

LL Pow(LL b,LL p){
    LL ret=1;
    while(p){
        if(p & 1) ret=(ret*b)%MOD;
        b=(b*b)%MOD;
        p>>=1;
    }
    return ret;
}

LL primitive_root(LL p){
    vector<LL>factor;
    LL phi = p-1,n=phi;

    for(LL i=2;i*i<=n;i++){
        if(n%i) continue;
        factor.emplace_back(i);
        while(n%i==0) n/=i;
    }

    if(n>1) factor.emplace_back(n);
    for(LL res=2;res<=p;res++){
        bool ok=true;
        for(LL i=0;i<factor.size() && ok;i
            ++) ok &= Pow(res,phi/factor[i])
             != 1;
        if(ok) return res;
    }
    return -1;
}

void prepare(LL n){
    LL sz=abs(31-__builtin_clz(n));
    LL r=Pow(g,(MOD-1)/n);
    inv_n=Pow(n,MOD-2);
    w[0]=w[n]=1;
    for(LL i=1;i<n;i++) w[i]= (w[i-1]*r)%MOD
        ;
    for(LL i=1;i<n;i++) rev[i]=(rev[i
        >>1]>>1) | ((i & 1)<<(sz-1));
}

void NTT(LL *a,LL n,LL dir=0){
    for(LL i=1;i<n-1;i++) if(i<rev[i]) swap(
        a[i],a[rev[i]]);
    for(LL m=2;m<=n;m <<= 1) {
```

```
        for(LL i=0;i<n;i+=m){
            for(LL j=0;j< (m>>1);j++){
                LL &u=a[i+j],&v=a[i+j+(m>>1)
                    ];
                LL t=v*w[dir ? n-n/m*j:n/m*j
                    ]%MOD;
                v=u-t<0?u-t+MOD:u-t;
                u=u+t>=MOD?u+t-MOD:u+t;
            }
        }
    }
    if(dir) for(LL i=0;i<n;i++) a[i]=(inv_n*
        a[i])%MOD;
}

vector<LL> mul(vector<LL>p,vector<LL>q){
    LL n=p.size(),m=q.size();
    LL t=n+m-1,sz=1;
    while(sz<t) sz <<= 1;
    prepare(sz);

    for(LL i=0;i<n;i++) a[i]=p[i];
    for(LL i=0;i<m;i++) b[i]=q[i];

    for(LL i=n;i<sz;i++) a[i]=0;
    for(LL i=m;i<sz;i++) b[i]=0;

    NTT(a,sz);
    NTT(b,sz);
    for(LL i=0;i<sz;i++) a[i]=(a[i]*b[i])%
        MOD;
    NTT(a,sz,1);

    vector<LL> c(a,a+sz);
    while(c.size() && c.back()==0) c.
        pop_back();
    return c;
}

/*
N different number box
Number of ways to make a number by picking
    any number from any of the boxes
*/

vector<LL> solve(LL l,LL r){
    if(l==r) return P[l];
    LL m=(l+r)/2;
    return mul(solve(l,m),solve(m+1,r));
}

int main(){
    LL m;
    cin >> m;
    for(LL i=1;i<=m;i++){
        LL num;
        cin >> num;
        vector<pii>v;
        LL mx=0;
        while(num--){
            LL typ,cnt;
            cin >> typ >> cnt;
            v.emplace_back(typ,cnt);
            mx=max(mx,typ);
        }
        P[i].resize(mx+1);
        for(pii p:v) P[i][p.first]=p.second;
    }
    g=primitive_root(MOD);
    vector<LL>c=solve(1,m);
    for(LL i=0;i<c.size();i++){
        if(c[i]){
            cout << i << ' ' << c[i] << '\n';
        }
    }
}
```

6.16 Pollard Rho and Factorization

```
// fast factorize
map <ull,int> fast_factorize(ull n){
    map <ull,int> ans;
    for(;n>1;n/=spf[n])
        ans[spf[n]]++;
    return ans;
}
inline ULL mul(ULL a,ULL b,ULL mod){
    LL ans = a * b - mod * (ULL) (1.L / mod
        * a * b);
    return ans + mod * (ans < 0) - mod * (
        ans >= (LL) mod);
}
inline ULL bigmod(ULL num,ULL pow,ULL mod){
    ULL ans = 1;
    for( ; pow > 0; pow >>= 1, num = mul(num
        , num, mod))
        if(pow&1) ans = mul(ans,num,mod);
    return ans;
}
inline bool is_prime(ULL n){
    if(n < 2 or n % 6 % 4 != 1)
        return (n|1) == 3;
    ULL a[] = {2, 325, 9375, 28178, 450775,
        9780504, 1795265022};
    ULL s = __builtin_ctzll(n-1), d = n >> s
        ;
    for(ULL x: a){
        ULL p = bigmod(x % n, d, n), i = s;
        for( ; p != 1 and p != n-1 and x % n
            and i--; p = mul(p, p, n));
        if(p != n-1 and i != s)
            return false;
    }
    return true;
}
ULL get_factor(ULL n) {
    auto f = [&](LL x) { return mul(x, x, n)
        + 1; };
    ULL x = 0, y = 0, t = 0, prod = 2, i =
        2, q;
    for( ; t++ %40 or gcd(prod, n) == 1; x =
        f(x), y = f(f(y)) ){
        (x == y) ? x = i++, y = f(x) : 0;
        prod = (q = mul(prod, max(x,y) - min
            (x,y), n)) ? q : prod;
    }
    return gcd(prod, n);
}
map <ULL, int> factorize(ULL n){
    map <ULL, int> res;
    if(n < 2)  return res;
    ULL small_primes[] = {2, 3, 5, 7, 11,
        13, 17, 19, 23, 29, 31, 37, 41, 43,
        47, 53, 59, 61, 67, 71, 73, 79,
        83, 89, 97 };
    for (ULL p: small_primes)
        for( ; n % p == 0; n /= p, res[p]++)
            ;

    auto _factor = [&](ULL n, auto &_factor)
        {
        if(n == 1)  return;
        if(is_prime(n))
            res[n]++;
        else {
            ULL x = get_factor(n);
            _factor(x, _factor);
            _factor(n / x, _factor);
        }
    };
    _factor(n, _factor);
    return res;
}
```

6.17 Prime Counting Function

