Team notebook

September 6, 2016

Contents

1 Combinatorics - Binomial Coefficient with DP

```
//Binomial Coefficient
//C(N, K) = N!/(K!(N - K)!)
//Dynamic Programming
int bin[N][K];

bin[0][0] = 1;

for (int n = 1; n < MAXN; n++) {
    bin[n][0] = 1;
    bin[n][n] = 1;

    for (int k = 1; k < n; k++) {
            bin[n][k] = bin[n - 1][k] + bin[n - 1][k - 1];
            if (bin[n][k] >= MOD) {
                    bin[n][k] -= MOD;
            }
    }
 }
```

2 Combinatorics - Binomial Coefficient

```
Int nCr(Int n, Int k) {
        Int res = 1;

        if (k > (n >> 1LL)) {
                k = n-k;
        }
        for (Int i = 1; i <= k; i++, n--) {
                res = (res * n) / i;
        }

        return res;
}
```

3 Combinatorics - nCr

```
long long pow(int a, int b, int MOD) {
    long long x=1,y=a;

    while(b > 0) {
        if(b%2 == 1) {
            x=(x*y);
            if(x>MOD) x%=MOD;
        }
        y = (y*y);
        if(y>MOD) y%=MOD;
        b /= 2;
    }
    return x;
}

long long InverseEuler(int n, int MOD) {
    return pow(n,MOD-2,MOD);
}

long long C(int n, int r, int MOD) {
    vector<long long> f(n + 1,1);
    for (int i=2; i<=n;i++) {
        f[i] = (f[i-1]*i) % MOD;
    }
    return (f[n]*((InverseEuler(f[r], MOD) * InverseEuler(f[n-r], MOD)) %
        MOD)) % MOD;
}
```

4 DP - Coin Change

```
//Coin Change
int dp[1001];
```

```
int coins[] = {1, 5, 10, 25, 50};

dp[0] = 0;

for(int i = 1; i <= N; i++) {
       int min = 1000001;
    for(int j = 0; j < M; j++) {
               if(coins[j] <= i) {
                      int m = dp[i - coins[j]] + 1;
           if(m < min) min = m;
       }
    }
    dp[i] = min;
}
```

5 DP - Convex Hull Trick

```
int pointer;
vector<long long> M;
vector<long long> B;

bool bad(int l1,int l2,int l3) {
    return (B[l3] - B[l1]) * (M[l1] - M[l2]) < (B[l2] - B[l1]) * (M[l1] -
        M[l3]);
}

void add(long long m,long long b) {
    M.push_back(m);
    B.push_back(b);

    while (M.size() >= 3 && bad(M.size() - 3, M.size() - 2, M.size() -
        1)) {
       M.erase(M.end() - 2);
       B.erase(B.end() - 2);
    }
}
//Returns the minimum y-coordinate of any intersection between a given
    vertical
//line and the lower envelope
long long query(long long x) {
    if (pointer >= M.size()) {
       pointer = M.size() - 1;
    }

    while (pointer<M.size() - 1 &&
          M[pointer+1] * x + B[pointer+1] < M[pointer] * x + B[pointer]) {
       pointer++;
    }
    return M[pointer] * x + B[pointer];
}

add(rect[0].second,0);
pointer=0;

for (int i = 0; i < N; i++) {
    cost = query(rect[i].first);
    if (i < N) {
       add(rect[i+1].second, cost);
    }
 }
```

6 DP - Count used Digits

```
Int func(int val) {
       int digitCount = (int) log10(val) + 1;
       Int ans = 0LL;
       Int p = 1LL;

       for (int i = 0; i < digitCount - 1; i++) {
              ans += p * 9 * (i + 1);
              p *= 10;
       }

       ans += (val - p + 1) * digitCount;

       return ans;
}
```

7 DP - Divide And Conqueror Optmization

```
void cost(int k, int l, int r, int optL, int optR) {
```

```
    if (l > r) return;

    int m = (l + r) / 2;

    Int best = INF;
    int id = optL;

    for (int i = optL; i <= min(m, optR); i++) {
        Int now = dp[i][k - 1] + cost(i + 1, m);

        if (now < best) {
            best = now;
            id = i;
        }
    }

    dp[m][k] = best;

    cost(k, l, m - 1, optL, id);
    cost(k, m + 1, r, id, optR);
}

for (int i = 1; i <= N; i++) {
    dp[i][0] = cost(1, i);
 }

for (int i = 1; i <= M; i++) {
    cost(i, 1, N, 1, N);
 }
```

8 DP - Divide-Conquerer Optimization

```
void cost(int k, int l, int r, int optL, int optR) {
    if (l > r) return;

    int m = (l + r) / 2;

    Int best = INF;
    int id = optL;

    for (int i = optL; i <= min(m, optR); i++) {
        Int now = dp[i][k - 1] + cost(i + 1, m);

        if (now < best) {
            best = now;
            id = i;
        }
    }

    dp[m][k] = best;

    cost(k, l, m - 1, optL, id);
    cost(k, m + 1, r, id, optR);
}

for (int i = 1; i <= N; i++) {
    dp[i][0] = cost(1, i);
 }

for (int i = 1; i <= M; i++) {
    cost(i, 1, N, 1, N);
 }
```

9 DP - Dynamic Convex Hull Trick

```
struct Line {
    Int m, b;
    mutable function<const Line*()> succ;

    bool operator<(const Line& rhs) const {
        if (rhs.b != INF) {
            //invert operator to get minimum
            return m < rhs.m;
        }
        const Line* s = succ();
        if (!s) {
            return 0;
        }
        Int x = rhs.m;
        //invert operator to get minimum
        return b - s->b < (s->m - m) * x;
    }
};
```

```
struct HullDynamic : public multiset<Line> { // will maintain upper hull
    for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) {
            return y->m == x->m && y->b <= x->b;
        }
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    void insert_line(Int m, Int b) {
        auto y = insert({ m, b });

        y->succ = [=] {
            return next(y) == end() ? 0 : &*next(y);
        };
        if (bad(y)) {
            erase(y);
            return;
        }

        while (next(y) != end() && bad(next(y))) {
            erase(next(y));
        }
        while (y != begin() && bad(prev(y))) {
            erase(prev(y));
        }
    }
    Int eval(Int x) {
        auto l = *lower_bound((Line) {x, INF});
        return l.m * x + l.b;
    }
};

HullDynamic trick;
trick.insert_line(def[0].second, 0);

Int ans = 0;

for (int i = 0; i < N; i++) {
    ans = trick.eval(def[i].first);
    trick.insert_line(def[i + 1].second, ans);
 }
```

10 DP - Fractionak Knapsack

```
int N, B;
pair<int, int> P[100005];

bool cmp(pair<int, int> a, pair<int, int> b) {
    double valA = a.second == 0 ? INF : a.first / (double) a.second;
    double valB = b.second == 0 ? INF : b.first / (double) b.second;

    return valA < valB;
}

//value
for (int i = 0; i < N; i++) {
    cin >> P[i].first;
 }
//price
for (int i = 0; i < N; i++) {
    cin >> P[i].second;
 }

sort(P, P + N, cmp);

int ans = 0;

for (int i = N - 1; i >= 0; i--) {
    if (P[i].second <= B) {
        ans += P[i].first;
        B -= P[i].second;
    } else {
        ans += floor((B * P[i].first) / (double) P[i].second);
        B = 0;
    }
 }
```

11 DP - Kadane 2D

```
//Kadane 2D
for (int i = 1; i <= N; i++) {
       for (int j = 1; j <= N; j++) {
              cin >> M[i][j];
       }
       for (int j = 1; j <= N; j++) {
              dp[i][j] = dp[i][j - 1] + M[i][j];
       }
 }

int ans = -INT_MAX / 3;
for (int i = 1; i <= N; i++) {
       for (int j = i; j <= N; j++) {
              int sum = 0;
              for (int k = 1; k <= N; k++) {
                     sum += dp[k][j] - dp[k][i - 1];
                     chmax(ans, sum);
                     if (sum < 0) sum = 0;
              }
       }
 }
```

12 DP - Linha de Paretto - (LIS 2D)

```
#include <stdio.h>
#include <set>
#include <vector>
#include <algorithm>
using namespace std;
#define MAX 100010
#define inf 2000000000
struct no{
    int x,y;
};

no v[MAX];
int n;
set <pair<int,int> > S[MAX];
int topo;
set <pair<int, int> > :: iterator it, it2, ini, fim;
vector <pair<int, int> > aux;

bool cobre (pair <int, int> p, int s){
       it2 = S[s].lower_bound (make_pair (p.first-1, inf));
       if (it2 == S[s].begin()) return false;
       it2--;
       if (p.second > (*it2).second) return true;
       return false;
}

int main (){
    pair <int, int> p;
       topo = 0;
       scanf("%d", &n);
       for (int i = 0; i < n; i++){
           scanf("%d %d", &v[i].x, &v[i].y);
       }
       for (int i = 0; i < n; i++) S[i].clear();
       int ans = 0;

       p = make_pair (v[0].x, v[0].y);
       S[topo++].insert (p);

       for (int i = 1; i < n; i++){
              /*cria o pair do ponto i*/
           p = make_pair (v[i].x, v[i].y);
           /*busca*/
           /*verifica se ele cobre a ultima linha de parreto*/
           if (cobre(p, topo-1)){
               S[topo++].insert (p);
               continue;
           }
           /*faz busca binaria pra descobrir menor linha q ele nao cubra
               ninguem*/
           int u = 0, v = topo-1;
           while (u < v-1){
               int mid = (u+v)/2;
               if (cobre(p, mid)) u = mid;
               else v = mid;
           }
           int quem;
           if (cobre (p, u)) quem = v;
           else  quem = u;
           /*insercao*/
           /*insere na linha de parreto, removendo quem for necessario*/
           aux.clear();
           ini = S[quem].lower_bound (make_pair (p.first-1, inf));
```

```
            if (ini != S[quem].begin()){
                ini--;
                if ((*ini).second <= p.second) continue;
                ini++;
            }
            for (it = ini; it != S[quem].end() && (*it).second > p.second;
                 it++){
                aux.push_back(*it);
            }
            for (int j = 0; j < aux.size(); j++){
                S[quem].erase(S[quem].find(aux[j]));
            }
            ans++;
            S[quem].insert (p);
         }
        printf("%d\n", topo);
    return 0;
}
```

13 DP - Longest Common Subsequence - Efficient

```
//Longest Commom Subsequence - (LCS) O(n^2) - O(n) in space
int m[2][1000]; // instead of [1000][1000]
for (i = M; i >= 0; i--) {
       int ii = i&1;
       for (int j = N; j >= 0; j--) {
              if (i == M || j == N) {
                     m[ii][j]=0; continue;
              }
              if (s1[i] == s2[j]) {
                     m[ii][j] = 1 + m[1-ii][j+1];
              } else {
                     m[ii][j] = max(m[ii][j+1], m[1-ii][j]);
              }
       }
 }
cout<<m[0][0];
```

14 DP - Longest Common Subsequence

```
//Longest Commom Subsequence - (LCS) O(N^2)
int lcs(string a, string b) {
       int n = a.size(), m = b.size();
       int[][] dp = new dp[n+1][m+1];

       for(int i = 0; i <= max(n, m); i++) {
              dp[i][0] = dp[0][i] = 0;
       }
       for(int i = 1; i <= n; i++) {
              for(int j = 1; j <= m; j++) {
                     if(a[i] == b[j]) {
                            dp[i][j] = dp[i - 1][j - 1] + 1;
                     } else {
                            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
                     }
              }
       }
       return dp[n][m];
}
```

15 DP - Longest Increasing Subsequence O(n log) - Fenwick

```
int get(int pos) {
    int ans = 0;

    while (pos > 0) {
        ans = max(ans, tree[pos]);
        pos -= pos & -pos;
    }

    return ans;
}

void update(int pos, int new_value) {
    while (pos < MAXN) {
        tree[pos] = max(tree[pos], new_value);
        pos += pos & -pos;
    }
```

```
}

int ans = 1;

for (int i = 0; i < N; i++) {
    int now = get(P[i] - 1);
    update(P[i].second, now + 1);
    ans = max(ans, now + 1);
 }
```

16 DP - Longest Increasing Subsequence O(n log)) - Retrieve LIS

```
#define STRICTLY_INCREASNG
vector<int> LongestIncreasingSubsequence(vector<int> v) {
    vector<pair<int, int> > best;
    vector<int> dad(v.size(), -1);
    for (int i = 0; i < v.size(); i++) {
#ifdef STRICTLY_INCREASNG
        vector<pair<int, int> > item = make_pair(v[i], 0);
        vector<pair<int, int> > ::iterator it = lower_bound(best.begin(),
            best.end(), item);
        item.second = i;
#else
        vector<pair<int, int> > item = make_pair(v[i], i);
        vector<pair<int, int> > ::iterator it = upper_bound(best.begin(),
            best.end(), item);
#endif
        if (it == best.end()) {
            dad[i] = (best.size() == 0 ? -1 : best.back().second);
            best.push_back(item);
        } else {
            dad[i] = dad[it->second];
            *it = item;
        }
    }
    vector<int> ret;
    for (int i = best.back().second; i >= 0; i = dad[i])
        ret.push_back(v[i]);
    reverse(ret.begin(), ret.end());
    return ret;
}
```

17 DP - Longest Increasing Subsequence O(n^2)

```
int lis(int array[], int n) {
    int best[n], prev[n];

    for(int i = 0; i < n; i++) {
        best[i] = 1;
        prev[i] = i;
    }

    for(int i = 1; i < n; i++) {
        for(int j = 0; j < i; j++) {
            if(array[i] > array[j] && best[i] < best[j] + 1) {
                best[i] = best[j] + 1; prev[i] = j;
            }
        }
    }
    int ans = 0; for(int i = 0; i < n; i++) ans = max(ans, best[i]);
    return ans;
}
```

18 DP - N-th Palindrome Number

```
//Return the N-th palindromic number

std::string number_palindrome(int N) {
    if(N < 10){
        return std::string(1, char('0' + N));
    }
    long long sum = 0, digits = 1, v;
    for(; ; digits++){
        if(digits % 2 == 0){
            v = std::pow(10, digits/2-1) * 9;
        }else{
            v = std::pow(10, (digits+1)/2-1) * 9;
        }
        if(v + sum >= N) break;
        sum += v;
```

```
    }
    //I have to find the M-th palindromic number with X digits:
    long long Mth = N-sum;
    long long sz = (digits+1) / 2;
    long long pattern = std::pow(10, sz-1);
    pattern += (Mth-1);
    std::vector<int> tmp;
    while(pattern > 0){
        tmp.insert(tmp.begin(), pattern % 10);
        pattern /= 10;
    }
    int idx = digits-tmp.size() - 1;
    std::string ans = "";
    for(int i = 0; i < tmp.size(); i++){
        ans += std::string(1, char('0' + tmp[i]));
    }
    for(;idx >= 0;){
        ans += std::string(1, char('0' + tmp[idx--]));
    }
    return ans;
}
```

19 DP - Palindromic Check with DP

```
//Checa por Palindromos
int T, N, dp[MAXN][MAXN];
char str[MAXN];

for (int i = 0; i < N; i++) {
    dp[i][i] = 1;
    if(i + 1 < N) dp[i][i + 1] = str[i] == str[i + 1];
}
for(int k = 2; k < N; k++) {
    for (int i = 0; i < N - k; i++) {
        dp[i][i + k] = dp[i + 1][i + k - 1] && str[i] == str[i + k];
    }
}
```

20 DP - String Edit Distance

```
int dist(string& s1, string& s2) {
    int N1 = s1.size(), N2 = s2.size();

    for (int i = 0; i <= N1; i++) dp[i][0] = i;
    for (int i = 0; i <= N2; i++) dp[0][i] = i;

    for (int i = 1; i <= N1; i++) {
            for (int j = 1; j <= N2; j++) {
                    if(s1[i-1] == s2[j-1]) {
                            dp[i][j] = dp[i-1][j-1];
                    } else {
                            dp[i][j] = 1 + min(min(dp[i-1][j],
                                dp[i][j-1]), dp[i-1][j-1]);
                    }
            }
        }
        return dp[N1][N2];
}
```

21 DP - Subset Sum

```
//Subset-Sum -> (G = O valor total sendo testado, N = numero de valores
    disponiveis no array 'values'
int values[n];
bool subsetSum(int n, int g) {
    for(j = 0; j <= g; j++) sub[j] = 0;
    sub[0] = 1;
    for(j = 0; j < n; j++) if(values[j] != g) {
        for(int k = g; k >= values[j]; k--) {
            sub[k] |= sub[k - values[j]];
        }
    }
    return sub[g];
}
```

22 Data Structure - Mo Algorithm - Values in Edges

```
//For the euler tour tree
int in[MAXN], out[MAXN], ID[MAXN*2], dfsCnt;
int lca[MAXN][LOGN], level[MAXN];
vector<vector<pair<int, int> > > graph(MAXN);//<next, value>
int incomingEdge[MAXN];

//Dfs to mount LCA table and do a in-order visit
//storing first and last time to visit a node
void eulerTour(int node, int parent, int cost){
    in[node] = ++dfsCnt;
    ID[dfsCnt] = node;
    incomingEdge[node] = cost;
    for(int i = 0; i < (int) graph[node].size(); i++){
        pair<int, int> next = graph[node][i];
        if(next.first != parent){
            level[next.first] = level[node] + 1;
            eulerTour(next.first, node, next.second);
        }
    }
    out[node] = ++dfsCnt;
    ID[dfsCnt] = node;
}

struct query{
    int l, r, id, lca;

    query(){}
    query(int L, int R, int ID_, int LCA){
        l = L;
        r = R;
        id = ID_;
        lca = LCA;
    }

    bool operator < (const query &o) const{
        return r < o.r;
    }
};

//For Mo's algo:
int resp[MAXN];
int seen[MAXN];
int f[MAXN], ff[MAXN], big;

//add and remove in this case gets how many times
//appears the most frequent element in a range
void add(int color){
    ff[f[color]]--;
    f[color]++;
    ff[f[color]]++;
    if(f[color] > big){
        big = f[color];
    }
}
void remove(int color){
    ff[f[color]]--;
    f[color]--;
    ff[f[color]]++;
    if(ff[big] <= 0){
        big--;
    }
}

//If a node is visited 0 or two times, then,
//this node is not part of path A and B.
void fix(int node, int color){
    if(color != 0){
        if(seen[node] == 1){
            remove(color);
        }else{
            add(color);
        }
        seen[node] ^= 1;
    }
}

void processMo(int pos, vector<vector<query> > &blocks){
    sort(blocks[pos].begin(), blocks[pos].end());
    int l = blocks[pos][0].l-1, r = blocks[pos][0].l-1, ql, qr, id, lca_;
    big = 0;
    for(int i = 0; i < (int) blocks[pos].size(); i++){
        ql = blocks[pos][i].l;
        qr = blocks[pos][i].r;
        id = blocks[pos][i].id;
        lca_ = blocks[pos][i].lca;

        while(r < qr){
            fix(ID[r], incomingEdge[ID[r]]);
            r++;
        }
```

```
        while(l < ql){
            fix(ID[l], incomingEdge[ID[l]]);
            l++;
        }
        while(l > ql){
            l--;
            fix(ID[l], incomingEdge[ID[l]]);
        }

        /*
          The corner case, if we the problemas asks something on edges,
          as we stored the values of the edge on the children (comming
              down from the root),
          the LCA will add a wrong information about the path of nodes A
              and B.
          So we just remove it sepparated from the query, then asnwer the
              query, then add again LCA.
        */
        if (ID[l] == lca_ || ID[r] == lca_) fix(lca_, incomingEdge[lca_]);
        resp[id] = big;
        if (ID[l] == lca_ || ID[r] == lca_) fix(lca_, incomingEdge[lca_]);
    }
    while(l < r){
        fix(ID[l], incomingEdge[ID[l]]);
        l++;
    }
}

int QU[MAXN], QV[MAXN], Q; //attention on array size

//fill the blocks with queries
void fillBlocks(vector<vector<query> > &blocks, const int BLOCK_SIZE){
    for(int i = 0; i < Q; i++){
        int u = QU[i], v = QV[i];
        if(u == v){
            resp[i] = 0;
            continue;
        }
        if(in[u] > in[v]) swap(u,v);
        query q(-1, -1, i, getLca(u,v));
        if(q.lca == u || q.lca == v){
            q.l = in[u], q.r = in[v];
        }else{
            q.l = out[u], q.r = in[v];
        }
        q.r++;
        blocks[q.l / BLOCK_SIZE].push_back(q);
    }
}

int N;

//build the tree -> vector graph
for(int i = 0; i < Q; i++){
    cin >> QU[i] >> QV[i];
 }

dfsCnt = 0;
eulerTour(1, -1, 0);

int BLOCK_SIZE = sqrt(N*2) + 1;
vector<vector<query> > blocks(BLOCK_SIZE);

fillBlocks(blocks, BLOCK_SIZE);//mounting blocks

for(int i = 0; i < BLOCK_SIZE; i++){
    if(blocks[i].size()){
        processMo(i, blocks);
    }
 }
```

23 Data Structure - Mo Algorithm - Values in Vertex

```
//For LCA
vector<vector<int> > graph(MAXN);
int lca[MAXN][LOGN], level[MAXN];

//For Euler Tour
int ID[MAXN*2], in[MAXN], out[MAXN], dfsCnt;

void dfs(int node, int parent){
    lca[node][0] = parent == -1 ? node : parent;
    for(int i = 1; i < LOGN; i++){
        lca[node][i] = lca[lca[node][i-1]][i-1];
    }
    in[node] = ++dfsCnt;
```

```
    ID[dfsCnt] = node;
    for(int i = 0; i < (int) graph[node].size(); i++){
        int next = graph[node][i];
        if(next != parent){
            level[next] = level[node] + 1;
            dfs(next, node);
        }
    }
    out[node] = ++dfsCnt;
    ID[dfsCnt] = node;
}

//For Mo's algorithm
struct query{
    int l, r, id, lca;
    query(int l_, int r_, int id_, int lca_){
        l = l_, r = r_, id = id_, lca = lca_;
    }
    query(){}
    bool operator < (const query &o) const{
        return r < o.r;
    }
};

const int BLOCK_SIZE = sqrt(MAXN*2) + 5;
vector<vector<query> > blocks(BLOCK_SIZE);

int N, U, V, Q, value[MAXN], valueTmp[MAXN];
int QU[MAXQ], QV[MAXQ], resp[MAXQ];
int seen[MAXN], cnt[MAXN], ansCnt;

void fillBlocks(){
    for(int i = 0; i < Q; i++){
        U = QU[i], V = QV[i];
        int l, r, lca_ = getLca(U, V);
        if(in[U] > in[V]) swap(U,V);
        if(lca_ == U || lca_ == V){
            l = in[U], r = in[V];
        }else{
            l = out[U], r = in[V];
        }
        r++;
        blocks[l / BLOCK_SIZE].push_back(query(l,r,i,lca_));
    }
}
```

```
void fix(int node){
    int color = value[node];//The value of node

    //If the node appears two ou zero times already, we need to add
    if(seen[node] == 0){
        if(cnt[color] == 0){
            ansCnt++;
        }
        cnt[color]++;
    }else{//Remove
        cnt[color]--;
        if(cnt[color] == 0){
            ansCnt--;
        }
    }
    seen[node] ^= 1;
}

void process(int pos){
    sort(blocks[pos].begin(), blocks[pos].end());
    int l = blocks[pos][0].l-1, r = l, ql, qr, id, lca_;
    ansCnt = 0;
    for(int i = 0; i < blocks[pos].size(); i++){
        ql = blocks[pos][i].l;
        qr = blocks[pos][i].r;
        id = blocks[pos][i].id;
        lca_ = blocks[pos][i].lca;
        while(r < qr) fix(ID[r++]);
        while(l < ql) fix(ID[l++]);
        while(l > ql) fix(ID[--l]);

        //The tree stores the values on the vertex,
        //so the LCA always is not in the path A, B
        //We just add, answer, remove
        if(ID[l] != lca_ && ID[r] != lca_) fix(lca_);
        resp[id] = ansCnt;
        if(ID[l] != lca_ && ID[r] != lca_) fix(lca_);
    }
    ansCnt = 0;
    memset(seen, 0, sizeof(seen));
    memset(cnt, 0, sizeof(cnt));
}

//values[i] = values of the nodes
```

```
//be careful, maybe needed to compress this values
//build tree on variable graph
for(int i = 0; i < Q; i++){
    scanf("%d%d", &QU[i], &QV[i]);
 }
dfsCnt = 0;
dfs(1, -1);
fillBlocks();
for(int i = 0; i < BLOCK_SIZE; i++){
    if((int) blocks[i].size()){
        process(i);
    }
 }
```

24 Data Structure - Mo Algorithm

```
const int MAXN = 100005;
const int LOGMAXVAL = 20;

int n, q, val[MAXN];
int BLOCK_SIZE = sqrt(MAXN)+5;
int resp[MAXN];
int seen[MAXN];
int answer = 0;

struct pt{
    int l, r, id;

    pt(){}
    pt(int L, int R, int ID){
        l = L;
        r = R;
        id = ID;
    }

    bool operator < (const pt &o) const{
        return r < o.r;
    }
};

vector<vector<pt> > cons(BLOCK_SIZE);

void add(int pos){
    if(seen[val[pos]] == 1) {
        answer += 2;
    } else if (seen[val[pos]] > 1) {
        answer += 1;
    }
    seen[val[pos]]++;
}

void remove(int pos){
    if(seen[val[pos]] > 2){
        answer--;
    } else if (seen[val[pos]] == 2) {
        answer -= 2;
    }
    seen[val[pos]]--;
}

void process(int pos){
    int l = pos*BLOCK_SIZE, r = pos*BLOCK_SIZE, ql, qr, id;
    answer = 0;
    sort(cons[pos].begin(), cons[pos].end());
    for(int i = 0; i < (int) cons[pos].size(); i++){
        ql = cons[pos][i].l;
        qr = cons[pos][i].r;
        id = cons[pos][i].id;
        while(r < qr){
            add(r);
            r++;
        }
        while(l < ql){
            remove(l);
            l++;
        }
        while(l > ql){
            l--;
            add(l);
        }
        resp[id] = answer;
    }
    for(int j = l; j < r; j++){
        remove(j);
    }
}
```

```
for (int i = 0; i < q; i++) {
    pt newQ;
    cin >> newQ.l >> newQ.r;
    newQ.id = i;
    //This is problem dependent, there are cases when newQ.r++ is
        necessary
    newQ.l--;
    cons[newQ.l / BLOCK_SIZE].push_back(newQ);
 }

for(int i = 0; i < BLOCK_SIZE; i++){
    if(cons[i].size()){
        process(i);
    }
 }

for (int i = 0; i < q; i++){
    printf("%d\n", resp[i]);
 }
```

25 Data Structure - Persistent Segment Tree

```
//Persistent Segment Tree
int root[MAXQ]; //The root of the new node
int INDEX;
int Lef[MAXN*4*LOGMAXVAL];
int Rig[MAXN*4*LOGMAXVAL];
int S[MAXN*4*LOGMAXVAL];

struct PersistentSegTree{

    PersistentSegTree(){
              INDEX = 1;
        build(0, 0, MAXN);
    }

        //build the initial and empty tree
    void build(int node, int l, int r){
        if(l == r){
            return;
        }else{
            int mid = (l+r) / 2;
            Lef[node] = INDEX++;
            Rig[node] = INDEX++;
            build(Lef[node], l, mid);
            build(Rig[node], mid+1, r);
        }
    }

        /*query to count how many elements are > K
          here is the key of the problem.*/
    int query(int node, int l, int r, int K){
              if(r <= K){
                    return 0;
              }else if(l > K){
            return S[node];
        }else{
            int mid = (l+r) / 2;
            return query(Lef[node], l, mid, K) + query(Rig[node], mid+1,
                r, K);
        }
    }

        /*add a new node, we just need to copy log(n) nodes
          from the previus tree add add the new one*/
    int update(int node, int l, int r, int pos){
        int next = INDEX++;
        Lef[next] = Lef[node];
        Rig[next] = Rig[node];
        S[next] = S[node];
        if(l == r){
            S[next] += 1;
        }else{
            int mid = (l+r) / 2;
            if(pos <= mid){
                Lef[next] = update(Lef[node], l, mid, pos);
            }else{
                Rig[next] = update(Rig[node], mid+1, r, pos);
            }
            S[next] = S[Lef[next]] + S[Rig[next]];
        }
        return next;
    }
};
```

26 Data Structure - Segment Tree - Lazy Propagation

```
void goDown(int node, int l, int r) {
    if (lazy[node]) {
        tree[node] += lazy[node];

        if (l != r) {
            lazy[2 * node] += lazy[node];
            lazy[2 * node + 1] += lazy[node];
        }
    }
    lazy[node] = 0;
}

void build(int node, int l, int r) {
    if (l == r) {
        tree[node] = A[l];
    } else {
        int m = (l + r) / 2;

        build(2 * node, l, m);
        build(2 * node + 1, m + 1, r);

        tree[node] = max(tree[2 * node], tree[2 * node + 1]);
    }
}

Int query(int node, int l, int r, int bl, int br) {
    goDown(node, l, r);
    if (l >= bl && r <= br) {
        return tree[node];
    } else if (l > br || r < bl) {
        return -INF;
    } else {
        int m = (l + r) / 2;

        Int a = query(2 * node, l, m, bl, br);
        Int b = query(2 * node + 1, m + 1, r, bl, br);

        return max(a, b);
    }
}
```

```
void update(int node, int l, int r, int bl, int br, Int value) {
    goDown(node, l, r);
    if (l > r) {
        return;
    } else if (l > br || r < bl) {
        return;
    } else if (l >= bl && r <= br) {
        lazy[node] = value;
        goDown(node, l, r);
    } else {
        int m = (l + r) / 2;

        update(2 * node, l, m, bl, br, value);
        update(2 * node + 1, m + 1, r, bl, br, value);

        tree[node] = max(tree[2 * node], tree[2 * node + 1]);
    }
}
```

27 Data Structure - Segment Tree - MergeSort

```
vector<int> tree[MAXN << 2];

int lower_search(vector<int> &arr, int key){
    int lo = 0, hi = arr.size() - 1, ans = INF;
    while(lo <= hi){
        int mid = (lo + hi) >> 1;
        if(arr[mid] >= key){
            ans = min(ans, mid);
            hi = mid-1;
        }else{
            lo = mid+1;
        }
    }
    return ans;
}

vector<int> merge(vector<int> &l, vector<int> &r){
    vector<int> ans;
    int idxl = 0;
    int idxr = 0;
    while(idxl < l.size() && idxr < r.size()){
```

```
        if(l[idxl] < r[idxr]){
            ans.push_back(l[idxl++]);
        }else if(l[idxl] > r[idxr]){
            ans.push_back(r[idxr++]);
        }else{
            ans.push_back(l[idxl++]);
            ans.push_back(r[idxr++]);
        }
    }
    while(idxl < l.size()){
        ans.push_back(l[idxl++]);
    }
    while(idxr < r.size()){
        ans.push_back(r[idxr++]);
    }
    return ans;
}

void build(int node, int l, int r){
    if(l > r) return;
    if(l == r){
        tree[node] = vector<int>(1, go[l]);
    }else{
        int mid = (r+l) >> 1;
        build(node << 1, l, mid);
        build((node << 1) | 1, mid+1, r);
        tree[node] = merge(tree[node << 1], tree[(node << 1) | 1]);
    }
}

//couting how many elements are greater than K
int query(int node, int l, int r, int bl, int br){
    if(l > br || r < bl || l > r){
        return 0;
    }else if(l >= bl && r <= br){
        //int greater = upper_bound(tree[node].begin(), tree[node].end(),
            br) - tree[node].begin();
        int greater = lower_search(tree[node], br+1);
        if(greater == INT_MAX){
            return 0;
        }else{
            return tree[node].size() - greater;
        }
    }else{
        int mid = (l+r) >> 1;
        int lq = query(node << 1, l, mid, bl, br);
        int rq = query((node << 1) | 1, mid+1, r, bl, br);
        return lq + rq;
    }
}
```