```
// initialize once by calling init()
#define MAXN 20000010    // initial sieve
    limit
#define MAX_PRIMES 2000010 // max size of
    the prime array for sieve
#define PHI_N 100000
#define PHI_K 100

int len = 0; // total number of primes
    generated by sieve
int primes[MAX_PRIMES];
int pref[MAXN];      // pref[i] --> number
    of primes <= i
int dp[PHI_N][PHI_K]; // precal of yo(n,k)
bitset<MAXN> f;
void sieve(int n) {
    f[1] = true;
    for (int i = 4; i <= n; i += 2) f[i] =
        true;
    for (int i = 3; i * i <= n; i += 2) {
        if (!f[i]) {
            for (int j = i * i; j <= n; j +=
                i << 1) f[j] = 1;
        }
    }
    for (int i = 1; i <= n; i++) {
        if (!f[i]) primes[len++] = i;
        pref[i] = len;
    }
}
void init() {
    sieve(MAXN - 1);
    // precalculation of phi upto size (
        PHI_N,PHI_K)
    for (int n = 0; n < PHI_N; n++) dp[n][0]
        = n;
    for (int k = 1; k < PHI_K; k++) {
        for (int n = 0; n < PHI_N; n++) {
            dp[n][k] = dp[n][k - 1] - dp[n /
                primes[k - 1]][k - 1];
        }
    }
}
// returns the number of integers less or
    equal n which are
// not divisible by any of the first k
    primes
// recurrence --> yo(n, k) = yo(n, k-1) - yo
    (n / p_k , k-1)
// for sum of primes yo(n, k) = yo(n, k-1) -
    p_k * yo(n / p_k , k-1)
long long yo(long long n, int k) {
    if (n < PHI_N && k < PHI_K) return dp[n
        ][k];
    if (k == 1) return ((++n) >> 1);
    if (primes[k - 1] >= n) return 1;
    return yo(n, k - 1) - yo(n / primes[k -
        1], k - 1);
}
// complexity: n^(2/3).(log n^(1/3))
long long Legendre(long long n) {
    if (n < MAXN) return pref[n];
    int lim = sqrt(n) + 1;
    int k = upper_bound(primes, primes + len
        , lim) - primes;
    return yo(n, k) + (k - 1);
}
// runs under 0.2s for n = 1e12
long long Lehmer(long long n) {
    if (n < MAXN) return pref[n];
    long long w, res = 0;
    int b = sqrt(n), c = Lehmer(cbrt(n)), a
        = Lehmer(sqrt(b));
    b = Lehmer(b);
    res = yo(n, a) + ((1LL * (b + a - 2) * (
        b - a + 1)) >> 1);
    for (int i = a; i < b; i++) {
        w = n / primes[i];
        int lim = Lehmer(sqrt(w));
        res -= Lehmer(w);
        if (i <= c) {
            for (int j = i; j < lim; j++) {
                res += j;
                res -= Lehmer(w / primes[j]);
            }
        }
    }
    return res;
}
```

6.18 Seive Upto 1e9

```
// credit: min_25
// takes 0.5s for n = 1e9
```

```
vector<int> sieve(const int N, const int Q =
     17, const int L = 1 << 15) {
  static const int rs[] = {1, 7, 11, 13, 17,
     19, 23, 29};
  struct P {
    P(int p) : p(p) {}
    int p; int pos[8];
  };
  auto approx_prime_count = [] (const int N)
       -> int {
    return N > 60184 ? N / (log(N) - 1.1)
                 : max(1., N / (log(N) -
                      1.11)) + 1;
  };

  const int v = sqrt(N), vv = sqrt(v);
  vector<bool> isp(v + 1, true);
  for (int i = 2; i <= vv; ++i) if (isp[i])
       {
    for (int j = i * i; j <= v; j += i) isp[
         j] = false;
  }

  const int rsize = approx_prime_count(N +
       30);
  vector<int> primes = {2, 3, 5}; int psize
       = 3;
  primes.resize(rsize);

  vector<P> sprimes; size_t pbeg = 0;
  int prod = 1;
  for (int p = 7; p <= v; ++p) {
    if (!isp[p]) continue;
    if (p <= Q) prod *= p, ++pbeg, primes[
         psize++] = p;
    auto pp = P(p);
    for (int t = 0; t < 8; ++t) {
      int j = (p <= Q) ? p : p * p;
      while (j % 30 != rs[t]) j += p << 1;
      pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
  }

  vector<unsigned char> pre(prod, 0xFF);
  for (size_t pi = 0; pi < pbeg; ++pi) {
    auto pp = sprimes[pi]; const int p = pp.
         p;
    for (int t = 0; t < 8; ++t) {
      const unsigned char m = ~(1 << t);
      for (int i = pp.pos[t]; i < prod; i +=
           p) pre[i] &= m;
    }
  }

  const int block_size = (L + prod - 1) /
       prod * prod;
  vector<unsigned char> block(block_size);
       unsigned char* pblock = block.data();
  const int M = (N + 29) / 30;

  for (int beg = 0; beg < M; beg +=
       block_size, pblock -= block_size) {
    int end = min(M, beg + block_size);
    for (int i = beg; i < end; i += prod) {
      copy(pre.begin(), pre.end(), pblock +
           i);
    }
    if (beg == 0) pblock[0] &= 0xFE;
    for (size_t pi = pbeg; pi < sprimes.size
         (); ++pi) {
      auto& pp = sprimes[pi];
      const int p = pp.p;
      for (int t = 0; t < 8; ++t) {
        int i = pp.pos[t]; const unsigned
             char m = ~(1 << t);
        for (; i < end; i += p) pblock[i] &=
             m;
        pp.pos[t] = i;
      }
    }
    for (int i = beg; i < end; ++i) {
      for (int m = pblock[i]; m > 0; m &= m
           - 1) {
          primes[psize++] = i * 30 + rs[
               __builtin_ctz(m)];
      }
    }
  }
  assert(psize <= rsize);
  while (psize > 0 && primes[psize - 1] > N)
       --psize;
  primes.resize(psize);
  return primes;
}

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, a, b; cin >> n >> a >> b;
  auto primes = sieve(n);
  vector<int> ans;
  for (int i = b; i < primes.size() &&
       primes[i] <= n; i += a) ans.push_back
       (primes[i]);
  cout << primes.size() << ' ' << ans.size()
       << '\n';
  for (auto x: ans) cout << x << ' '; cout
       << '\n';
  return 0;
}
```

6.19 Simpson Integration

```
/*
    For finding the length of an arc in a
         range
    L = integrate(ds) from start to end of
         range
    where ds = sqrt(1+(d/dy(x))^2)dy
*/
const double SIMPSON_TERMINAL_EPS = 1e-12;
/// Function whose integration is to be
     calculated
double F(double x);
double simpson(double minx, double maxx)
{
    return (maxx - minx) / 6 * (F(minx) + 4
         * F((minx + maxx) / 2.) + F(maxx));
}
double adaptive_simpson(double minx, double
     maxx, double c, double EPS)
{
//    if(maxx - minx < SIMPSON_TERMINAL_EPS)
      return 0;

    double midx = (minx + maxx) / 2;
    double a = simpson(minx, midx);
    double b = simpson(midx, maxx);

    if(fabs(a + b - c) < 15 * EPS) return a
         + b + (a + b - c) / 15.0;

    return adaptive_simpson(minx, midx, a,
         EPS / 2.) + adaptive_simpson(midx,
         maxx, b, EPS / 2.);
}
double adaptive_simpson(double minx, double
     maxx, double EPS)
{
    return adaptive_simpson(minx, maxx,
         simpson(minx, maxx, i), EPS);
}
```

6.20 Sqrt Field

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

const int M = 1e9+7;
typedef long long LL;

LL mod(LL x) {
    LL ans = x%M;
    if (ans < 0)   ans += M;
    return ans;
}

template <LL X>
struct SqrtField {
    LL a, b, c;    /// (a + b*sqrt(X))/c;
    SqrtField(LL A=0, LL B=0, LL C=1) : a(A)
         , b(B), c(C) {}


    SqrtField operator+(const SqrtField &y)
         {
        return SqrtField(mod(a*y.c + y.a*c),
             mod(b*y.c + y.b*c), mod(c*y.c))
             ;
    }

    SqrtField operator-(const SqrtField &y)
         {
        return SqrtField(mod(a*y.c - y.a*c),
             mod(b*y.c - y.b*c), mod(c*y.c))
             ;
    }

    SqrtField operator*(const SqrtField &y)
         {
        return SqrtField(mod(a*y.a + X*y.b*b
            ), mod(a*y.b + b*y.a), mod(c*y.c
            ));
    }

    SqrtField operator/(const SqrtField &y)
         {
        LL A = mod(a*y.a - X*y.b*b);
        LL B = mod(b*y.a - a*y.b);
        LL C = mod(y.a*y.a - X*y.b*y.b);
        A = mod(A*y.c);
        B = mod(B*y.c);
        C = mod(C*c);
        return SqrtField(A,B,C);
    }
};

template<LL X>
ostream& operator<<(ostream &os, const
     SqrtField<X> &x) {
    return os<<"("<<x.a<<"+"<<x.b<<"V"<<X<<"
         )/"<<x.c;
}

int main() {
    SqrtField<2> c(5);          ///5
    SqrtField<2> b(0, 1);       ///3 sqrt(2)
    SqrtField<2> a(3,7,2);      ///(3+7*sqrt(2)
         )/2

    cout<<a+b<<" "<<a-b<<" "<<a*b<<" "<<a/c
         <<endl;
    cout<<a*2<<" "<<a/2<<" "<<a+2<<" "<<a
         -1<<endl;
}
```

6.21 Stirling Numbers

```
//stirling number 2nd kind variation(number
     of ways to place n marbles in k boxes
     so that each box has at least x marbles
     )
ll solve(int marble, int box) {
  if (marble < 1ll * box * x) return 0;
  if (box == 1 && marble >= x) return 1;
  if (vis[marble][box] == cs) return dp[
       marble][box];
  vis[marble][box] = cs;
  ll a = ( 1ll * box * solve(marble - 1, box
       ) ) % MOD;
  ll b = ( 1ll * box * ncr(marble - 1, x -
       1) ) % MOD;
  b = (b * solve(marble - x, box - 1)) % MOD
       ;
  ll ret = (a + b) % MOD;
  return dp[marble][box] = ret;
}
//number of ways to place n marbles in k
     boxes so that no box is empty
ll stir(ll n, ll k) {
  ll ret = 0;
```

```
  for (int i = 0; i <= k; i++) {
    ll v = ncr(k, i) * bigmod(i, n) % MOD;
    if ( (k - i) % 2 == 0 ) ret = (ret + v)
         % MOD;
    else ret = (ret - v + MOD) % MOD;
  }
  return ret;
}
```

7 String Algorithms

7.1 Aho Corasick

```
struct state {
  int len, par, link, next_lif;
  ll val;
  int next[26];
  char p_char;
  bool lif;
  state(int par = -1, char p_char = '$', int
        len = 0) : par(par), p_char(p_char),
        len(len) {
    lif = false;
    link = 0;
    next_lif = 0;
    val = 0;
    memset(next, 0, sizeof next);
  }
};
vector<state>aho;
inline void add_str(const string &s, ll val)
     {
  int now = 0;
  for (int i = 0; i < s.size(); i++) {
    int c = s[i] - 'a';
    if (!aho[now].next[c]) {
      aho[now].next[c] = (int)aho.size();
      aho.emplace_back(now, s[i], aho[now].
           len + 1);
    }
    now = aho[now].next[c];
  }
  aho[now].lif = true;
  aho[now].val = val;
}
inline void push_link() {
  queue<int>q;
  q.push(0);
  while (!q.empty()) {
    int cur = q.front();
    int link = aho[cur].link;
    q.pop();

    aho[cur].next_lif = aho[link].lif ? link
         : aho[link].next_lif;
    for (int c = 0; c < 26; c++) {
      if (aho[cur].next[c]) {
        aho[ aho[cur].next[c] ].link = cur ?
             aho[link].next[c] : 0;
        q.push( aho[cur].next[c] );
      } else aho[cur].next[c] = aho[link].
           next[c];
    }
    aho[cur].val += aho[link].val;
  }
}
inline int count(string &s) {
  int now = 0, ret = 0;
  for (int i = 0; i < s.size(); i++) {
    now = aho[now].next[s[i] - 'a'];
    ret += aho[now].val;
  }
  return ret;
}
struct dynamic_aho {
  aho_corasick ac[20];
  vector<string> dict[20];
  dynamic_aho() {
    for (int i = 0; i < 20; i++) {
      ac[i].aho.clear();
      dict[i].clear();
    }
  }
  void add_str(string &s) {
    int idx = 0;
    for (; idx < 20 && !ac[idx].aho.empty();
         idx++) {}
    ac[idx] = aho_corasick();
    ac[idx].add_str(s, 1), dict[idx].pb(s);
    for (int i = 0; i < idx; i++) {
      for (string x : dict[i]) {
        ac[idx].add_str(x, 1);
        dict[idx].pb(x);
      }
      ac[i].aho.clear(), dict[i].clear();
    }
    ac[idx].push_link();
  }
  inline int count(string &s) {
    int ret = 0;
    for (int i = 0; i < 20; i++) {
      if (!ac[i].aho.empty()) ret += ac[i].
           count(s);
    }
    return ret;
  }
};
int arr[MAX];
int main() {
  fastio;
  dynamic_aho add, del;
  int m;
  cin >> m;
  while (m--) {
    int type;
    string s;
    cin >> type >> s;
    if (type == 1) add.add_str(s);
    else if (type == 2) del.add_str(s);
    else cout << add.count(s) - del.count(s)
         << "\n" << flush;
  }
}
```