28 Data Structure - Segment Tree 2D

```
// Segment Tree 2D


pair<int, int> tree[4 * MAXN][4 * MAXN];

void build_y(int nx, int ny, int xl, int xr, int yl, int yr) {
        if (yl == yr) {
                if (xl == xr) {
                        tree[nx][ny].first = tree[nx][ny].second =
                            P[xl][yl];
                } else {
                        tree[nx][ny].first = min(tree[2 * nx][ny].first,
                            tree[2 * nx + 1][ny].first);
                        tree[nx][ny].second = max(tree[2 * nx][ny].second,
                            tree[2 * nx + 1][ny].second);
                }
        } else {
                int m = (yl + yr) / 2;

                build_y(nx, 2 * ny, xl, xr, yl, m);
                build_y(nx, 2 * ny + 1, xl, xr, m + 1, yr);

                tree[nx][ny].first = min(tree[nx][2 * ny].first,
                    tree[nx][2 * ny + 1].first);
                tree[nx][ny].second = max(tree[nx][2 * ny].second,
                    tree[nx][2 * ny + 1].second);
        }
}

void build_x(int nx, int xl, int xr) {
        if (xl > xr) {
                return;
        } else if (xl != xr) {
                int m = (xl + xr) / 2;
```

```
            build_x(2 * nx, xl, m);
            build_x(2 * nx + 1, m + 1, xr);
        }
        build_y(nx, 1, xl, xr, 0, M - 1);
}

pair<int, int> query_y(int nx, int ny, int xl, int xr, int yl, int yr,
    int bound_lx, int bound_rx, int bound_ly, int bound_ry) {
        if (yl > yr || yl > bound_ry || yr < bound_ly) {
            return make_pair(INF, -INF);
        } else if (yl >= bound_ly && yr <= bound_ry) {
            return tree[nx][ny];
        } else {
            int m = (yl + yr) / 2;

            pair<int, int> q1 = query_y(nx, 2 * ny, xl, xr, yl, m,
                bound_lx, bound_rx, bound_ly, bound_ry);
            pair<int, int> q2 = query_y(nx, 2 * ny + 1, xl, xr, m + 1,
                yr, bound_lx, bound_rx, bound_ly, bound_ry);

            return make_pair(min(q1.first, q2.first), max(q1.second,
                q2.second));
        }
}

pair<int, int> query_x(int nx, int ny, int xl, int xr, int yl, int yr,
    int bound_lx, int bound_rx, int bound_ly, int bound_ry) {
        if (xl > xr || xl > bound_rx || xr < bound_lx) {
            return make_pair(INF, -INF);
        } else if (xl >= bound_lx && xr <= bound_rx) {
            return query_y(nx, 1, xl, xr, 0, M - 1, bound_lx,
                bound_rx, bound_ly, bound_ry);
        } else {
            int m = (xl + xr) / 2;

            pair<int, int> q1 = query_x(2 * nx, ny, xl, m, yl, yr,
                bound_lx, bound_rx, bound_ly, bound_ry);
            pair<int, int> q2 = query_x(2 * nx + 1, ny, m + 1, xr, yl,
                yr, bound_lx, bound_rx, bound_ly, bound_ry);

            return make_pair(min(q1.first, q2.first), max(q1.second,
                q2.second));
        }
}
```

```
pair<int, int> query(int nx, int ny, int xl, int xr, int yl, int yr, int
    bound_lx, int bound_rx, int bound_ly, int bound_ry) {
        return query_x(1, 1, xl, xr, yl, yr, bound_lx, bound_rx, bound_ly,
            bound_ry);
}

void update_y(int nx, int ny, int xl, int xr, int yl, int yr, int posx,
    int posy, int value) {
        if (yl == yr) {
            if (xl == xr) {
                tree[nx][ny].first = tree[nx][ny].second = value;
            } else {
                tree[nx][ny].first = min(tree[2 * nx][ny].first,
                    tree[2 * nx + 1][ny].first);
                tree[nx][ny].second = max(tree[2 * nx][ny].second,
                    tree[2 * nx + 1][ny].second);
            }
        } else {
            int m = (yl + yr) / 2;

            if (posy <= m) {
                update_y(nx, 2 * ny, xl, xr, yl, m, posx, posy,
                    value);
            } else {
                update_y(nx, 2 * ny + 1, xl, xr, m + 1, yr, posx,
                    posy, value);
            }

            tree[nx][ny].first = min(tree[nx][2 * ny].first,
                tree[nx][2 * ny + 1].first);
            tree[nx][ny].second = max(tree[nx][2 * ny].second,
                tree[nx][2 * ny + 1].second);
        }
}

void update_x(int nx, int ny, int xl, int xr, int yl, int yr, int posx,
    int posy, int value) {
        if (xl != xr) {
            int m = (xl + xr) / 2;

            if (posx <= m) {
                update_x(2 * nx, ny, xl, m, yl, yr, posx, posy,
                    value);
            } else {
```

```
                    update_x(2 * nx + 1, ny, m + 1, xr, yl, yr, posx,
                        posy, value);
                }
        }
        update_y(nx, 1, xl, xr, 0, M - 1, posx, posy, value);
}

void update(int nx, int ny, int xl, int xr, int yl, int yr, int posx, int
    posy, int value) {
        return update_x(1, 1, xl, xr, yl, yr, posx, posy, value);
}
```

29 Data Structure - Treap Implicit

```
struct Node {
    Node* L;
    Node* R;

    Int value;
    int priority;
    int size;

    Int sum;
    Int lazy;

    Node(Int v) {
        value = v;
        size = 1;
        sum = v;
        lazy = 0;

        priority = rand() % 1000000;
    }

    void update_size() {
        size = 1;

        if (L) {
            size += L->size;
        }
        if (R) {
            size += R->size;
        }
    }

    void updateLazy() {
        if (lazy) {
            value += lazy;
            sum += lazy * size;

            if (L) {
                L->lazy += lazy;
            }
            if (R) {
                R->lazy += lazy;
            }
        }
        lazy = 0;
    }

    void fix() {
        sum = value;

        if (L) {
            L->updateLazy();
            sum += L->sum;
        }
        if (R) {
            R->updateLazy();
            sum += R->sum;
        }

        update_size();
    }
};

void split(Node* root, Node*& l, Node*& r, int pos, int add=0) {
    if (!root) {
        l = NULL;
        r = NULL;
    } else {
        root->updateLazy();

        int curr_pos = add;

        if (root->L) {
            curr_pos += (root->L)->size;
```

```
        }

        if (curr_pos <= pos) {
            split(root->R, root->R, r, pos, curr_pos + 1);
            l = root;
        } else {
            split(root->L, l, root->L, pos, add);
            r = root;
        }
    }
    if (root) {
        root->update_size();
        root->fix();
    }
}

void merge(Node*& root, Node*& l, Node*& r) {
    if (l) {
        l->updateLazy();
    }
    if (r) {
        r->updateLazy();
    }
    if (l == NULL || r == NULL) {
        if (l != NULL) {
            root = l;
        } else {
            root = r;
        }
    } else {
        if (l->priority > r->priority) {
            merge(l->R, l->R, r);
            root = l;
        } else {
            merge(r->L, l, r->L);
            root = r;
        }
    }
    if (root) {
        root->update_size();
        root->fix();
    }
}
```

```
void insert(Node*& root, int pos, int value) {
    Node* inserted = new Node(value);

    if (root == NULL) {
        root = inserted;
    } else {

        Node* left;
        Node* right;
        Node* buff;


        split(root, left, right, pos - 1);
        merge(root, left, inserted);
        merge(buff, root, right);
        root = buff;
    }
}

Int range_query(Node*& root, int l, int r) {
    Node* left;
    Node* mid;
    Node* right;

    split(root, left, mid, l-1);
    split(mid, root, right, r-l);

    Int ans = root->sum;

    merge(mid, left, root);
    merge(root, mid, right);

    return ans;
}

void range_update(Node*& root, int l, int r, Int val){
    Node* left;
    Node* mid;
    Node* right;

    split(root, left, mid, l-1);
    split(mid, root, right, r-l);

    root->lazy+=val;
```

```
    merge(mid, left, root);
    merge(root, mid, right);
}
```

30 Data Structure - Treap

```
const int MAXN = 100005;

struct Node {
    Node* L;
    Node* R;

    int value;
    int priority;
    int size;

    Node(int v) {
        value = v;
        size = 1;
        priority = rand() % MAXN;
    }

    void update_size() {
        size = 1;

        if (L) {
            size += L->size;
        }
        if (R) {
            size += R->size;
        }
    }
};

void printP(Node* root) {
    if (root == NULL) {
        return;
    } else {
        printP(root->L);
        cout << root->value << " ";
        printP(root->R);
    }
}
void printI(Node* root) {
    if (root == NULL) {
        return;
    } else {
        cout << root->value << " ";
        printI(root->L);
        printI(root->R);
    }
}


void split(Node* root, Node*& l, Node*& r, int val) {
    if (!root) {
        l = NULL;
        r = NULL;
    } else {
        if (root->value <= val) {
            split(root->R, root->R, r, val);
            l = root;
        } else {
            split(root->L, l, root->L, val);
            r = root;
        }
    }
    if (root) {
        root->update_size();
    }
}

void merge(Node*& root, Node*& l, Node*& r) {
    if (l == NULL || r == NULL) {
        if (l != NULL) {
            root = l;
        } else {
            root = r;
        }
    } else {
        if (l->priority > r->priority) {
            merge(l->R, l->R, r);
            root = l;
        } else {
            merge(r->L, l, r->L);
            root = r;
        }
```

```
    }
    if (root) {
        root->update_size();
    }
}

void insert(Node*& root, Node*& inserted) {
    if (root == NULL) {
        root = inserted;
    } else {
        if (root->priority < inserted->priority) {
            split(root, inserted->L, inserted->R, inserted->value);
            root = inserted;
        } else {
            if (root->value <= inserted->value) {
                insert(root->R, inserted);
            } else {
                insert(root->L, inserted);
            }
        }
    }
    if (root) {
        root->update_size();
    }
}

void remove(Node*& root, int value) {
    if (root == NULL) {
        return;
    } else {
        if (root->value == value) {
            merge(root, root->L, root->R);
        } else {
            if (root->value < value) {
                remove(root->R, value);
            } else {
                remove(root->L, value);
            }
        }
    }
    if (root) {
        root->update_size();
    }
}
```

```
bool find(Node* root, int value) {
    if (root == NULL) {
        return false;
    } else if (root->value == value) {
        return true;
    } else {
        if (root->value <= value) {
            return find(root->R, value);
        } else {
            return find(root->L, value);
        }
    }
}

//What's the kth smallest number ?
Node* kth(Node* root, int pos) {
    if (!root) {
        return NULL;
    } else {
        int curr_pos = 1;

        if (root->L) {
            curr_pos += root->L->size;
        }

        if (curr_pos == pos) {
            return root;
        } else if (root->L && curr_pos > pos) {
            return kth(root->L, pos);
        } else if (root->R) {
            return kth(root->R, pos - 1 - (root->L ? root->L->size : 0));
        } else {
            return NULL;
        }
    }
}

//How many numbers are smaller than value ?
int query(Node* root, int value) {
    if (root == NULL) {
        return 0;
    } else {
        if (root->value < value) {
            int ans = 1;
```

```
            if (root->L != NULL) {
                ans += root->L->size;
            }

            return ans + query(root->R, value);
        } else {
            return query(root->L, value);
        }
    }
}
```

31 Data Structures - Fenwick Tree 2D

```
const int INF = 1000 * 1000 * 1000;

int n, m;
vector <vector <int>> t;

void init(int _n, int _m) {
    n = _n;
    m = _m;
    for(int i = 0; i < n; i++) {
        t.push_back(vector<int>(m, 0));
    }
}

int sum(int x, int y) {
    int result = 0;
    for (int i = x; i> = 0; i = (i & (i +1)) - 1) {
        for (int j = y; j> = 0; j = (j & (j +1)) - 1) {
            result += t[i][j];
        }
    }
    return result;
}

void inc (int x, int y, int delta) {
    for (int i = x; i <n; i = (i | (i +1))) {
        for (int j = y; j <m; j = (j | (j +1))) {
            t[i][j] += delta;
        }
    }
}

void update(int x, int y, int new_value) {
    for (int i = x; i> = 0; i = (i & (i +1)) - 1) {
        for (int j = y; j> = 0; j = (j & (j +1)) - 1) {
            t[i][j] = new_value;
        }
    }
}

// sum[(r1, c1), (r2, c2)]
int sum(int[][] t, int r1, int c1, int r2, int c2) {
        return sum(t, r2, c2) - sum(t, r1 - 1, c2) - sum(t, r2, c1 - 1) +
            sum(t, r1 - 1, c1 - 1);
}
```

32 Data Structures - Fenwick Tree

```
template<typename T = int>
struct FenwickTree {
    int N;
    T *values;

    FenwickTree(int N) {
        this->N = N;
        values = new T[N+5];

        for(int i = 1; i <= N; i++) values[i] = 0;
    }

    void increase(int index, T add) {
        while(index <= N) {
            values[index] += add;
            index += (index & -index);
        }
    }

    void update(int index, T new_value) {
        increase(index, new_value - readSingle(index));
    }

    T read(int index) {
```

```
        T sum = 0;

        while(index > 0) {
            sum += values[index];
            index -= (index & -index);
        }

        return sum;
    }

    T readSingle(int index){
        T sum = values[index];
        if(index > 0) {
            int z = index - (index & -index);
            index--;
            while(index != z) {
                sum -= values[index];
                index -= (index & -index);
            }
        }
        return sum;
    }

    T read(int low, int high) {
        return read(high) - read(low - 1);
    }

    void scale(T factor) {
        for(int i = 1; i <= N; i++) {
            values[i] /= factor;
        }
    }

    void power(T factor) {
        for(int i = 1; i <= N; i++) {
            values[i] *= factor;
        }
    }
};
```

33 Data Structures - Median Online Algorithm

```
//Get median of a sequence in O(log(n))
int median_retrieve(void) {
        if (minHeap.empty() && maxHeap.empty()) return 0;

        if (minHeap.size() == maxHeap.size()) {
                return min(minHeap.top(), maxHeap.top());
        } else {
                if (minHeap.size() > maxHeap.size()) {
                        return minHeap.top();
                } else {
                        return maxHeap.top();
                }
        }
}

void median_insert(int x) {
        if (x > median_retrieve()) {
                minHeap.push(x);
        } else {
                maxHeap.push(x);
        }

        while (abs((int) (minHeap.size() - maxHeap.size())) > 1) {
                if (minHeap.size() > maxHeap.size()) {
                        int tmp = minHeap.top();
                        minHeap.pop();
                        maxHeap.push(tmp);
                } else {
                        int tmp = maxHeap.top();
                        maxHeap.pop();
                        minHeap.push(tmp);
                }
        }
}
```

34 Data Structures - Sliding Window RMQ Faster

```
//Sliding RMQ in O(N) - Faster (No use of STL)
int Q[MAXN];

Int maxSlidingWindow(Int A[], int n, int w, Int B[]) {
```

```
        int b = 0, e = 0;
        Int ans = 0LL;
        for (int i = 0; i < w; i++) {
                while (!(b == e) && A[i] >= A[Q[e-1]]) {
                        e -= 1;
                }
                Q[e++] = i;
        }
        for (int i = w; i < n; i++) {
                B[i-w] = A[Q[b]];
                ans += B[i-w];
                while (!(e == b) && A[i] >= A[Q[e-1]])
                        e--;
                while (!(e == b) && Q[b] <= i-w)
                        b += 1;
                Q[e++] = i;
        }
        ans += A[Q[b]];

        return ans;
}
```