7.2 Double Hash

```
ostream& operator << (ostream& os, pll hash)
     {
  return os << "(" << hash.ff << ", " <<
       hash.ss << ")";
}

pll operator + (pll a, ll x)  {return pll(a.
     ff + x, a.ss + x);}
pll operator - (pll a, ll x)  {return pll(a.
     ff - x, a.ss - x);}
pll operator * (pll a, ll x)  {return pll(a.
     ff * x, a.ss * x);}
pll operator + (pll a, pll x) {return pll(a.
     ff + x.ff, a.ss + x.ss);}
pll operator - (pll a, pll x) {return pll(a.
     ff - x.ff, a.ss - x.ss);}
pll operator * (pll a, pll x) {return pll(a.
     ff * x.ff, a.ss * x.ss);}
pll operator % (pll a, pll m) {return pll(a.
     ff % m.ff, a.ss % m.ss);}

pll base(1949313259, 1997293877);
pll mod(2091573227, 2117566807);

pll power (pll a, ll p) {
  if (!p) return pll(1, 1);
  pll ans = power(a, p / 2);
  ans = (ans * ans) % mod;
  if (p % 2) ans = (ans * a) % mod;
  return ans;
}

pll inverse(pll a) {
  return power(a, (mod.ff - 1) * (mod.ss -
       1) - 1);
}

pll inv_base = inverse(base);

pll val;
vector<pll> P;
void hash_init(int n) {
  P.resize(n + 1);
  P[0] = pll(1, 1);
  for (int i = 1; i <= n; i++) P[i] = (P[i -
       1] * base) % mod;
}

///appends c to string
pll append(pll cur, char c) {
  return (cur * base + c) % mod;
}

///prepends c to string with size k
pll prepend(pll cur, int k, char c) {
  return (P[k] * c + cur) % mod;
}

///replaces the i-th (0-indexed) character
     from right from a to b;
pll replace(pll cur, int i, char a, char b)
     {
  cur = (cur + P[i] * (b - a)) % mod;
  return (cur + mod) % mod;
}

///Erases c from the back of the string
pll pop_back(pll hash, char c) {
  return (((hash - c) * inv_base) % mod +
       mod) % mod;
}

///Erases c from front of the string with
     size len
pll pop_front(pll hash, int len, char c) {
  return ((hash - P[len - 1] * c) % mod +
       mod) % mod;
}

///concatenates two strings where length of
     the right is k
pll concat(pll left, pll right, int k) {
  return (left * P[k] + right) % mod;
}

///Calculates hash of string with size len
     repeated cnt times
///This is O(log n). For O(1), pre-calculate
      inverses
pll repeat(pll hash, int len, ll cnt) {
  pll mul = (P[len * cnt] - 1) * inverse(P[
       len] - 1);
  mul = (mul % mod + mod) % mod;
  pll ret = (hash * mul) % mod;

  if (P[len].ff == 1) ret.ff = hash.ff * cnt
       ;
  if (P[len].ss == 1) ret.ss = hash.ss * cnt
       ;
  return ret;
}

ll get(pll hash) {
  return ( (hash.ff << 32) ^ hash.ss );
}

struct hashlist {
  int len;
  vector<pll> H, R;

  hashlist() {}
  hashlist(string &s) {
    len = (int)s.size();
    hash_init(len);
    H.resize(len + 1, pll(0, 0)), R.resize(
         len + 2, pll(0, 0));
    for (int i = 1; i <= len; i++) H[i] =
         append(H[i - 1], s[i - 1]);
    for (int i = len; i >= 1; i--) R[i] =
         append(R[i + 1], s[i - 1]);
  }

  /// 1-indexed
  inline pll range_hash(int l, int r) {
    int len = r - l + 1;
```

```
    return ((H[r] - H[l - 1] * P[len]) % mod
          + mod) % mod;
  }

  inline pll reverse_hash(int l, int r) {
    int len = r - l + 1;
    return ((R[l] - R[r + 1] * P[len]) % mod
          + mod) % mod;
  }

  inline pll concat_range_hash(int l1, int
        r1, int l2, int r2) {
    int len_2 = r2 - l2 + 1;
    return concat(range_hash(l1, r1),
          range_hash(l2, r2), len_2);
  }

  inline pll concat_reverse_hash(int l1, int
        r1, int l2, int r2) {
    int len_1 = r1 - l1 + 1;
    return concat(reverse_hash(l2, r2),
          reverse_hash(l1, r1), len_1);
  }
};
```

7.3 Faster Hash Table

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct custom_hash {
  const ll rnd = chrono::
      high_resolution_clock::now().
      time_since_epoch().count();
  static unsigned long long hash_f(unsigned
      long long x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9
      ;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb
      ;
    return x ^ (x >> 31);
  }
  ll operator() (ll x) const { return hash_f
      (x) ^ rnd; }
  // static int combine_hash(pii hash) {
      return hash.ff * 31 + hash.ss; }
  // static ll combine_hash(pll hash) {
      return (hash.ff << 32) ^ hash.ss; }
  // ll operator() (pll x) const {
  //   x.ff = hash_f(x.ff) ^ rnd; x.ss =
      hash_f(x.ss) ^ rnd;
  //   return combine_hash(x);
  // }
};
gp_hash_table<ll, int, custom_hash> mp;
unordered_map<ll, int, custom_hash> mp;
```

7.4 Knuth Morris Pratt

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

7.5 Manacher

```
vector<int> d1(n); ///odd palindromes
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 1 : min(d1[l + r - i], r
      - i + 1);
  while (0 <= i - k && i + k < n && s[i - k]
      == s[i + k]) {
    k++;
  }
  d1[i] = k--;
  if (i + k > r) {
    l = i - k;
    r = i + k;
  }
}

vector<int> d2(n); ///even palindromes
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 0 : min(d2[l + r - i +
      1], r - i + 1);
  while (0 <= i - k - 1 && i + k < n && s[i
      - k - 1] == s[i + k]) {
    k++;
  }
  d2[i] = k--;
  if (i + k > r) {
    l = i - k - 1;
    r = i + k ;
  }
}
```

7.6 Palindromic Tree

```
struct state {
  int len, link;
  map<char, int> next;
};
state st[MAX];
int id, last;
string s;
ll ans[MAX];
void init() {
  for (int i = 0; i <= id; i++) {
    st[i].len = 0; st[i].link = 0;
    st[i].next.clear(); ans[i] = 0;
  }
  st[1].len = -1; st[1].link = 1;
  st[2].len = 0; st[2].link = 1;
  id = 2; last = 2;
}
void extend(int pos) {
  while (s[pos - st[last].len - 1] != s[pos
      ]) last = st[last].link;
  int newlink = st[last].link;
  char c = s[pos];
  while (s[pos - st[newlink].len - 1] != s[
      pos]) newlink = st[newlink].link;
  if (!st[last].next.count(c)) {
    st[last].next[c] = ++id;
    st[id].len = st[last].len + 2;
    st[id].link = (st[id].len == 1 ? 2 : st[
        newlink].next[c]);
    ans[id] += ans[st[id].link];
    if (st[id].len > 2) {
      int l = st[id].len / 2 + (st[id].len %
          2 ? 1 : 0);
      if (h.range_hash(pos - st[id].len + 1,
          pos - st[id].len + l) == h.
          reverse_hash(pos - st[id].len +
          1, pos - st[id].len + l)) ans[id
          ]++;
    }
  }
  last = st[last].next[c];
}
```

7.7 String Match FFT