35 Data Structures - Sliding Window RMQ

```
void maxSlidingWindow(int A[], int n, int w, int B[]) {
  deque<int> Q;
  for (int i = 0; i < w; i++) {
    while (!Q.empty() && A[i] >= A[Q.back()])
      Q.pop_back();
    Q.push_back(i);
  }
  for (int i = w; i < n; i++) {
    B[i-w] = A[Q.front()];
    while (!Q.empty() && A[i] >= A[Q.back()])
      Q.pop_back();
    while (!Q.empty() && Q.front() <= i-w)
      Q.pop_front();
    Q.push_back(i);
  }
  B[n-w] = A[Q.front()];
}
```

36 Data Structures - Sparse Table O(1) Query

```
int n, val[MAXN], pre[MAXN];
int dp[MAXN][LOGVAL];
void preProcess(){
    int base = 1;
    int pot = 0;
    for(int i = 0; i < MAXN; i++){
        if(i >= base * 2){
            pot++;
            base *= 2;
        }
        pre[i] = pot;
        dp[i][0] = i;
    }
    base = 2;
    pot = 1;
    while(base <= n){
        for(int i = 0; i + base / 2 - 1 < n; i++){
            int before = base / 2;
            if(val[dp[i][pot-1]] < val[dp[i + before][pot-1]]){
                dp[i][pot] = dp[i][pot-1];
            }else{
                dp[i][pot] = dp[i + before][pot-1];
            }
        }
        base *= 2;
        pot++;
    }
}
int query(int l, int r){
    int len = r-l+1;
    if(len == 1){
        return dp[r][0];
    }else{
        int base = (1 << pre[len]);
        int pot = pre[len];
        if(val[dp[l][pot]] < val[dp[r-base+1][pot]]){
            return dp[l][pot];
        }else{
            return dp[r-base+1][pot];
        }
    }
}
```

```
//0-based, dentro da main:
preProcess();
val[query(left, right)] //->should be the answer
```

37 Data Structures - SparseTable

```
void build() {
    int pw = 1; //2^pw
    int base = 2;

    for (int i = 0; i < N; i++) {
        dp[i][0] = P[i];
    }

    while (base <= N) {
        for (int i = 0; i + base / 2 - 1 < N; i++) {
            int before = base / 2;
            dp[i][pw] = min(dp[i][pw - 1], dp[i + before][pw - 1]);
        }
        pw += 1;
        base *= 2;
    }
}

int query(int l, int r) {
    int len = r - l + 1;

    if (len == 1) return dp[l][0];

    int ps = 1;
    int pw = 0;

    while (l + 2 * ps <= r) {
        ps *= 2;
        pw += 1;
    }

    int a = dp[l][pw];
    int b = dp[r - ps + 1][pw];

    return min(a, b);
}
```

38 Data Structures - Trie

```
//Trie
struct Trie {
    Trie *child[MAXN];
    int prefixes;
    int words;

    Trie() {
        int i;
        prefixes = words = 0;
        for(i = 0; i < MAXN; i++) {
            child[i] = NULL;
        }
    }

    void addWord(string s, int pos = 0) {
        if(pos == s.size()) {
            words++;
            return;
        }

        int letter_pos = s[pos] - 'a';

        Trie *t = child[letter_pos];

        if(child[letter_pos] == NULL) {
            t = child[letter_pos] = new Trie();
            t->prefixes = 1;
        } else {
            t->prefixes = t->prefixes + 1;
        }
        t->addWord(s, pos + 1);
    }

    int count(string s, int pos = 0, int k = 0) {
        if(pos == s.size()) return k;
        Trie *t = child[s[pos] - 'a'];
        if(t == NULL) return 0;
        return t->count(s, pos + 1, (prefixes == t->prefixes) ? k: k + 1);
    }
};
```

```
    }
};
```

39 Data Structures - Union Find

```
//Union Find
struct UnionFind {
    int N, *id, *sz;

    UnionFind(int _N) {
        id = new int[_N];
        sz = new int[_N];
        for(int i = 0; i < _N; i++) {
            id[i] = i;
            sz[i] = 1;
        }
        N = _N;
    }
    int root(int i) {
        while(i != id[i]) {
            id[i] = id[id[i]];
            i = id[i];
        }
        return i;
    }
    bool find(int p, int q) {
        return root(p) == root(q);
    }
    void unite(int p, int q) {
        int i = root(p);
        int j = root(q);
        if(i == j) return;
        if(sz[i] < sz[j]) {
            id[i] = j; sz[j] += sz[i];
        } else {
            id[j] = i; sz[i] += sz[j];
        }
    }
};
```

40 Games - Nim-Misere

```
int curr = 0;
bool has = false;

for (int i = 0; i < N; i++) {
    if (i == 0) {
        curr = P[i];
    } else {
        curr ^= P[i];
    }

    if (P[i] > 1) {
        has = true;
    }
 }

if (has) {
    if (curr != 0) {
        cout << "F";
    } else {
        cout << "S";
    }
 } else {
    if (curr == 0) {
        cout << "F";
    } else {
        cout << "S";
    }
 }
```

41 Geometry - Area Of Iintersecting Circles

```
typedef long double ld;

const ld PI = acos(-1.);
const ld EPS = 1e-8;

ld dist(Point& a, Point& b) {
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
```

```
ld get_area(Point& c1, Point& c2, ld r1, ld r2) {
    ld d = dist(c1, c2), ans;

    if(r1 > r2) {
        ld aux = r1;
        r1 = r2;
        r2 = aux;
    }

    if(d >= r1 + r2 - EPS) {
        ans = 0.;
    } else if(r1 + d <= r2 + EPS) {
        ans = PI * r1*r1;
    } else {
        ld alfa = acos((r2*r2 + d*d - r1*r1)/(2*r2*d));
        ld beta = acos((r1*r1 + d*d - r2*r2)/(2*r1*d));

        ld A1 = alfa * r2*r2;
        ld A2 = beta * r1*r1;

        ld At1 = r2*r2 * sin(alfa) * cos(alfa);
        ld At2 = r1*r1 * sin(beta) * cos(beta);

        ans = (A1 - At1) + (A2 - At2);
    }

    return ans;
}
```

42 Geometry - Circle Line Intersection

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

/*
  Given a circle with center C(x0, y0) and radius r,
  a line determined by the equation y = mx + b, check if there is
  a intersection then get the intersection points;;
*/

const double EPS = 1e-9;
const double PI = acos(-1);

struct point {
    double x, y;

    point() {}
    point(double x, double y): x(x), y(y) {}

    bool operator<(const point& r) const {
        if(abs(x - r.x) < EPS) return y < r.y;
        return x < r.x;
    }

    bool operator==(const point& r) const {
        return abs(x - r.x) < EPS &&
               abs(y - r.y) < EPS;
    }

    point operator-(const point& r) const {
        return point(x - r.x, y - r.y);
    }
};

vector<point> cl_intersection(point& P, double r, double m, double b) {
    double p = P.x, q = P.y;

    double A = m*m + 1;
    double B = 2*(m*b - m*q - p);
    double C = (q*q - r*r + p*p - 2*b*q + b*b);

    double delta = B*B - 4*A*C;

    vector<point> ans;
    if(delta > 0) {
        double ax = (-B + sqrt(delta)) / (2*A);
        double bx = (-B - sqrt(delta)) / (2*A);

        ans.push_back(point(ax, m*ax + b));
        ans.push_back(point(bx, m*bx + b));
    } else if(delta == 0) {
        double ax = (-B) / (2*A);

        ans.push_back(point(ax, m*ax + b));
    }

    return ans;
```

```
}

int main(void) {
    point c = point(0, 0);
    vector<point> p = cl_intersection(c, 5, 1, 1);

    for(int i=0; i<p.size(); i++) {
        cout << p[i].x << " " << p[i].y << "\n";
    }

    return 0;
}
```

43 Geometry - Closest Pair

```
///----Closes pair with divide and conquer----///
struct point{
    double x, y;
    point(double a, double b): x(a), y(b){}
    point(){};
};

bool compareX(point a, point b){
    return a.x < b.x;
}

bool compareY(point a, point b){
    return a.y < b.y;
}

double bruteForce(vector<point> &p){
    double ans = 40000.*40001.;
    for(int i = 0; i < p.size(); i++){
        for(int j = i + 1; j < p.size(); j++){
            double dst = hypot(p[j].x - p[i].x, p[j].y - p[i].y);
            if(dst < ans){
                ans = dst;
            }
        }
    }
    return ans;
}

double strip(vector<point> &p, double d){
    sort(p.begin(), p.end(), compareY);
    double ans = d;
    for(int i = 0; i < p.size(); i++){
        for(int j = i + 1; j < p.size() && (p[j].y - p[i].y) < d; j++){
            double dst = hypot(p[j].x - p[i].x, p[j].y - p[i].y);
            if(dst < ans){
                ans = dst;
            }
        }
    }
    return ans;
}

double X, Y;
int n;
double closest(vector<point> v){
    int n = v.size();
    if(n <= 3){
            return bruteForce(v);
    }
    vector<point> left;
    vector<point> right;
    int mid = n >> 1;
    for(int i = 0; i < mid; i++){
        left.push_back(v[i]);
    }
    for(int i = mid; i < n; i++){
        right.push_back(v[i]);
    }

    double lh = closest(left);
    double rh = closest(right);
    double d = min(lh,rh);
    vector<point> stripArray;
    for(int i = 0; i < n; i++){
        if(fabs(v[i].x - v[mid].x) < d){
            stripArray.push_back(v[i]);
        }
    }
    return min(d, strip(stripArray,d));
}

sort(pos.begin(), pos.begin()+n, compareX);
```

```
double ans = closest(pos);
```

44 Geometry - Closest-Pair-Sweepline

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

typedef pair<double, double> point;

#define x second
#define y first

int n;
point p[10010];

bool cmp(const point& a, const point& b) {
    return a.x < b.x;
}

double dist(point a, point b) {
    return sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
}

double bruteForce() {
    double ans = 1e20;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) continue;
            ans = min(ans, dist(p[i], p[j]));
        }
    }

    return ans;
}

double sweepLine() {
    double ans = 1e20;

    sort(p, p + n, cmp);

    set<point> box;
    box.insert(p[0]);
    int lm = 0;

    for (int i = 1; i < n; i++) {
        while (lm < i && p[i].x - p[lm].x > ans) {
            box.erase(p[lm++]);
        }

        point head(p[i].y - ans, p[i].x - ans);

        auto low = box.lower_bound(head);

        for (auto it = low; it != box.end() && p[i].y + ans >= it->y;
             it++) {
            ans = min(ans, dist(p[i], (*it)));
        }

        box.insert(p[i]);
    }

    return ans;
}

int main(void) {
    while (scanf("%d", &n) && n) {
        for (int i = 0; i < n; i++) {
            scanf("%lf%lf", &p[i].x, &p[i].y);
        }

        double ans;

        if (n < 75) {
            ans = bruteForce();
        }
        else {
            ans = sweepLine();
        }

        if (ans >= 10000.) {
            printf("INFINITY\n");
        }
        else {
            printf("%.4lf\n", ans);
        }
    }
```

```
    return 0;
}
```

45 Geometry - Closest$_{pair}(line - sweep)$

```
// if there are less than 75 points run brute force !!
// in this case a Point is (y, x) instead of (x, y)
// ex: read x and y -> Point p = Point(y, x)
double dist(Point& a, Point& b) {
    return sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
}

double bruteForce(vector<Point> p) {
    double ans = 1e20;

    for (int i = 0; i < p.size(); i++) {
        for (int j = i + 1; j < p.size(); j++) {
            ans = min(ans, dist(p[i], p[j]));
        }
    }

    return ans;
}

double sweepLine(vector<Point> p) {
    double ans = 1e20;

    sort(p.begin(), p.end());

    set<Point> box;
    box.insert(p[0]);
    int lm = 0;

    for (int i = 1; i < n; i++) {
        while (lm < i && p[i].x - p[lm].x > ans) {
            box.erase(p[lm++]);
        }

        Point head(p[i].y - ans, p[i].x - ans);

        set<Point>::iterator low = box.lower_bound(head);

        for (set<Point>::iterator it = low;
             it != box.end() && p[i].y + ans >= it->y; it++) {
            ans = min(ans, dist(p[i], (*it)));
        }

        box.insert(p[i]);
    }

    return ans;
}
```

46 Geometry - Convex Hull

```
//Convex Hull
struct point {
    int x, y;
    point(int x, int y): x(x), y(y){}
    point(){}
    bool operator <(const point &p) const {
        return x < p.x || (x == p.x && y < p.y);
    }
    bool operator==(const point &p) const {
        return x == p.x && y == p.y;
    }
};
ll cross(const point &O, const point &A, const point &B) {
    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}

vector<point> convex_hull(vector<point> &P) {
    int n = P.size(), k = 0;
    vector<point> H(2*n);

    sort(P.begin(), P.end());

    for (int i = 0; i < n; i++) {
        while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= /*change to < to
            remove equal points */ 0) k--;
        H[k++] = P[i];
    }
    for (int i = n-2, t = k+1; i >= 0; i--) {
```

```
        while (k >= t && cross(H[k-2], H[k-1], P[i]) <= /*change to < to
            remove equal points */ 0) k--;
        H[k++] = P[i];
    }
    H.resize(k);
    return H;
}
```

47 Geometry - Convex Polygon Area

```
//Area de um Poligono Convexo
double area() {
        int N = 4;

        //Points
        int[] x = { 2, -4, 5, 2 };
        int[] y = { 5, 3, 1, 5 };

        double ma = x[N - 1] * y[0], mb = x[0] * y[N - 1];

        for (int i = 0; i < N - 1; i++) {
                ma += (x[i] * y[i + 1]);
                mb += (x[i + 1] * y[i]);
        }

        double ans = Math.abs((ma - mb) * 0.5);
}
```

48 Geometry - Distance Point Line Segment

```
// a and b are points of the segment and p the query point
double dist_point_seg(Point& a, Point& b, Point& p) {
    double dx = b.x - a.x;
    double dy = b.y - a.y;

    double s = dx*dx + dy*dy;

    double u = ((p.x - a.x) * dx + (p.y - a.y) * dy) / s;

    if(u > 1) {
        u = 1;
    } else if(u < 0) {
        u = 0;
    }

    double x1 = a.x + u * dx;
    double y1 = a.y + u * dy;

    double x2 = x1 - p.x;
    double y2 = y1 - p.y;

    return sqrt(x2*x2 + y2*y2);
}
```