```
//find occurrences of t in s where '?'s are
      automatically matched with any
      character
//res[i + m - 1] = sum_j=0 to m - 1_{s[i + j
      ] * t[j] * (s[i + j] - t[j])
vector<int> string_matching(string &s,
    string &t) {
  int n = s.size(), m = t.size();
  vector<int> s1(n), s2(n), s3(n);
  for(int i = 0; i < n; i++) s1[i] = s[i] ==
      '?' ? 0 : s[i] - 'a' + 1; //assign
      any non zero number for non '?'s
  for(int i = 0; i < n; i++) s2[i] = s1[i] *
      s1[i];
  for(int i = 0; i < n; i++) s3[i] = s1[i] *
      s2[i];
  vector<int> t1(m), t2(m), t3(m);
  for(int i = 0; i < m; i++) t1[i] = t[i] ==
      '?' ? 0 : t[i] - 'a' + 1;
  for(int i = 0; i < m; i++) t2[i] = t1[i] *
      t1[i];
  for(int i = 0; i < m; i++) t3[i] = t1[i] *
      t2[i];
  reverse(t1.begin(), t1.end());
  reverse(t2.begin(), t2.end());
  reverse(t3.begin(), t3.end());
  vector<int> s1t3 = multiply(s1, t3);
  vector<int> s2t2 = multiply(s2, t2);
  vector<int> s3t1 = multiply(s3, t1);
  vector<int> res(n);
  for(int i = 0; i < n; i++) res[i] = s1t3[i
      ] - s2t2[i] * 2 + s3t1[i];
  vector<int> oc;
  for(int i = m - 1; i < n; i++) if(res[i]
      == 0) oc.push_back(i - m + 1);
  return oc;
}
```

7.8 Suffix Array

```
vector<vector<int> >c;
vector<int>sort_cyclic_shifts(string const&
    s) {
  int n = s.size();
  const int alphabet = 256;
  vector<int> p(n), cnt(alphabet, 0);
  c.clear(); c.emplace_back(); c[0].resize(n
      );
  for (int i = 0; i < n; i++) cnt[s[i]]++;
  for (int i = 1; i < alphabet; i++) cnt[i]
      += cnt[i - 1];
  for (int i = 0; i < n; i++) p[--cnt[s[i]]]
      = i;
  c[0][p[0]] = 0;
  int classes = 1;
  for (int i = 1; i < n; i++) {
    if (s[p[i]] != s[p[i - 1]]) classes++;
    c[0][p[i]] = classes - 1;
  }
  vector<int> pn(n), cn(n); cnt.resize(n);
  for (int h = 0; (1 << h) < n; h++) {
    for (int i = 0; i < n; i++) {
      pn[i] = p[i] - (1 << h);
      if (pn[i] < 0) pn[i] += n;
    }
    fill(cnt.begin(), cnt.end(), 0);
    /// radix sort
    for (int i = 0; i < n; i++) cnt[c[h][pn[
        i]]]++;
    for (int i = 1; i < classes; i++) cnt[i]
        += cnt[i - 1];
    for (int i = n - 1; i >= 0; i--) p[--cnt
        [c[h][pn[i]]]] = pn[i];
    cn[p[0]] = 0; classes = 1;
    for (int i = 1; i < n; i++) {
      pii cur = {c[h][p[i]], c[h][(p[i] + (1
          << h)) % n]};
      pii prev = {c[h][p[i - 1]], c[h][(p[i
          - 1] + (1 << h)) % n]};
      if (cur != prev) ++classes;
      cn[p[i]] = classes - 1;
    }
    c.push_back(cn);
  }
  return p;
}
vector<int> suffix_array_construction(string
    s) {
  s += "!";
  vector<int> sorted_shifts =
      sort_cyclic_shifts(s);
  sorted_shifts.erase(sorted_shifts.begin())
      ;
  return sorted_shifts;
}
/// compare two suffixes starting at i and j
    with length 2^k
int compare(int i, int j, int n, int k) {
  pii a = {c[k][i], c[k][(i + 1 - (1 << k))
      % n]};
  pii b = {c[k][j], c[k][(j + 1 - (1 << k))
      % n]};
  return a == b ? 0 : a < b ? -1 : 1;
```

```
}
int lcp(int i, int j) {
  int log_n = c.size() - 1;
  int ans = 0;
  for (int k = log_n; k >= 0; k--) {
    if (c[k][i] == c[k][j]) {
      ans += 1 << k;
      i += 1 << k; j += 1 << k;
    }
  }
  return ans;
}
vector<int> lcp_construction(string const& s
    , vector<int> const& p) {
  int n = s.size();
  vector<int> rank(n, 0);
  for (int i = 0; i < n; i++) rank[p[i]] = i
    ;
  int k = 0;
  vector<int> lcp(n, 0);
  for (int i = 0; i < n; i++) {
    if (rank[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = p[rank[i] + 1];
    while (i + k < n && j + k < n && s[i + k
        ] == s[j + k]) k++;
    lcp[rank[i]] = k;
    if (k) k--;
  }
  return lcp;
}
//kth lexicographically smallest substring (
    considering duplicates)
int arr[MAX], lg[MAX], mn[MAX][25];
vector<int>p, lcp;
string s;
int k;

// sparse table for min in lcp goes here

///find the rightmost position where get(l,r
    ) > val
int khoj(int l, int r, int val) {
  int lo = l + 1, hi = r, ret = -1;
  while (lo <= hi) {
    int mid = lo + (hi - lo) / 2;
    if (get(l, mid - 1) > val) {
      ret = mid; lo = mid + 1;
    } else hi = mid - 1;
  }
  return ret;
}
int done[MAX];
int arr[MAX];
int main() {
  fastio;
  cin >> s >> k;
  p = suffix_array_construction(s);
  lcp = lcp_construction(s, p);
  build();
  int n = s.size();
  int milaisi = 0;
  for (int i = 0; i < n; i++) {
    milaisi += done[i];
    int len = n - p[i];
    int cur = milaisi;
    /// cur = i ? lcp[i-1] : 0; this can
        replace all the milaisi and done
        parts
    while (cur < len) {
      int r = khoj(i, n - 1, cur);
      int koyta, milabo;
      if (r == -1) {
        milabo = len - cur;
        koyta = 1;
      } else {
        milabo = get(i, r - 1) - cur;
        koyta = r - i + 1;
      }
      if (koyta * milabo < k) k -= (koyta *
          milabo);
      else {
        int d = k / koyta; int m = k % koyta
            ;
        if (!m) {
          cout << s.substr(p[i], cur + d) <<
              "\n";
          return 0;
        } else {
          cout << s.substr(p[i], cur + d + 1)
              << "\n";
          return 0;
        }
      }
      if (r == -1) break;
      done[r + 1] -= milabo;
      cur = get(i, r - 1);
      milaisi += milabo;
    }
  }
  cout << "No such line.\n";
}
```

7.9 Suffix Automaton

```
struct state {
  int len, link;
  map<char, int> next;
  bool is_clone;
  int first_pos;
  vector<int>inv_link;
};
state st[2 * MAX];
int mn[2 * MAX], mx[2 * MAX];
int sz, last;
void sa_init() {
  st[0].len = 0;
  st[0].link = -1;
  st[0].next.clear();
  sz = 1;
  last = 0;
}
void sa_extend(char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  st[cur].first_pos = st[cur].len - 1;
  st[cur].is_clone = false;
  st[cur].next.clear();
  ///for lcs of n strings
  // mn[cur] = st[cur].len;
  int p = last;
  while (p != -1 && !st[p].next.count(c)) {
    st[p].next[c] = cur;
    p = st[p].link;
  }
  if (p == -1) {
    st[cur].link = 0;
  } else {
    int q = st[p].next[c];
    if (st[p].len + 1 == st[q].len) {
      st[cur].link = q;
    } else {
      int clone = sz++;
      st[clone].len = st[p].len + 1;
      st[clone].next = st[q].next;
      st[clone].link = st[q].link;
      st[clone].first_pos = st[q].first_pos;
      st[clone].is_clone = true;
      ///for lcs of n strings
      // mn[clone] = st[clone].len;
      while (p != -1 && st[p].next[c] == q)
          {
        st[p].next[c] = clone;
        p = st[p].link;
      }
      st[q].link = st[cur].link = clone;
    }
  }
  last = cur;
}
void radix_sort() {
  for (int i = 0; i < sz; i++) cnt[st[i].len
      ]++;
  for (int i = 1; i < sz; i++) cnt[i] += cnt
      [i - 1];
  for (int i = 0; i < sz; i++) order[--cnt[
      st[i].len]] = i;
}
// after constructing the automaton
for (int v = 1; v < sz; v++) {
  st[st[v].link].inv_link.push_back(v);
}
// output all positions of occurrences
void output_all_occurrences(int v, int
    P_length) {
  if (!st[v].is_clone)
    cout << st[v].first_pos - P_length + 1
        << endl;
  for (int u : st[v].inv_link)
    output_all_occurrences(u, P_length);
}
//lcs of two strings
string lcs (string S, string T) {
  sa_init();
  for (int i = 0; i < S.size(); i++)
      sa_extend(S[i]);
  int v = 0, l = 0, best = 0, bestpos = 0;
  for (int i = 0; i < T.size(); i++) {
    while (v && !st[v].next.count(T[i])) {
      v = st[v].link; l = st[v].len;
    }
    if (st[v].next.count(T[i])) {
      v = st[v].next[T[i]]; l++;
    }
    if (l > best) best = l; bestpos = i;
  }
  return T.substr(bestpos - best + 1, best);
}
//lcs of n strings
void add_str(string s) {
  for (int i = 0; i < sz; i++) mx[i] = 0;
  int v = 0, l = 0;
  for (int i = 0; i < s.size(); i++) {
    while (v && !st[v].next.count(s[i])) {
      v = st[v].link; l = st[v].len;
    }
    if (st[v].next.count(s[i])) {
      v = st [v].next[s[i]]; l++;
    }
    mx[v] = max(mx[v], l);
  }
  for (int i = sz - 1; i > 0; i--) mx[st[i].
      link] = max(mx[st[i].link], mx[i]);
  for (int i = 0; i < sz; i++) mn[i] = min(
      mn[i], mx[i]);
}
int lcs() {
  int ret = 0;
  for (int i = 0; i < sz; i++) ret = max(ret
      , mn[i]);
  return ret;
}
string s[15];
int arr[MAX];
int main() {
  fastio;
  int n = 0;
  while (cin >> s[n]) n++;
  sa_init();
  for (int i = 0; i < s[0].size(); i++)
      sa_extend(s[0][i]);
  for (int i = 1; i < n; i++) add_str(s[i]);
  cout << lcs() << "\n";
}
```

7.10 Z Algorithm

```
vector<int> calcz(string s) {
  int n = s.size();
  vector<int> z(n);
  int l, r; l = r = 0;
  for (int i = 1; i < n; i++) {
    if (i > r) {
      l = r = i;
      while (r < n && s[r] == s[r - l]) r++;
      z[i] = r - l; r--;
    } else {
      int k = i - l;
      if (z[k] < r - i + 1) z[i] = z[k];
```

```
      else {
        l = i;
        while (r < n && s[r] == s[r - l]) r
             ++;
        z[i] = r - l; r--;
      }
    }
  }
  return z;
}
```

8 Tricks

8.1 2 Satisfiability

```
/*
  2-Sat Note: Assign true or false values to
       n variables in order to satisfy
  a system of constraints on pairs of
      variables.
  E.g: (x1 or !x2) and (x2 or x3) and (!x3
      or !x3)

  x1 = true
  x2 = true
  x3 = false
  is a solution to make the above formula
      true.
  MAX must be equal to the maximum number of
       variables.
  n passed in init() is the number of
      variables.
  O(V+E)

  !a is represented as neg(a).
  example xor:
  |a|b|
  |0|0| x or(a,b)
  |0|1|
  |1|0|
  |1|1| x or(!a, !b)
  do OR of negation of values of variables
      for each undesired situation
  to make it impossible.
*/

struct two_sat {
  int n, id;
  vector<int> g[2 * MAX], rg[2 * MAX], order
     , st;
  bool state[2 * MAX], vis[2 * MAX];
  int scc[2 * MAX];

  void init(int _n) {
    n = _n;
    for (int i = 0; i <= 2 * n; i++) {
      g[i].clear(), rg[i].clear();
      state[i] = vis[i] = false;
      scc[i] = -1;
    }
    st.clear(), order.clear();
  }

  void add_edge(int u, int v) {
    g[u].pb(v);
    rg[v].pb(u);
  }

  void OR(int u, int v) {
    add_edge(neg(u), v);
    add_edge(neg(v), u);
  }

  void XOR(int u, int v) {
    OR(u, v);
    OR(neg(u), neg(v));
  }

  void ForceTrue(int u) {
    add_edge(neg(u), u);
  }

  void ForceFalse(int u) {
    add_edge(u, neg(u));
  }

  void imply(int u, int v) {
    OR(neg(u), v);
  }

  int neg(int u) {
    if (u <= n) return u + n;
    return u - n;
  }

  void dfs(int u, vii g[], bool topsort) {
    vis[u] = true;
    for (int v : g[u]) {
      if (!vis[v]) dfs(v, g, topsort);
    }
    if (topsort) st.pb(u);
    else scc[u] = id, order.pb(u);
  }

  void build_scc() {
    for (int i = 1; i <= 2 * n; i++) {
      if (!vis[i]) dfs(i, g, true);
    }
    reverse(st.begin(), st.end());
    fill(vis, vis + 2 * n + 1, false);
    for (int u : st) {
      if (!vis[u]) id++, dfs(u, rg, false);
    }
  }

  bool solve() {
    build_scc();
    for (int i = 1; i <= n; i++) {
      if (scc[i] == scc[i + n]) return false
          ;
    }
    for (int i = (int)order.size() - 1; i >=
         0; i--) {
      int u = order[i];
      if (state[neg(u)] == false) state[u] =
           true;
    }
    return true;
  }
} solver;
```

8.2 Array Compression