49 Geometry - Geometry Utils

```
//Point structure
//Piece of code stracted from the hichhikin guide to programming
//start from any initial values.

const double PI = 2.0*acos(0.0);
const double EPS = 1e-9; //too small/big?????
struct PT {
        double x,y;
        double length() {
                return sqrt(x*x+y*y);
        }
        int normalize() {
                double l = length();
                if(fabs(l)<EPS) return -1;
                x/=l; y/=l;
                return 0;
        }
        PT operator-(PT a) {
                PT r;
                r.x=x-a.x; r.y=y-a.y;
                return r;
        }
        PT operator+(PT a){
                PT r;
                r.x=x+a.x; r.y=y+a.y;
```

```
                return r;
        }
        PT operator*(double sc) {
                PT r;
                r.x=x*sc; r.y=y*sc;
                return r;
        }
};
bool operator<(const PT& a,const PT& b) {
        if(fabs(a.x-b.x)<EPS) return a.y<b.y;
        return a.x<b.x;
}
double dist(PT& a, PT& b){
        return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
double dot(PT& a, PT& b) {
        return(a.x*b.x+a.y*b.y);
}
r.x=x*sc; r.y=y*sc;
// Areas
// ================================================================
double trap(PT a, PT b) {
        return (0.5*(b.x - a.x)*(b.y + a.y));
}
double area(vector<PT> &vin) {
        // Area of a simple polygon, not neccessary convex
        int n = vin.size();
        double ret = 0.0;
        for(int i = 0; i < n; i++) {
                ret += trap(vin[i], vin[(i+1)%n]);
        }
        return fabs(ret);
}
double peri(vector<PT> &vin) {
        // Perimeter of a simple polygon, not neccessary convex
        int n = vin.size();
        double ret = 0.0;
        for(int i = 0; i < n; i++) {
                ret += dist(vin[i], vin[(i+1)%n]);
        }
        return ret;
}
double triarea(PT a, PT b, PT c) {
        //Triangle area
        return fabs(trap(a,b)+trap(b,c)+trap(c,a));
}
```

```
}
double height(PT a, PT b, PT c) {
        // height from a to the line bc
        double s3 = dist(c, b);
        double ar = triarea(a,b,c);
        return (2.0*ar/s3);
}
//************************************
//Check wheter a polygon is convex
int sideSign(PT& p1,PT& p2,PT& p3) {
        // which side is p3 to the line p1->p2? returns: 1 left, 0 on, -1
            right
        double sg = (p1.x-p3.x)*(p2.y-p3.y)-(p1.y - p3.y)*(p2.x-p3.x);
        if (fabs(sg)<EPS) return 0;
        if (sg>0) return 1;
        return -1;
}
int isConvex(vector<PT>& v) {
        // test whether a simple polygon is convex
        // return 0 if not convex, 1 if strictly convex,
        // 2 if convex but there are points unnecesary
        // this function does not work if the polycon is self intersecting
        // in that case, compute the convex hull of v, and see if both
            have the same area
        int i,j,k;
        int c1=0; int c2=0; int c0=0;
        int n=v.size();
        for(i = 0;i < n; i++) {
                j= (i+1)%n;
                k= (j+1)%n;
                int s = sideSign(v[i], v[j], v[k]);
                if (s == 0) c0++;
                if (s > 0) c1++;
                if (s < 0) c2++;
        }
        if(c1 && c2) return 0;
        if(c0) return 2;
        return 1;
}
// ================================================
// Points and Lines
// ================================================
int intersection( PT p1, PT p2, PT p3, PT p4, PT &r) {
        // two lines given by p1->p2, p3->p4 r is the intersection point
        // return -1 if two lines are parallel
```

```
        double d = (p4.y - p3.y)*(p2.x-p1.x) - (p4.x - p3.x)*(p2.y - p1.y);
        if( fabs( d ) < EPS ) return -1;
        // might need to do something special!!!
        double ua, ub;
        ua = (p4.x - p3.x)*(p1.y-p3.y) - (p4.y-p3.y)*(p1.x-p3.x);
        ua /= d;
        // ub = (p2.x - p1.x)*(p1.y-p3.y) - (p2.y-p1.y)*(p1.x-p3.x);
        //ub /= d;
        r = p1 + (p2-p1)*ua;
        return 0;
}

void closestpt( PT p1, PT p2, PT p3, PT &r) {
        // the closest point on the line p1->p2 to p3
        if (fabs( triarea( p1, p2, p3)) < EPS) {
                r = p3;
                return;
        }
        PT v = p2-p1;
        v.normalize();
        double pr; // inner product
        pr = (p3.y-p1.y)*v.y + (p3.x-p1.x)*v.x;
        r = p1+v*pr;
}
int hcenter( PT p1, PT p2, PT p3, PT& r) {
        // point generated by altitudes
        if (triarea( p1, p2, p3 ) < EPS) return -1;
        PT a1, a2;
        closestpt( p2, p3, p1, a1 );
        closestpt( p1, p3, p2, a2 );
        intersection( p1, a1, p2, a2, r );
        return 0;
}
int center( PT p1, PT p2, PT p3, PT& r) {
        // point generated by circumscribed circle
        if (triarea( p1, p2, p3 ) < EPS) return -1;
        PT a1, a2, b1, b2;
        a1 = (p2+p3)*0.5;
        a2 = (p1+p3)*0.5;
        b1.x = a1.x - (p3.y-p2.y);
        b1.y = a1.y + (p3.x-p2.x);
        b2.x = a2.x - (p3.y-p1.y);
        b2.y = a2.y + (p3.x-p1.x);
        intersection(a1, b1, a2, b2, r);
        return 0;
}
```

50 Geometry - KD-Tree

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

struct point {
    Int x, y, z;
    point(Int x=0, Int y=0, Int z=0): x(x), y(y), z(z) {}
    point operator-(point q) { return point(x-q.x, y-q.y, z-q.z); }
    Int operator*(point q) { return x*q.x + y*q.y + z*q.z; }
};
typedef vector<point> polygon;

struct KDTreeNode {
    point p;
    int level;
    KDTreeNode *left, *right;

    KDTreeNode (const point& q, int levl) {
        p = q;
        level = levl;
        left = right = 0;
    }
    ~KDTreeNode() { delete left; delete right; }

    int diff (const point& pt) {
        switch (level) {
        case 0: return pt.x - p.x;
        case 1: return pt.y - p.y;
        case 2: return pt.z - p.z;
        }
        return 0;
    }
    Int distSq (point& q) { return (p-q)*(p-q); }

    int rangeCount (point& pt, Int K) {
        int count = (distSq(pt) < K*K) ? 1 : 0;
        int d = diff(pt);
        if (-d <= K && right != 0)
```

```
            count += right->rangeCount(pt, K);
        if (d <= K && left != 0)
            count += left->rangeCount(pt, K);
        return count;
    }
};

class KDTree {
public:
    polygon P;
    KDTreeNode *root;
    int dimention;
    KDTree() {}
    KDTree(polygon &poly, int D) {
        P = poly;
        dimention = D;
        root = 0;
        build();
    }
    ~KDTree() { delete root; }

    //count the number of pairs that has a distance less than K
    Int countPairs(Int K) {
        Int count = 0;
               for (int i = 0; i < (int) P.size(); i++) {
            count += root->rangeCount(P[i], K) - 1;
               }
        return count;
    }

protected:
    void build() {
        random_shuffle(P.begin(), P.end());
               for (int i = 0; i < (int) P.size(); i++) {
            root = insert(root, P[i], -1);
        }
    }
    KDTreeNode *insert(KDTreeNode* t, const point& pt, int parentLevel) {
        if (t == 0) {
            t = new KDTreeNode (pt, (parentLevel+1) % dimention);
            return t;
        } else {
            int d = t->diff(pt);
            if (d <= 0) t->left = insert (t->left, pt, t->level);
            else t->right = insert (t->right, pt, t->level);
        }
        return t;
    }
};

int main() {
    int n, k;
    point e;
    polygon p;
    while (cin >> n >> k && n+k) {
        p.clear();
               for (int i = 0; i < n; i++) {
            cin >> e.x >> e.y >> e.z;
            p.push_back(e);
        }
        KDTree tree(p, 3);
        cout << tree.countPairs(k) / 2LL << endl;
    }
    return 0;
}
```

51 Geometry - Line Point Distance

```
//Distance between point - line
double dot(pair<int, int> &A, pair<int, int> &B, pair<int, int> &C) {
    return (double) (B.first - A.first) * (C.first - B.first) + (B.second
        - A.second) * (C.second - B.second);
}

double cross(pair<int, int> &A, pair<int, int> &B, pair<int, int> &C) {
    return (double) (B.first-A.first) * (C.second-A.second) -
        (B.second-A.second) * (C.first-A.first);
}

double _distance(pair<int, int> A, pair<int, int> B) {
    int d1 = A.first - B.first;
    int d2 = A.second - B.second;
    return sqrt(d1*d1+d2*d2);
}

double linePointDist(pair<int, int> A, pair<int, int> B, pair<int, int>
    C, bool isSegment) {
```

```
    double dist = cross(A,B,C) / _distance(A,B);
    if(isSegment) {
        int dot1 = dot(A,B,C);
        if(dot1 > 0)return _distance(B,C);
        int dot2 = dot(B,A,C);
        if(dot2 > 0)return _distance(A,C);
    }
    return abs(dist);
}
```

52 Geometry - Line Point Intesection

```
struct Point
{
    int x;
    int y;
};

// Given three colinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r) {
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y)) {
       return true;
        }

    return false;
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r) {
    // See 10th slides from following link for derivation of the formula
    // http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear

    return (val > 0)? 1: 2; // clock or counterclock wise
}

// The main function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    // Find the four orientations needed for general and
    // special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    // General case
    if (o1 != o2 && o3 != o4)
        return true;

    // Special Cases
    // p1, q1 and p2 are colinear and p2 lies on segment p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;

    // p1, q1 and p2 are colinear and q2 lies on segment p1q1
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;

    // p2, q2 and p1 are colinear and p1 lies on segment p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;

     // p2, q2 and q1 are colinear and q1 lies on segment p2q2
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;

    return false; // Doesn't fall in any of the above cases
}
```

53 Geometry - Line2D

```
struct Line {
    double a, b, c;

    Line() {}
    Line(double a, double b, double c): a(a), b(b), c(c) {}
    Line(Point p1, Point p2) {
```

```
        if(p1.x == p2.x) { // vertical line treatment
            a = 1.;
            b = 0.;
            c = -p1.x;
        } else {
            a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
            b = 1.; // easier using b = 1
            c = -(double)(a * p1.x) - ( b * p1.y);
        }
    }

    bool contains(Point p) { // check if p is on the line
        return (a*p.x + b*p.y + c) < EPS;
    }

    bool parallel(Line r) { // checking a and b
        return (fabs( a-r.a) < EPS && fabs( b-r.b) < EPS);
    }
    bool collinear(Line r) { // now checking c
        return (parallel(r) && fabs( c-r.c) < EPS);
    }

    // if there is put intersect point on p
    bool intersect(Line r, Point& p) {
        if(collinear(r)) return false; // infinite points
        if(parallel(r)) return false; // no point

        p.x = (double) (r.b * c - b * r.c) / (r.a * b - a * r.b);

        if(b > EPS) { // check if it is not a vertical line
            p.y = -(a * p.x + c) / b;
        } else { // vertical line treatment
            p.y = -(r.a * p.x + r.c) / r.b;
        }
        return true;
    }
};
```

54 Geometry - Minimum Enclosing Circle

```
struct Circle {
    Point center;
    double radius;

    Circle() {}
    Circle(Point _center, double _radius) {
        center = _center;
        radius = _radius;
    }

    bool has_inside(Point p) {
        return hypot(p.x - center.x, p.y-center.y) < radius + EPS;
    }
};

vector<Point> points;

double cross(Point o, Point a, Point b) {
    return ((a.x-o.x)*(b.y-o.y) - (a.y-o.y)*(b.x-o.x));
}

Circle make_diameter(Point p, Point q) {
    Point center((p.x + q.x) / 2., (p.y + q.y) / 2.);
    double radius = hypot(p.x - q.x, p.y - q.y) / 2.;
    return Circle(center, radius);
}

bool make_circumcircle(Circle& ans, Point a, Point b, Point c) {
    double ax = a.x, ay = a.y;
    double bx = b.x, by = b.y;
    double cx = c.x, cy = c.y;

    double d = (ax * (by - cy) + bx * (cy - ay) + cx * (ay - by)) * 2.0;
    if(d == 0.0) return false;
    double xc = ((ax * ax + ay * ay) * (by - cy) +
                 (bx * bx + by * by) * (cy - ay) +
                 (cx * cx + cy * cy) * (ay - by)) / d;
    double yc = ((ax * ax + ay * ay) * (cx - bx) +
                 (bx * bx + by * by) * (ax - cx) +
                 (cx * cx + cy * cy) * (bx - ax)) / d;
    ans = Circle(Point(xc, yc), hypot(xc - ax, yc - ay));
    return true;
}

Circle make_circle_two_points(int right, Point p, Point q) {
    Circle diameter = make_diameter(p, q);
```

```
    bool ok = true;
    for(int i=0; i<=right; i++) {
        if(!diameter.has_inside(points[i])) {
            ok = false;
            break;
        }
    }

    if(ok) return diameter;

    Circle l, r;
    bool nonel = true, noner = true;
    for(int i=0; i<=right; i++) {
        double cr = cross(p, q, points[i]);
        Circle c;
        if(!make_circumcircle(c, p, q, points[i])) continue;

        if(cr > 0.0 &&
           (nonel || cross(p, q, c.center) > cross(p, q, l.center))) {
            l = c;
            nonel = false;
        }
        else if(cr < 0.0 &&
                (noner || cross(p, q, c.center) < cross(p, q, r.center))) {
            r = c;
            noner = false;
        }
    }
    if(noner || (!nonel && l.radius <= r.radius)) return l;
    return r;
}

Circle make_circle_one_point(int right, Point p) {
    Circle ans(p, 0.0);
    for(int i=0; i<=right; i++) {
        if(!ans.has_inside(points[i])) {
            if(ans.radius == 0.0) {
                ans = make_diameter(p, points[i]);
            } else {
                ans = make_circle_two_points(i, p, points[i]);
            }
        }
    }
    return ans;
}
```

```
Circle make_circle(vector<Point> points) {
    Circle ans;
    bool none = true;

    for(int i=0; i<points.size(); i++) {
        if(none || !ans.has_inside(points[i])) {
            ans = make_circle_one_point(i, points[i]);
            none = false;
        }
    }
    return ans;
}
```

55 Geometry - Point Inside Triangle

```
/* A utility function to calculate area of triangle formed by (x1, y1),
   (x2, y2) and (x3, y3) */
float area(int x1, int y1, int x2, int y2, int x3, int y3) {
   return abs((x1*(y2-y3) + x2*(y3-y1)+ x3*(y1-y2))/2.0);
}

/* A function to check whether point P(x, y) lies inside the triangle
     formed
   by A(x1, y1), B(x2, y2) and C(x3, y3) */
bool isInside(int x1, int y1, int x2, int y2, int x3, int y3, int x, int
     y) {
   /* Calculate area of triangle ABC */
   float A = area (x1, y1, x2, y2, x3, y3);

   /* Calculate area of triangle PBC */
   float A1 = area (x, y, x2, y2, x3, y3);

   /* Calculate area of triangle PAC */
   float A2 = area (x1, y1, x, y, x3, y3);

   /* Calculate area of triangle PAB */
   float A3 = area (x1, y1, x2, y2, x, y);

   /* Check if sum of A1, A2 and A3 is same as A */
   return (A == A1 + A2 + A3);
}
```

56 Geometry - Point2D-double

```
const double EPS = 1e-9;
const double PI = acos(-1);

double DEG_to_RAD(double theta) {
    return theta*PI/180.;
}

double RAD_to_DEG(double theta) {
    return theta*180. / PI;
}

struct Point {
    double x, y;

    Point() {}
    Point(double x, double y): x(x), y(y) {}

    // vector (0, 0) -> (x, y)
    double mod() { return hypot(x, y); }
    double angle() { return atan2(y, x); }
    double dist(Point p) {
        return hypot(x - p.x, y - p.y);
    }
    // rotate point by theta degrees CCW r.t. origin
    void rotate(double theta) {
        theta = DEG_to_RAD(theta);
        x = x * cos(theta) - y * sin(theta);
        y = x * sin(theta) + y * cos(theta);
    }

    bool operator<(const Point& r) const {
        if(fabs(x-r.x) < EPS) return y < r.y;
        return x < r.x;
    }

    bool operator==(const Point& r) const {
        return (fabs(x-r.x) < EPS && fabs(y-r.y) < EPS);
    }

    Point operator-(const Point& r) const {
        return Point(x-r.x, y-r.y);
    }
};
```

57 Geometry - Point2D-int

```
const double EPS = 1e-9;
const double PI = acos(-1);

double DEG_to_RAD(double theta) {
    return theta*PI/180.;
}

double RAD_to_DEG(double theta) {
    return theta*180. / PI;
}

struct Point {
    int x, y;

    Point() {}
    Point(int x, int y): x(x), y(y) {}

    // vector (0, 0) -> (x, y)
    double mod() { return hypot(x, y); }
    double angle() { return atan2(y, x); }
    double dist(Point p) {
        return hypot(x - p.x, y - p.y);
    }
    // rotate point by theta degrees CCW r.t. origin
    void rotate(double theta) {
        theta = DEG_to_RAD(theta);
        x = x * cos(theta) - y * sin(theta);
        y = x * sin(theta) + y * cos(theta);
    }

    bool operator<(const Point& r) const {
        if(x == r.x) return y < r.y;
        return x < r.x;
    }

    bool operator==(const Point& r) const {
        return (x == r.x && y == r.y);
    }
```

```
    Point operator-(const Point& r) const {
        return Point(x-r.x, y-r.y);
    }
};
```

58 Geomtry - Distance Point Line

```
// returns the distance from p to the line defined by
// two points a and b (a and b must be different)
// the closest point is stored in the 4th parameter (byref)
double distToLine(Point p, Point a, Point b, Point &c) {
    // formula: c = a + u * ab
    Vector ap = Vector(a, p), ab = Vector(a, b);
    double u = ap.dot(ab) / (ab.mod()*ab.mod());
    ab.scale(u);
    ab.translate(a);
    c = a;
    return p.dist(c);
}
```

59 Geomtry - Triangle Area

```
double area(double a, double b, double c) {
       double s = (a + b + c) / 2.0;

       double area = sqrt(s * (s - a) * (s - b) * (s - c));

       return area;
}
```

60 Geomtry - Vector2D-double

```
struct Vector {
    double x, y;

    Vector() {}
    Vector(double x, double y): x(x), y(y) {}
    Vector(Point p1, Point p2) { // vector p1 -> p2
        x = p2.x - p1.x;
        y = p2.y - p1.y;
    }

    double mod() { return hypot(x, y); }
    double angle(Vector v) {
        return acos(dot(v) / (mod() * v.mod()));
    }
    void scale(double s) {
        x *= s;
        y *= s;
    }
    // translate p in the direction of this vector
    void translate(Point& p) {
        p.x += x;
        p.y += y;
    }
    double dot(Vector v) {
        return (x * v.x + y * v.y);
    }
    double cross(Vector v) {
        return (x * v.y - y * v.x);
    }
};
```

61 Graph - 2-SAT

```
map<string, int> mp;
string c1, c2;
int vis[MAXN], low[MAXN], num[MAXN], sat[MAXN],dfs_counter, scc_counter
    ,n, test = 1;
vector<vector<int> > graph(MAXN);
stack<int> st;

//Find strongly connected components
void tarjan(int u, int depth) {
    low[u] = num[u] = depth;
    st.push(u);
    vis[u] = 1;
    for(int i = 0; i < graph[u].size(); i++) {
```

```
        int v = graph[u][i];
        if(num[v] == -1){
            tarjan(v, depth+1);
        }
        if (vis[v]){
            low[u] = min(low[u], low[v]);
        }
    }
    if(low[u] == depth) {
        while(1) {
            int next = st.top();st.pop();
            sat[next] = scc_counter;
            vis[next] = 0;
            if(u == next) break;
        }
        ++scc_counter;
    }
}

/*
  To use the 2-SAT property, it could be transformed in a boolean logic
      with AND or OR
  A | B, means !A -> B
  !A | B, means !!A -> B equals to A -> B
  A | !B means !A -> !B
  !A | !B means !!A -> !B equals to A -> !B

  "-> is an implicance to separate the usege of A AND B"

  2-sat property is YES if all the components have no disturbs, e.g.:
  If you find !A and A in the same "scc", you're talking that !A = true
      AND A = true, it is not right.
*/
int main(void){
    ios::sync_with_stdio(0);
    while(cin >> n){
        mp.clear();
        for(int i = 0; i < MAXN; i++){
            graph[i].clear();
            vis[i] = 0;
            num[i] = -1;
            sat[i] = -1;
            low[i] = 0;
        }
        cin.ignore();
        int index = 0;
        //Graph mount:
        for(int i = 0; i < n; i++){
            cin >> c1 >> c2;
            string tmpA, tmpB;
            bool A = 1, B = 1;
            if(c1[0] == '!'){
                tmpA = c1.substr(1);
                A = 0;
            }else{
                tmpA = c1;
            }
            if(c2[0] == '!'){
                tmpB = c2.substr(1);
                B = 0;
            }else{
                tmpB = c2;
            }
            if(mp.find(tmpA) == mp.end()){
                mp[tmpA] = index;
                index += 2;
            }
            if(mp.find(tmpB) == mp.end()){
                mp[tmpB] = index;
                index += 2;
            }
            int U = mp[tmpA] + !A, V = mp[tmpB] + !B;
            graph[U^1].push_back(V);
            graph[V^1].push_back(U);
        }

        dfs_counter = scc_counter = 0;
        for(int i = 0; i < index; i++){
            if(num[i] == -1){
                tarjan(i, 0);
            }
        }
        int win = 1;
        //Checking disturbs
        for(int i = 0; i < index && win; i += 2){
            if(sat[i^1] == sat[i]) win = 0;
        }
        printf("Instancia %d\n", test++);
        if(win){
            printf("sim\n");
```

```
        }else{
            printf("nao\n");
        }
        printf("\n");
    }
    return 0;
}
```

62 Graph - Articulation Point in Graph

```
vector<int> graph[410];
set<int> ans;
set<int>::iterator it;

int dfs(int u){
    int less = vis[u] = times++;
    int filhos = 0;
    for(int i = 0; i< graph[u].size(); i++){
       if(vis[graph[u][i]]==0){
          filhos++;
          int m = dfs(graph[u][i]);
          less = min(less,m);
          if(vis[u] <= m && (u != 0 || filhos >= 2)){
              ans.insert(u);
          }
       }else{
          less = min(less, vis[graph[u][i]]);
       }
    }
    return less;
}
times = 1;
ans.clear();
dfs(0);
```

63 Graph - Articulation Vertex

```
set<int> ans;
int times;

int dfs(int u){
    int less = vis[u] = times++;
    int filhos = 0;
    for(int i = 0; i< graph[u].size(); i++){
       if(vis[graph[u][i]]==0){
          filhos++;
          int m = dfs(graph[u][i]);
          less = min(less,m);
          if(vis[u] <= m && (u != 0 || filhos >= 2)){
              ans.insert(u);
          }
       }else{
          less = min(less, vis[graph[u][i]]);
       }
    }
    return less;
}

times = 1;
ans.clear();
dfs(0);
```

64 Graph - Bellman Ford

```
vector <pair<int, int> > edges;
int graph[MAXN][MAXN];
int dist[MAXN];

int N;
bool bellman_ford(int s) {
       int M = edges.size();
       memset (dist, INF, sizeof(int)*n);
       dist[s] = 0;
       for (int k = 0; k < N-1; ++k) {
              for (int j = 0; j < M; ++j) {
                     int u = edges[j].first;
                     int v = edges[j].second;
                     if (dist[u] < INF && dist[v] > dist[u] +
                         graph[u][v])
                            dist[v] = dist[u] + graph[u][v];
              }
```

```
        }
        //Negative Cycle
        for (int j = 0; j < m; ++j) {
                int u = edges[j].first, v = edges[j].second;
                if (dist[u] < INF && dist[v] > dist[u] + graph[u][v]) {
                        return false;
                }
        }
        return true;
}
```

65 Graph - Bipartite Check Algorithm

```
bool dfs(int node, int c) {
    if(color[node] != 0) {
        if(color[node] == c) {
            return true;
        } else {
            return false;
        }
    }
    color[node] = c;
    for(int i = 1; i <= n; i++)
                if(gr[node][i] == 1) {
                        if(!dfs(i, -c)) {
                                return false;
                        }
        }
    }
    return true;
}
```

66 Graph - Bridges

```
int dfsct, bridges, num[MAXN], low[MAXN], parent[MAXN];

void bridge(int atual){
    num[atual] = low[atual] = dfsct++;
    for(int i = 0; i < graph[atual].size(); i++){
        int next = graph[atual][i];
        if(num[next] == -1){
            parent[next] = atual;
            bridge(next);
            if(low[next] > num[atual]){
                bridges++;
            }
            low[atual] = min(low[atual], low[next]);
        }else if(next != parent[atual]){
            low[atual] = min(low[atual], num[next]);
        }
    }
}

void countBridges(){
    dfsct = bridges = 0;
    for(int i = 0; i < n; i++){
        num[i] = -1;
        parent[i] = 0;
    }
    for(int i = 0; i < n; i++){
        if(num[i] == -1) bridge(i);
    }
}
```

67 Graph - Cycle Retrieval Algorithm

```
//It only works in graphs without compound cycles
bool inq[MAXN], vis[MAXN];

void dfs(int node, int parent, int len) {
        vis[node] = true;
        cle[node] = len;

        stk[stk_pointer++] = node;
        inq[node] = true;

        for (int i = 0; i < (int) graph[node].size(); i++) {
                int next = graph[node][i].first;
                int cost = graph[node][i].second;

                if (next == parent) continue;
```

```
            if (!vis[next]) {
                dfs(next, node, len + cost);
            } else {
                if (inq[next]) {
                    int curr;
                    int real_len = len + cost - cle[next];

                    while (stk_pointer > 0) {
                        curr = stk[--stk_pointer];
                        inq[curr] = false;
                        cycle_len[curr] = real_len;
                        if (curr == next) break;
                    }
                }
            }
        }

        if (inq[node]) {
            while (stk_pointer > 0) {
                inq[stk[stk_pointer-1]] = false;
                if (stk[stk_pointer-1] == node) {
                    stk_pointer--;
                    break;
                }
                stk_pointer--;
            }
        }
}

stk_pointer = 0;
dfs(1, -1, 0);
```

68 Graph - Dijkstra Algorithm

```
struct MyLess {
    bool operator()(int x, int y) {
        return dist[x] > dist[y];
    }
};

int dijsktra(int source, int destiny) {
        for(int i = 0; i <= 110; i++) {
            dist[i] = INT_MAX;
        }
        priority_queue<int, vector<int>, MyLess> q;
        dist[source] = 0;
        q.push(source);

        while(!q.empty()) {
            int tmp = q.top(); q.pop();
            for(int i = 0; i < graph[tmp].size(); i++) {
                int aux_dist = dist[tmp] + graph[tmp][i].second;
                int actual_dist = dist[graph[tmp][i].first];
                if(aux_dist < actual_dist) {
                    dist[graph[tmp][i].first] = aux_dist;
                    q.push(graph[tmp][i].first);
                }
            }
        }
    }
        return dist[destiny];
}
// Reconstruo  do Caminho
vector<int> path;
int start = destiny;

while(start != -1) {
        path.push_back(start);
        start = prev[start];
}
```

69 Graph - Floyd Warshall

```
//Floyd-Warshall - O(n^3)
for(int k = 0; k < n; k++) {
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                dist[i][j] = min(dist[i][j], dist[i][k] +
                    dist[k][j]);
            }
        }
}
```

70 Graph - Hungarian Algorithm

```
const int MAXN = ;
typedef int num;
const num inf = 100000000;
int N, MA[MAXN], MB[MAXN], PB[MAXN], mn[MAXN];
num c[MAXN][MAXN], d[MAXN];

bool S[MAXN], T[MAXN];
int st[MAXN], sn;
num y[MAXN], z[MAXN];

void reset_all() {
    // need to be changed for costs < 0
    for(int i = 0; i < MAXN; i++)
        y[i] = z[i] = num(0);
    int i;
    for(i = 0; i < N; i++)
        MA[i] = MB[i] = -1;
}

bool increase(int b) {
    int a = PB[b];
    while(true) {
        int n_b = MA[a];
        MB[b] = a;
        MA[a] = b;
        if(n_b == -1) break;
        b = n_b;
        a = PB[b];
    }
    return true;
}

// O(n)
bool visit(int a) {
    S[a] = true;
    for(int b = 0; b < N; b++) {
        if(T[b]) continue;
        if(c[a][b] - y[a] - z[b] < d[b]) {
            d[b] = c[a][b] - y[a] - z[b];
            mn[b] = a;
        }
        if(c[a][b] == y[a] + z[b]) {
            T[b] = true;
            PB[b] = a;
            st[sn++] = b;
            if(MB[b] == -1)
                return increase(b);
        }
    }
    return false;
}

// O(n)
bool update_dual() {
    int mb = -1, b;
    for(b = 0; b < N; b++)
        if(!T[b] && (mb == -1 || d[b] < d[mb]))
            mb = b;
    num e = d[mb];
    for(b = 0; b < N; b++)
        if(T[b]) z[b] -= e;
        else d[b] -= e;
    for(int a = 0; a < N; a++)
        if(S[a]) y[a] += e;
    PB[mb] = mn[mb];
    if(MB[mb] == -1) return increase(mb);
    st[sn++] = mb;
    T[mb] = true;
    return false;
}

// O(n^2)
void find_path() {
    int i, a;
    for(a = 0; MA[a] != -1; a++);
    memset(S, 0, sizeof S);
    memset(T, 0, sizeof T);
    for(int i = 0; i < MAXN; i++) d[i] = inf;
    sn = 0;
    if(visit(a)) return;
    while(true) {
        if(sn) { if(visit(MB[st[--sn]])) break; }
        else if(update_dual()) break;
    }
}

num min_match() {
```

```
    reset_all();
    for(int i = 0; i < N; i++)
        find_path();
    num all = 0;
    for(int a = 0; a < N; a++)
        all += c[a][MA[a]];
    return all;
}

int maiN() {
    int i, j;
    scanf("%d", &N);
    for(i = 0; i < N; i++)
        for(j = 0; j < N; j++)
            scanf("%d", &c[i][j]);
    printf("%d\n", min_match());
}
```

71 Graph - Kruskal Algorithm

```
//Kruskal Algorithm
struct edge {
    int from, to, cost;
    edge() {}
    edge(int from, int to, int cost): from(from), to(to), cost(cost) {};

    bool operator<(const edge& e) const {
                return cost < e.cost;
    }
};
//Sendo 'M' o numero de arestas, 'u' uma implementao do conjunto disjunto
    'UnionFind' e 'ans' o menor custo
vector<edge> edges; //Populado com as arestas
int ans = 0;
UnionFind u(N);
for(i = 0; i < m; i++) {
        if(!u.find(edges[i].from, edges[i].to)) {
                u.unite(edges[i].from, edges[i].to);
                ans += edges[i].cost;
        }
}
```

72 Graph - Maximum Bipartite Matching

```
//Maximum Bipartite Matching (Prefereed implementation)
vector<int> graph[MAXN];

bool bpm(int u, bool seen[], int matchR[]) {
    for (int i = 0; i < (int) graph[u].size(); i++) {
                int v = graph[u][i];

        if (!seen[v]) {
            seen[v] = true;

            if (matchR[v] < 0 || bpm(matchR[v], seen, matchR)) {
                matchR[v] = u;
                return true;
            }
        }
    }
    return false;
}

int maxBPM() {
    int matchR[MAXN];

    memset(matchR, -1, sizeof(matchR));

    int result = 0;
    for (int u = 1; u <= C; u++) {
        bool seen[MAXN];
        memset(seen, 0, sizeof(seen));

        if (bpm(u, seen, matchR)) {
            result++;
                }
    }
    return result;
}
```

73 Graph - Maximum Flow

```
const int MAXN = 101010;
const int INF = 101;
```

```
struct edge {
    int to,rev;
    Int cap;
    edge(int to, Int cap, int rev): to(to), cap(cap), rev(rev) {}
};

vector<edge> G[MAXN];
Int level[MAXN];
int iter[MAXN];

void init(int N) {
    for (int i = 0; i < N; i++) {
        G[i].clear();
    }
}

void add_edge(int from,int to,Int cap) {
    G[from].push_back(edge(to, cap, G[to].size()));
    G[to].push_back(edge(from, 0, G[from].size()-1));
}

void bfs(int s) {
    memset(level, -1, sizeof(level));
    queue<int> que;
    level[s] = 0;
    que.push(s);

    while(!que.empty()) {
        int v = que.front();
        que.pop();
        for (int i = 0; i < (int) G[v].size(); i++) {
            edge& e = G[v][i];
            if(e.cap > 0 && level[e.to] < 0) {
                level[e.to] = level[v] + 1;
                que.push(e.to);
            }
        }
    }
}

Int dfs(int v, int t, Int f) {
    if(v == t) return f;
    for(int& i = iter[v]; i < (int) G[v].size(); i++) {
        edge &e = G[v][i];
        if(e.cap > 0 && level[v] < level[e.to]) {
            Int d = dfs(e.to, t, min(f, e.cap));
            if (d > 0) {
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

int max_flow(int s, int t) {
    Int flow = 0;
    for( ; ; ) {
        bfs(s);
        if (level[t] < 0) {
            return flow;
        }
        memset(iter, 0, sizeof(iter));
        int f;
        while ((f=dfs(s,t,INF*INF)) > 0) {
            flow += f;
        }
    }
}
```

74 Graph - Min Cost Max Flow