```
void compress(int n) {
  vector<int> v;
  for (int i = 0; i < n; i++) v.pb(arr[i]);
  sort(v.begin(), v.end());
  v.erase(unique(v.begin(), v.end()), v.end
     ());
  for (int i = 0; i < n; i++) {
    int zipped = lower_bound(v.begin(), v.
       end(), arr[i]) - v.begin() + 1;
    arr[i] = zipped;
  }
}
```

8.3 Bitset With Range Operations

```
struct Bitset {
    const static int B = 6, K = 64, X = 63;
    ///returns mask with bits l to r set,
        and others reset
    static inline ULL getmask(int l, int r)
        {
        if (r==X) return -(1ULL<<l);
        return (1ULL<<(r+1)) - (1ULL<<l);
    }
    vector<ULL> bs;
    int N;
    Bitset(int n) {
        N = n/K+1;
        bs.resize(N);
    }
    void assign(ULL x) {
        fill(bs.begin()+1, bs.end(), 0);
        bs[0] = x;
    }
    bool get(int i) {
        return bs[i>>B] & (1ULL<<(i&X));
    }
    void set(int i) {
        bs[i>>B] |= (1ULL<<(i&X));
    }
    void reset(int i) {
        bs[i>>B] &= ~(1ULL<<(i&X));
    }
    void flip(int i) {
        bs[i>>B] ^= (1ULL<<(i&X));
    }
    void set(int l, int r) {
        int idl = l>>B, idr = r>>B;
        int posl = l&X, posr = r&X;
        if (idl == idr) {
            bs[idl] |= getmask(posl, posr);
            return;
        }
        bs[idl] |= getmask(posl, X);
        bs[idr] |= getmask(0, posr);
        for (int id = idl+1; id < idr; id++)
            bs[id] = -1;
    }
    void reset(int l, int r) {
        int idl = l>>B, idr = r>>B;
        int posl = l&X, posr = r&X;
        if (idl == idr) {
            bs[idl] &= ~getmask(posl, posr);
            return;
        }
        bs[idl] &= ~getmask(posl, X);
        bs[idr] &= ~getmask(0, posr);
        for (int id = idl+1; id < idr; id++)
            bs[id] = 0;
    }
    void flip(int l, int r) {
        int idl = l>>B, idr = r>>B;
        int posl = l&X, posr = r&X;
        if (idl == idr) {
            bs[idl] ^= getmask(posl, posr);
            return;
        }
        bs[idl] ^= getmask(posl, X);
        bs[idr] ^= getmask(0, posr);
        for (int id = idl+1; id < idr; id++)
            bs[id] = ~bs[id];
    }
};
```

8.4 Different Cumulative Sum

```
///cost = sum of (i * a[i]) where i starts
    from the beginning for every range
///sum[] = prefix sum of value[i]
///isum[] = prefix sum of i*value[i]
ll cost(int i, int j) {
  ll ret = isum[j];
  if (i) ret -= isum[i - 1];
  ll baad = sum[j];
  if (i) baad -= sum[i - 1];
  return ret - i * baad;
}
```

8.5 Factoradic Permutation Trick

```
vector<int> to_factoradic(ll n, int sz = 0)
    {
  vector<int> ret;
  int base = 1;
  while (n) {
    ret.pb(n % base);
    n /= base; base++;
  }
  while ((int)ret.size() < sz) ret.pb(0);
  reverse(ret.begin(), ret.end());
  return ret;
}
ll to_decimal(vector<int> &v) {
  int n = (int)v.size();
  ll ret = 0, base = n;
  for (int i = 0; i < n; i++, base--) {
```

```
    ret = ( ( (ret * base) % MOD ) + v[i] )
        % MOD;
  }
  return ret;
}
// returns permutation of size n from given
    factoradic number
vector<int> to_permutation(vector<int> &v) {
  vector<int> ret;
  ordered_set<int> st;
  int n = (int)v.size();
  for (int i = 0; i < n; i++) st.insert(i);
  for (int x : v) {
    int val = *st.find_by_order(x);
    st.erase(val); ret.pb(val);
  }
  return ret;
}
// returns lexicographical index of
    permutation in factoradic system
vector<int> order_of_permutation(vector<int>
     &p) {
  vector<int> ret; ordered_set<int> st;
  int n = (int)p.size();
  for (int i = 0; i < n; i++) st.insert(i);
  for (int x : p) {
    int idx = st.order_of_key(x);
    st.erase(x); ret.pb(idx);
  }
  return ret;
}
// returns sum of indices a and b in
    factoradic system
vector<int> add_order(vector<int> &a, vector
    <int> &b) {
  int n = (int)a.size();
  vector<int> ret(n); int carry = 0;
  for (int i = n - 1, base = 1; i >= 0; i--,
       base++) {
    ret[i] = a[i] + b[i] + carry;
    carry = ret[i] / (base); ret[i] %= base;
  }
  return ret;
}
// returns kth lexicographically smallest
    permutation of size n
// 0th permutation is 0 1 2 ... n-1
vector<int> kth_permutation(int k, int n) {
  // need to handle k >= n! if necessary
  vector<int> k_factoradic = to_factoradic(k
       , n);
  return to_permutation(k_factoradic);
}
```

8.6 Fractional Binary Search

```
/**
Given a function f and n, finds the smallest
     fraction p / q in [0, 1] or [0,n]
such that f(p / q) is true, and p, q <= n.
Time: O(log(n))
**/
struct frac { long long p, q; };
bool f(frac x) {
 return 6 + 8 * x.p >= 17 * x.q + 12;
}
frac fracBS(long long n) {
  bool dir = 1, A = 1, B = 1;
  frac lo{0, 1}, hi{1, 0}; // Set hi to 1/0
      to search within [0, n] and {1, 1} to
       search within [0, 1]
  if (f(lo)) return lo;
  assert(f(hi)); //checking if any solution
      exists or not
  while (A || B) {
    long long adv = 0, step = 1; // move hi
        if dir, else lo
    for (int si = 0; step; (step *= 2) >>=
        si) {
      adv += step;
      frac mid{lo.p * adv + hi.p, lo.q * adv
          + hi.q};
      if (abs(mid.p) > n || mid.q > n || dir
          == !f(mid)) {
        adv -= step; si = 2;
      }
    }
    hi.p += lo.p * adv;
    hi.q += lo.q * adv;
    dir = !dir;
    swap(lo, hi);
    A = B; B = !!adv;
  }
  return dir ? hi : lo;
}
```

8.7 Time Count