```
typedef int Flow;
typedef int Cost;
const Flow INF = 0x3f3f3f3f;
struct Edge {
    int src, dst;
    Cost cst;
    Flow cap;
    int rev;
};
bool operator<(const Edge a, const Edge b) {
    return a.cst > b.cst;
}
```

```
typedef vector<Edge> Edges;
typedef vector<Edges> Graph;

void add_edge(Graph&G, int u, int v, Flow c, Cost l) {
    G[u].push_back((Edge){ u, v, l, c, int(G[v].size()) });
    G[v].push_back((Edge){ v, u, -l, 0, int(G[u].size()-1) });
}

pair<Flow, Cost> flow(Graph&G, int s, int t, int K) {
    int n = G.size();
    Flow flow = 0;
    Cost cost = 0;
    for ( ; ; ) {
        priority_queue<Edge> Q;
        vector<int> prev(n, -1), prev_num(n, -1);
        vector<Cost> length(n, INF);
        Q.push((Edge){-1,s,0,0,0});
        prev[s]=s;
        for (;!Q.empty();) {
            Edge e=Q.top();
                    Q.pop();
            int v = e.dst;
            for (int i=0; i<(int)G[v].size(); i++) {
                if (G[v][i].cap>0 &&
                    length[G[v][i].dst]>e.cst+G[v][i].cst) {
                    prev[G[v][i].dst]=v;
                    Q.push((Edge){v, G[v][i].dst, e.cst+G[v][i].cst,0,0});
                    prev_num[G[v][i].dst]=i;
                    length[G[v][i].dst]=e.cst+G[v][i].cst;
                }
            }
        }
        if (prev[t]<0) return make_pair(flow, cost);

        Flow mi=INF;
        Cost cst=0;
        for (int v=t; v!=s; v=prev[v]) {
            mi=min(mi, G[prev[v]][prev_num[v]].cap);
            cst+=G[prev[v]][prev_num[v]].cst;
        }

               K -= cst*mi;
        cost+=cst*mi;

        for (int v=t; v!=s; v=prev[v]) {
            Edge &e=G[prev[v]][prev_num[v]];
            e.cap-=mi;
            G[e.dst][e.rev].cap+=mi;
        }
        flow += mi;
    }

}
```

75 Graph - Prim Algorithm

```
int g[MAXN][MAXN], used[MAXN], min_e[MAXN], sel_e[MAXN];
min_e[0] = 0;
for (int i = 0; i < n; ++i) {
        int v = -1;
        for(int j = 0; j < n; ++j) {
                if (!used[j] && (v == -1 || min_e[j] < min_e[v])) {
                        v = j;
                }
        }
        used[v] = true;
        if (sel_e[v] != -1) {
                ans += min_e[v];
        }
        for (int to = 0; to < n; ++to) {
                if (g[v][to] < min_e[to]) {
                        min_e[to] = g[v][to];
                        sel_e[to] = v;
                }
        }
}
```

76 Graph - Stoer Wagner Algorithm

```
//Global Min-Cut Stoer-Wager O(N^3)
int graph[MAXN][MAXN] //Matrix de Adjacencia do grafo.

int minCut(int n) {
   bool a[n];
```

```
    int v[n];
    int w[n];
    for(int i = 0; i < n; i++) v[i] = i;
    int best = INF;
    while(n > 1) {
        int maxj = 1;
        a[v[0]] = true;
        for(int i = 1; i < n; ++i) {
            a[v[i]] = false;
            w[i] = graph[v[0]][v[i]];
            if(w[i] > w[maxj]) {
                maxj = i;
            }
        }
        int prev= 0 ,buf = n;
        while(--buf) {
            a[v[maxj]]=true;
            if(buf == 1) {
                best = min(best, w[maxj]);
                for(int k = 0; k < n; k++) {
                    graph[v[k]][v[prev]]= (graph[v[prev]][v[k]] +=
                        graph[v[maxj]][v[k]]);
                }
                v[maxj] = v[--n];
            }
            prev = maxj;
            maxj = -1;
            for(int j = 1; j < n; ++j) {
                if(!a[v[j]]) {
                    w[j] += graph[v[prev]][v[j]];
                    if(maxj < 0 || w[j] > w[maxj]) {
                        maxj=j;
                    }
                }
            }
        }
    }
    return best;
}
```

77 Graph - Topological Sort - Iterative

```
priority_queue<int, vector<int>, greater<int> > pq;

for (int i = 0; i < N; i++) {
        if(deg[i] == 0) {
                pq.push(i);
        }
 }
int on = 0;
while (!pq.empty()) {
        int now = pq.top();
        pq.pop();
        order.push_back(now);
        for (int i = 0; i < (int) graph[now].size(); i++) {
                int next = graph[now][i];
                deg[next] -= 1;

                if(deg[next] == 0) {
                        pq.push(next);
                }
        }
 }
```

78 Graph - Topological Sort - Recursive

```
void dfs(int x) {
    vis[x] = 1;
    for(int u = 0; u < n; u++) {
        if(vis[u] == 1 && graph[x][u] == 1) has = true;
        if(vis[u] == 0 && graph[x][u] == 1) {
            dfs(u);
        }
    }
    vis[x] = 2;
    order.push_back(x);
}
```

79 Graph- Dinic Algorithm

```
//Max Flow dinic O(V^2*E)
```

```
const int MAXN = 101010;
const int INF = 101011;

struct edge {
    int to,rev;
    Int cap;
    edge(int to, Int cap, int rev): to(to), cap(cap), rev(rev) {}
};

vector<edge> G[MAXN];
Int level[MAXN];
int iter[MAXN];

void init(int N) {
    for (int i = 0; i < N; i++) {
        G[i].clear();
    }
}

void add_edge(int from,int to,Int cap) {
    G[from].push_back(edge(to, cap, G[to].size()));
    G[to].push_back(edge(from, 0, G[from].size()-1));
}

void bfs(int s) {
    memset(level, -1, sizeof(level));
    queue<int> que;
    level[s] = 0;
    que.push(s);

    while(!que.empty()) {
        int v = que.front();
        que.pop();
        for (int i = 0; i < G[v].size(); i++) {
            edge& e = G[v][i];
            if(e.cap > 0 && level[e.to] < 0) {
                level[e.to] = level[v] + 1;
                que.push(e.to);
            }
        }
    }
}

Int dfs(int v, int t, Int f) {
    if(v == t) return f;
    for(int& i = iter[v]; i < (int) G[v].size(); i++) {
        edge &e = G[v][i];
        if(e.cap > 0 && level[v] < level[e.to]) {
            Int d = dfs(e.to, t, min(f, e.cap));
            if (d > 0) {
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

int max_flow(int s, int t) {
    Int flow = 0;
    for( ; ; ) {
        bfs(s);
        if (level[t] < 0) {
            return flow;
        }
        memset(iter, 0, sizeof(iter));
        int f;
        while ((f=dfs(s,t,INF*INF)) > 0) {
            flow += f;
        }
    }
}
```

80 Mathematcis - Euler Phi Function

```
//Memoizing
#include <iostream>
#include <limits.h>
#include <cstdlib>
#include <cmath>
using namespace std;

const int N1 = 50001, N2 = 5133;
bool isPrime[N1];
int prime[N2], nPrime, totient[N1];
```

```
void sieveAndTotient() {
    // reset
    for (int i = 0; i < N1; ++i)
        totient[i] = i;
    isPrime[0] = isPrime[1] = false;
    for (int i = 3; i < N1; i += 2)
        isPrime[i] = true;
    for (int i = 4; i < N1; i += 2)
        isPrime[i] = false;
    nPrime = 0;
    // 2
    // update for 2
    prime[nPrime++] = 2;
    for (int j = 2; j < N1; j += 2) {
        isPrime[j] = false;
        // totient for 2
        totient[j] -= totient[j] / 2;
    }
    isPrime[2] = true;
    // odds
    for (int i = 3; i < N1; i += 2)
        if (isPrime[i]) {
            // update for i
            prime[nPrime++] = i;
            if (i < INT_MAX)
                for (int j = i; j < N1; j += i) {
                    isPrime[j] = false;
                    // totient for i
                    totient[j] -= totient[j] / i;
                }
            isPrime[i] = true;
        }
}

//Direct
int fi(int n) {
        int result = n;
        for (int i = 2; i * i <= n; i++) {
                if (n % i == 0) {
                        result -= result / i;
                }
                while (n % i == 0) {
                        n /= i;
                }
        }
        if (n > 1) {
                result -= result / n;
        }
        return result;
}
```

81 Mathematics - Highly Decomposite Number

```
bool p[MAXN];
vector<int> primes;

void build(void) {
        memset(p, true, sizeof(p));

        for (int i = 2; i <= MAXN; i++) {
                if (p[i]) {
                        primes.push_back(i);
                        for (int j = i * i; j <= MAXN; j += i) {
                                p[j] = false;
                        }
                }
        }
}

int func(Int x) {
        int ans = 1;

        for (int i = 0; i < (int) primes.size() && x > 1; i++) {
                if (x % primes[i] == 0) {
                        int curr = 0;
                        while (x % primes[i] == 0) {
                                x /= primes[i];
                                curr += 1;
                        }
                        ans *= (curr + 1);
                }
        }
        return ans;
}

set<Int> st;
```

```
void go(int id, Int v, int last) {
	Int base = primes[id];
	if (v > MAXV) return;
	st.insert(v);

	for (int i = 0; i <= last; i++) {
		v *= (Int) base;
		if (v > MAXV) break;
		go(id + 1, v, i);
	}
}
vector<Int> ans;

for (set<Int>::iterator it = st.begin(); it != st.end(); it++) {
	int s = func(*it);
		if (s > curr) {
		ans.push_back(*it);
		curr = s;
	}
}
```

82 Mathematics - Catalan

```
//Catalan numbers with DP
void getCatalan(int n){
    int catalan[n+1];
    int MOD = 100000000;
    for (int i=0; i <= n; i++){
        if (i==0 || i==1){
            catalan[i] = 1;
        }else{
            int sum =0;
            int l, r;
            for (int k=1;k<=i;k++){
                l = catalan[k-1] % MOD;
                r = catalan[i-k] % MOD;
                sum = (sum + (l * r) % MOD) % MOD;
            }
            catalan[i] = sum;
        }
    }
}
```

```
//Preprocessing Fatorial numbers and Answer in O(1)
Int catalan(int N) {
	Int ans = fat[2 * N];
	Int p = ((fat[N] * fat[N + 1]) % MOD) % MOD;
	ans *= modpow(p, MOD - 2, MOD);

	ans = ((ans % MOD) + MOD) % MOD;

	return ans;
}
```

83 Mathematics - Chinese Remainder Theorem

```
// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b. Here, z is unique modulo M = lcm(x,y).
// Return (z,M). On failure, M = -1.

pair<int, int> chinese_remainder_theorem(int x, int a, int y, int b) {
    int s, t;
    int d = extended_euclid(x, y, s, t);
    if (a%d != b%d) return make_pair(0, -1);
    return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i. Note that the solution is
// unique modulo M = lcm_i (x[i]). Return (z,M). On
// failure, M = -1. Note that we do not require the a[i]'s
// to be relatively prime.

pair<int, int> chinese_remainder_theorem(const VI &x, const VI &a) {
    pair<int, int> ret = make_pair(a[0], x[0]);
    for (int i = 1; i < x.size(); i++) {
        ret = chinese_remainder_theorem(ret.second, ret.first, x[i], a[i]);
        if (ret.second == -1) break;
    }
    return ret;
}
```

84 Mathematics - Extended GCD - Reduced

```
// returns d = gcd(a,b); finds x,y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
    int xx = y = 0;
    int yy = x = 1;
    while (b) {
        int q = a/b;
        int t = b; b = a%b; a = t;
        t = xx; xx = x-q*xx; x = t;
        t = yy; yy = y-q*yy; y = t;
    }
    return a;
}
```

85 Mathematics - Extended GCD

```
//Inverse mod using extended euclid algorith,

/* This function return the gcd of a and b followed by
    the pair x and y of equation ax + by = gcd(a,b)*/
pair<int, pair<int, int> > extendedEuclid(int a, int b) {
    int x = 1, y = 0;
    int xLast = 0, yLast = 1;
    int q, r, m, n;
    while(a != 0) {
        q = b / a;
        r = b % a;
        m = xLast - q * x;
        n = yLast - q * y;
        xLast = x, yLast = y;
        x = m, y = n;
        b = a, a = r;
    }
    return make_pair(b, make_pair(xLast, yLast));
}

int modInverse(int a, int m) {
    return (extendedEuclid(a,m).second.first + m) % m;
}
```

86 Mathematics - FasterSieve

```
//O(n)
const int N = 10000000;
int lp[N+1];
vector<int> pr;

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

87 Mathematics - Fibonnaci - Fast Doubling

```
typedef long long int lli;
typedef pair<lli, lli> ii;

ii fast_doubling(lli n, lli mod) {
        if(n == 1) return ii(1, 1);
        else if(n == 2) return ii(1, 2);

        ii aux = fast_doubling(n/2, mod);
        ii ret;
        ret.first = (aux.first*(aux.second*2 + mod - aux.first))%mod;
        ret.second = ((lli)pow(aux.first, 2)+(lli)pow(aux.second, 2))%mod;

        if(n%2 == 0) {
                return ret;
        } else {
                return ii(ret.second, (ret.first+ret.second)%mod);
        }
}
```

88 Mathematics - Fraction Library

```
struct fraction {
    int num, denom;
    fraction(int num, int denom): num(num), denom(denom){
    }
    fraction() { num = 0; denom = 0; }
    void reduce(fraction& f) {
        int l = gcd(f.num, f.denom);
        f.num = f.num/l;
        f.denom = f.denom/l;
    }
    fraction operator+(const fraction& f) {
        fraction ans;
        int l = lcm(denom, f.denom);
        ans.num = ((l / denom) * num) + ((l / f.denom) * f.num);
        ans.denom = l;
        reduce(ans);
        return ans;
    }
    fraction operator-(const fraction& f) {
        fraction ans;
        ans.num = num - f.num;
        ans.denom = denom - f.denom;
        reduce(ans);
        return ans;
    }
    fraction operator*(const fraction& f) {
        fraction ans;
        ans.num = num * f.num;
        ans.denom = denom * f.denom;
        reduce(ans);
        return ans;
    }
    fraction operator/(const fraction& f) {
        fraction ans;
        ans.num = num * f.denom;
        ans.denom = denom * f.num;
        reduce(ans);
        return ans;
    }
    bool operator!=(const fraction& f) {
        return num != f.num || denom != f.denom;
    }
    bool operator==(const fraction& f) {
        return num == f.num && denom == f.denom;
    }
    friend ostream &operator<<(ostream &out, fraction f) {
        out << f.num << "/" << f.denom << "\n";
        return out;
    }
    friend istream &operator>>(istream &in, fraction f) {
        in >> f.num >> f.denom;
        return in;
    }
};
```

89 Mathematics - GCD LCM

```
//GCD - Maximo Divisor Comum
int gcd(int a, int b) {
        if(b == 0) return a;
        return gcd(b, a % b);
}
//******************************************************
//LCM - Minimo Multiplo Comum
int lcm(int a, int b) {
        return a * b / gcd(a, b);
}
```

90 Mathematics - Gaussian Elimination

```
vector<double> gauss(vector< vector<double> > A) {
    int n = A.size();

    for (int i=0; i<n; i++) {
        // Search for maximum in this column
        double maxEl = abs(A[i][i]);
        int maxRow = i;
        for (int k=i+1; k<n; k++) {
            if (abs(A[k][i]) > maxEl) {
                maxEl = abs(A[k][i]);
                maxRow = k;
            }
        }
```

```
        // Swap maximum row with current row (column by column)
        for (int k=i; k<n+1;k++) {
            double tmp = A[maxRow][k];
            A[maxRow][k] = A[i][k];
            A[i][k] = tmp;
        }

        // Make all rows below this one 0 in current column
        for (int k=i+1; k<n; k++) {
            double c = -A[k][i]/A[i][i];
            for (int j=i; j<n+1; j++) {
                if (i==j) {
                    A[k][j] = 0;
                } else {
                    A[k][j] += c * A[i][j];
                }
            }
        }
    }

    // Solve equation Ax=b for an upper triangular matrix A
    vector<double> x(n);
    for (int i=n-1; i>=0; i--) {
        x[i] = A[i][n]/A[i][i];
        for (int k=i-1;k>=0; k--) {
            A[k][n] -= A[k][i] * x[i];
        }
    }
    return x;
}
```