```
clock_t start,finish;
double timespent;
start=clock();
//Here Comes your Code/Function
finish=clock();
timespent=(double)(finish-start)/
     CLOCKS_PER_SEC;
cout << timespent << '\n';
```

9 Equations and Formulas

9.1 Catalan Numbers

$C_n = \frac{1}{n+1}\binom{2n}{n}$ $C_0 = 1, C_1 = 1$ and $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$

The number of ways to completely parenthesize $n+1$ factors.
The number of triangulations of a convex polygon with $n+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
The number of ways to connect the $2n$ points on a circle to form n disjoint i.e. non-intersecting chords.
The number of rooted full binary trees with $n+1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
Number of permutations of $1, \ldots, n$ that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing sub-sequence. For $n = 3$, these permutations are 132, 213, 231, 312 and 321.

9.2 Stirling Numbers First Kind

The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
$S(n, k)$ counts the number of permutations of n elements with k disjoint cycles.
$S(n, k) = (n-1) \cdot S(n-1, k) + S(n-1, k-1),\ where,\ S(0, 0) = 1, S(n, 0) = S(0, n) = 0 \sum_{k=0}^{n} S(n, k) = n!$
The unsigned Stirling numbers may also be defined algebraically, as the coefficient of the rising factorial:

$$x^{\bar{n}} = x(x+1)...(x+n-1) = \sum_{k=0}^{n} S(n, k)x^k$$

Lets $[n, k]$ be the stirling number of the first kind, then

$$\left[{n \atop n-k}\right] = \sum_{0 \le i_1 < i_2 < i_k < n} i_1 i_2 .... i_k.$$

9.3 Stirling Numbers Second Kind

Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets.
$S(n, k) = k \cdot S(n-1, k) + S(n-1, k-1),\ where\ S(0, 0) = 1, S(n, 0) = S(0, n) = 0$ $S(n, 2) = 2^{n-1} - 1$ $S(n, k) \cdot k! =$ number of ways to color n nodes using colors from 1 to k such that each color is used at least once.
An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements. It is denoted by $S_r(n, k)$ and obeys the recurrence relation. $S_r(n+1, k) = kS_r(n, k) + \binom{n}{r-1} S_r(n-r+1, k-1)$
Denote the n objects to partition by the integers $1, 2, \ldots, n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n, k)$, to be the number of ways to partition the integers $1, 2, \ldots, n$ into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i - j| \ge d$. It has been shown that these numbers satisfy, $S^d(n, k) = S(n-d+1, k-d+1), n \ge k \ge d$

9.4 Other Combinatorial Identities

$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$

$\sum_{i=0}^{k} \binom{n+i}{i} = \sum_{i=0}^{k} \binom{n+i}{n} = \binom{n+k+1}{k}$

$n, r \in N, n > r, \sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$

If $P(n) = \sum_{k=0}^{n} \binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} \cdot P(k)$$

If $P(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \cdot P(k)$$

9.5 Different Math Formulas

$$(1-x)(1-x^2)(1-x^3)\cdots = 1 - x - x^2 + x^5 + x^7 - x^{12} - x^{15} + x^{22} + x^{26} - \cdots .$$

The exponents $1, 2, 5, 7, 12, \cdots$ on the right hand side are given by the formula $g_k = \frac{k(3k-1)}{2}$ for $k = 1, -1, 2, -2, 3, \cdots$ and are called (generalized) pentagonal numbers. It is useful to find the partition number in $O(n\sqrt{n})$

Let a and b be coprime positive integers, and find integers $a\prime$ and $b\prime$ such that $aa\prime \equiv 1 \mod b$ and $bb\prime \equiv 1 \mod a$. Then the number of representations of a positive integers (n) as a non negative linear combination of a and b is

$$\frac{n}{ab} - \left\{\frac{b\prime n}{a}\right\} - \left\{\frac{a\prime n}{b}\right\} + 1$$

9.6 GCD and LCM

if m is any integer, then $\gcd(a + m \cdot b, b) = \gcd(a, b)$
The gcd is a multiplicative function in the following sense: if a_1 and a_2 are relatively prime, then $\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$.
$\gcd(a, \mathrm{lcm}(b, c)) = \mathrm{lcm}(\gcd(a, b), \gcd(a, c))$.
$\mathrm{lcm}(a, \gcd(b, c)) = \gcd(\mathrm{lcm}(a, b), \mathrm{lcm}(a, c))$.
For non-negative integers a and b, where a and b are not both zero, $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$

$\gcd(a, b) = \sum_{k|a \text{ and } k|b} \phi(k)$

$\sum_{i=1}^{n} [\gcd(i, n) = k] = \phi\left(\frac{n}{k}\right)$

$\sum_{k=1}^{n} \gcd(k, n) = \sum_{d|n} d \cdot \phi\left(\frac{n}{d}\right)$

$\sum_{k=1}^{n} x^{\gcd(k,n)} = \sum_{d|n} x^d \cdot \phi\left(\frac{n}{d}\right)$

$\sum_{k=1}^{n} \frac{1}{\gcd(k, n)} = \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{1}{n} \sum_{d|n} d \cdot \phi(d)$

$\sum_{k=1}^{n} \frac{k}{\gcd(k, n)} = \frac{n}{2} \cdot \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{n}{2} \cdot \frac{1}{n} \cdot \sum_{d|n} d \cdot \phi(d)$

$\sum_{k=1}^{n} \frac{n}{\gcd(k, n)} = 2 * \sum_{k=1}^{n} \frac{k}{\gcd(k, n)} - 1$, for $n > 1$

$\sum_{i=1}^{n} \sum_{j=1}^{n} [\gcd(i, j) = 1] = \sum_{d=1}^{n} \mu(d) \lfloor \frac{n}{d} \rfloor^2$

$\sum_{i=1}^{n} \sum_{j=1}^{n} \gcd(i, j) = \sum_{d=1}^{n} \phi(d) \lfloor \frac{n}{d} \rfloor^2$

$\sum_{i=1}^{n} \sum_{j=1}^{n} i \cdot j [\gcd(i, j) = 1] = \sum_{i=1}^{n} \phi(i) i^2$

$F(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathrm{lcm}(i, j) = \sum_{l=1}^{n} \left(\frac{\left(1 + \lfloor \frac{n}{l} \rfloor\right) \left(\lfloor \frac{n}{l} \rfloor\right)}{2} \right)^2 \sum_{d|l} \mu(d) l d$