91 Mathematics - Mathematical Expression Solver

```
//Solver for mathematical expressions
void doOp(stack<double> &num, stack<char> &op){
        double A = num.top(); num.pop();
        double B = num.top(); num.pop();
        char oper = op.top(); op.pop();
        double ans;
        if(oper == '+'){
                ans = A+B;
        }else if(oper == '-'){
                ans = B-A;
        }else if(oper == '*'){
                ans = A*B;
        }else{
                if(A != 0){
                        ans = B/A;
                }else{
                        //division by 0
                        ans = -1;
                }
        }
        num.push(ans);
}

double parse(string s){
    stack<char> op;
    stack<double> num;
    map<char,int> pr;

    //setting the priorities, greater values with higher pr
    pr['+'] = 0;
    pr['-'] = 0;
    pr['*'] = 1;
    pr['/'] = 1;

    for (int i = 0; i < s.size(); i++){
        if (s[i] == ')'){
            while(!op.empty() && op.top() != '('){
                doOp(num,op);
            }
            op.pop();
        } else if(s[i] == '('){
                        op.push('(');
        } else if(!(s[i] >= '0' && s[i] <= '9')){
            while(!op.empty() && pr[s[i]] <= pr[op.top()] && op.top() !=
                '('){
                doOp(num,op);
            }
            op.push(s[i]);
        } else {
            double ans = 0;
            while(i < s.size() && s[i] >= '0' && s[i] <= '9'){
                ans = ans * 10 + (s[i] - '0');
```

```
            i++;
        }
        i--;
        num.push(ans);
    }
  }
  while (op.size()) {
    doOp(num,op);
  }
  return num.top();
}
```

92 Mathematics - Matrix Multiplication

```
vector<vector<int> > multiply(vector<vector<int> > a, vector<vector<int>
    > b) {
    vector<vector<int> > res(c, vector<int>(c));
    for(int i = 0; i < c; i++) {
        for(int j = 0; j < c; j++) {
            int sum = 0;
            for (int k = 0; k < c; k++) {
                sum |= a[i][k] & b[k][j];
            }
            res[i][j] = sum;
        }
    }
    return res;
}

vector<vector<int> > binPow(vector<vector<int> > a, int n) {
    if (n == 1) {
        return a;
    } else if ((n & 1) != 0) {
        return multiply(a, binPow(a, n - 1));
    } else {
        vector<vector<int> > b = binPow(a, n / 2);
        return multiply(b, b);
    }
}
```

93 Mathematics - Miller-Rabin Primality Test

```
bool miillerTest(Int d, Int n) {
  // Pick a random number in [2..n-2]
  // Corner cases make sure that n > 4
  Int a = 2 + rand() % (n - 4);

  // Compute a^d % n
  Int x = modPow(a, d, n);

  if (x == 1 || x == n-1) {
    return true;
  }

  // Keep squaring x while one of the following doesn't
  // happen
  // (i)  d does not reach n-1
  // (ii) (x^2) % n is not 1
  // (iii) (x^2) % n is not n-1
  while (d != n-1) {
    x = (x * x) % n;
    d *= 2;

    if (x == 1) {
      return false;
    }
    if (x == n-1) {
      return true;
    }
  }

  // Return composite
  return false;
}

// It returns false if n is composite and returns true if n
// is probably prime. k is an input parameter that determines
// accuracy level. Higher value of k indicates more accuracy.
bool isPrime(Int n, int k) {
  if (n <= 1 || n == 4) return false;
  if (n <= 3) return true;

  Int d = n - 1;
  while (d % 2 == 0) {
```

```
        d /= 2;
    }

    for (int i = 0; i < k; i++) {
        if (miillerTest(d, n) == false) {
             return false;
        }
    }

    return true;
}
```

94 Mathematics - Mod Pow

```
//modpow(a, n, mod) - calcula a^n % mod de maneira eficiente
int modpow(int a, int n, int mod) {
    int res = 1;
    while (n) {
        if (n & 1) {
            res = (res * a) % mod;
        }
        a = (a * a) % mod;
        n /= 2;
    }
    return res;
}
```

95 Mathematics - Modular Inverse for Primes

```
/* This function calculates (a^b)%MOD */
int pow(int a, int b, int MOD) {
int x = 1, y = a;
    while(b > 0) {
        if(b%2 == 1) {
            x=(x*y);
            if(x>MOD) x%=MOD;
        }
        y = (y*y);
        if(y>MOD) y%=MOD;
        b /= 2;
    }
    return x;
}

int modInverse(int a, int m) {
    return pow(a,m-2,m);
}
```

96 Mathematics - Modular Linear Equation Solver

```
// finds all solutions to ax = b (mod n)
vector<int> modular_linear_equation_solver(int a, int b, int n) {
    int x, y;
    vector<int> solutions;
    int d = extended_euclid(a, n, x, y);
    if (!(b%d)) {
        x = mod (x*(b/d), n);
        for (int i = 0; i < d; i++) {
            solutions.push_back(mod(x + i*(n/d), n));
        }
    }
    return solutions;
}
```

97 Mathematics - Sieve

```
//Crivo de Erastotenes Comum - (Todos os numeros primes <= N)
bool sieve(int n) {
       bool prime[n+1];
       fill(prime, prime + n + 1,true);
       prime[0] = false;
       prime[1] = false;

       int m = (int) sqrt(n);

       for(int i = 2; i <= m; i++) {
              if(prime[i]) {
```

```
                    for (int k=i*i; k<=n; k+=i) {
                            prime[k]=false;
                    }
            }
        }
        return prime;
}
```

98 Miscellaneous - 3-Partition Array

```
Int sum[MAXN], cnt[MAXN], suffix[MAXN], v[MAXN];
Int waysToTear() {
  suffix[n] = 0;
  sum[0] = 0;
  for(int i = 0; i < n; i++){
    sum[i] = v[i];
    if(i) sum[i] += sum[i-1];
  }
  for(int i = n-1; i >= 0; i--){
        suffix[i] = v[i] + suffix[i+1];
  }
  if(sum[n-1] % 3 != 0) return 0;
  Int top = sum[n-1] / 3, ans = 0;
  for(int i = 0; i < n; i++){
        if(sum[i] == top) cnt[i] = 1;
        else cnt[i] = 0;
        if(i) cnt[i] += cnt[i-1];
  }
  for(int i = 2; i < n; i++){
        if(suffix[i] == top){
                ans += cnt[i-2];
        }
  }
  return ans;
}
```

99 Miscellaneous - Closed Interval Xor

```
//xor [a .. b]
uInt f(uInt a) {
    uInt res[] = {a,1,a+1,0};
    return res[a%4];
}

uInt getXor(uInt a, uInt b) {
        if (a == b) return a;
        uInt ans = (f(b)^f(a-1));
        return ans;
}
```

100 Miscellaneous - Days Counting

```
int meses[] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
int dp[8000][13][34];

for(int i = -3113; i <= 4000; i++){
        for(int j = 1; j <= 12; j++) {
                for(int k = 1; k <= meses[j] + (isLeap(i) && j == 2); k++){
                        dp[i + 3113][j][k] = past++;
                }
        }
 }
```

101 Miscellaneous - Fast Integer Input

```
inline void rd(int &x) {
    register int c = getchar_unlocked();
    x = 0;
    int neg = 0;

    for (; ((c<48 || c>57) && c != '-'); c = getchar_unlocked());

    if (c=='-') {
        neg = 1;
        c = getchar_unlocked();
    }

    for ( ; c>47 && c<58 ; c = getchar_unlocked()) {
```

```
        x = (x<<1) + (x<<3) + c - 48;
    }

    if (neg) {
        x = -x;
    }
}
```

102 Miscellaneous - First Highest Value to the Left

```
void fillL(void) {
    stack<int> s;

    for (int i = 0; i < N; i++) {
        if (s.empty()) {
            L[i] = i;
        } else {
            while (!s.empty() && P[s.top()] <= P[i]) {
                s.pop();
            }
            if (!s.empty()) {
                L[i] = s.top();
            } else {
                L[i] = i;
            }
        }
        s.push(i);
    }
}
```

103 Miscellaneous - Minimum Interval Coverage

```
sort(p, p + N);

int l = 0;
int r = 0;

vector<pair<int, int> > ans;

for (int i = 0; i < N; i++) {
    if (r >= M) break;
    if (p[i].first <= l) {
        int pos = i;

        r = p[i].second;

        while (i < N && p[i].first <= l) {
            if (p[i].second > r) {
                pos = i;
                r = p[i].second;
            }
            i++;
        }
        ans.push_back(p[pos]);

        l = r;
        i--;
    }
 }

if (r < M) {
    fail = true;
 }
```

104 Miscellaneous - Next Permutation in Java

```
boolean next_permutation(int[] p) {
        for (int a = p.length - 2; a >= 0; --a)
                if (p[a] < p[a + 1])
                        for (int b = p.length - 1; --b)
                                if (p[b] > p[a]) {
                                        int t = p[a];
                                        p[a] = p[b];
                                        p[b] = t;
                                        for (++a, b = p.length - 1; a < b;
                                            ++a, --b) {
                                                t = p[a];
                                                p[a] = p[b];
                                                p[b] = t;
                                        }
```

```
                return true;
            }
        return false;
}
```

105 Miscellaneous - Overflow Checker

```
int highestOneBitPosition(unsigned long long a) {
    int bits=0;
    while (a!=0) {
        ++bits;
        a>>=1;
    }
    return bits;
}

bool ms(unsigned long long a, unsigned long long b) {
    int a_bits=highestOneBitPosition(a), b_bits=highestOneBitPosition(b);
    return (a_bits+b_bits <= 64);
}
```

106 Miscellaneous - Polyomino Generator

```
int dx[4] = {0, 0, -1, 1};
int dy[4] = {-1, 1, 0, 0};
vector<pair<int, int> > q[1000010];

bool cmp(pair<int, int> a, pair<int, int> b) {
    if (abs(a.first) != abs(b.first)) {
        return abs(a.first) < abs(b.first);
    } else {
        return abs(a.second) < abs(b.second);
    }
}

uInt getHash(vector<pair<int, int> > arg) {
    uInt ans = 10000007ULL;

    for (int i = 0; i < (int) arg.size(); i++) {
        ans = ans * 1234567891 + abs(arg[i].first) + 1074178147781ULL;
        ans = ans * 1234567891 + abs(arg[i].second) + 1074178147781ULL;
    }

    return ans;
}

vector<vector<pair<int, int> > > generatePoly(int len) {
    vector<vector<pair<int, int> > > ans;

    int qf = 0, qt = 0;

    vector<pair<int, int> > base;
    base.push_back(make_pair(0, 0));
    q[qt++] = base;

    map<uInt, bool> vis;

    for ( ; qf < qt; ) {

        vector<pair<int, int> > now = q[qf++];

        if (len == (int) now.size()) {
            ans.push_back(now);
            /*
            for (int i = 0; i < (int) now.size(); i++) {
                cout << now[i].first << " " << now[i].second << " ";
            }
            cout << endl;
            */
            continue;
        }

        for (int i = 0; i < (int) now.size(); i++) {
            for (int j = 0; j < 4; j++) {
                int now_i = now[i].first + dx[j];
                int now_j = now[i].second + dy[j];

                pair<int, int> curr = make_pair(now_i, now_j);

                if (find(now.begin(), now.end(), curr) == now.end()) {
                    vector<pair<int, int> > poss = now;

                    poss.push_back(curr);
```

```
                int smx = INF;
                int smy = INF;

                for (int k = 0; k < (int) poss.size(); k++) {
                    chmin(smx, poss[k].first);
                    chmin(smy, poss[k].second);
                }

                smx = abs(smx);
                smy = abs(smy);

                for (int k = 0; k < (int) poss.size(); k++) {
                    poss[k].first += smx;
                    poss[k].second += smy;
                }

                sort(poss.begin(), poss.end());

                uInt c_hash = getHash(poss);

                if (vis[c_hash] == false) {
                    vis[c_hash] = true;
                    q[qt++] = poss;
                }
            }
        }
    }

    return ans;
}
```

107 Sorting - Heap Sort

```
int n, a[MAXN];

void downheap(int v) {
    int w = 2*v+1;
    while (w < n) {
        if(w + 1 < n) {
            if (a[w+1]>a[w]) w++;
        }
        if(a[v] >= a[w]) return;
        swap(a[v], a[w]);
        v = w;
        w = 2*v+1;
    }
}

void buildheap() {
    for (int v = n/2-1; v >= 0; v--) {
        downheap(v);
    }
}

void heapsort() {
    buildheap();
    while (n > 1) {
        n--;
        swap(a[0], a[n]);
        downheap(0);
    }
}
```

108 Sorting - Quicksort

```
//Worst Case O(n^2) but usually O(n log(n))
void quicksort(int lo, int hi) {
    int i=lo, j=hi, h;

    int x=a[(lo+hi)/2];

    do {
        while (a[i]<x) i++;
        while (a[j]>x) j--;
        if (i<=j) {
            swap(a[i], a[j]);
            i++;
            j--;
        }
    } while (i<=j);

    if (lo<j) quicksort(lo, j);
    if (i<hi) quicksort(i, hi);
```

```
}
```

109 Sortings - Merge Sort

```
//Merge-Sort O(N log N)
vector<int> merge(vector<int>& b, vector<int>& c) {
	vector<int> a;

	while(!b.empty() && !c.empty()) {
		if(*b.begin() < *c.begin()) {
			a.push_back(*b.begin());
			b.erase(b.begin());
		} else if(*b.begin() > *c.begin()) {
			a.push_back(*c.begin());
			c.erase(c.begin());
		} else {
			a.pb(*b.begin());
			a.pb(*c.begin());
			b.erase(b.begin());
			c.erase(c.begin());
		}
	}
	while(!b.empty()) { a.pb(*b.begin()); b.erase(b.begin()); }
	while(!c.empty()) { a.pb(*c.begin()); c.erase(c.begin()); }
	return a;
}

vector<int> mergeSort(vector<int>& a) {
	if(sz(a) <= 1) {
		return a;
	}
	vector<int> b;
	vector<int> c;

	for(int i = 0; i < sz(a) / 2; i++) {
		b.pb(a[i]);
	}
	for(int i = sz(a) / 2; i < sz(a); i++) {
		c.pb(a[i]);
	}
	vector<int> sb = mergeSort(b);
	vector<int> sc = mergeSort(c);
	return merge(sb, sc);
}
```

110 String - Aho Corasick

```
int T[MAX_AHO], term[MAX_AHO], sig[MAX_AHO][MAX_ALPHA], cnt;
vector <int> indice[MAX];

void add(string& arg) {
    int x = 0, n = (int) arg.size();

    for (int i = 0; i < n; i++){
        int c = (int) arg[i];
        if (sig[x][c] == 0) {
            term[cnt] = 0;
            sig[x][c] = cnt++;
        }
        x = sig[x][c];
    }
    term[x] = 1;
}


void aho() {
    queue <int> q;
    for (int i = 0; i < cc; i++){
        int v = sig[0][i];

        if (v > 0) {
            q.push(v);
            T[v] = 0;
        }
    }
    while (!q.empty()){
        int u = q.front();
        q.pop();

        for (int i = 0; i < cc; i++){
            int x = sig[u][i];

            if (x == 0) {
```

```
            continue;
        }

        int v = T[u];

        while (sig[v][i] == 0 && v != 0) {
            v = T[v];
        }

        int y = sig[v][i];
        q.push(x);

        T[x] = y;
        term[x] |= term[y];
    }
  }
}

void busca (char s[MAXT]){
    int n = strlen (s);
    int pos = 0;
    for (int i = 0; i < n; i++){
        if (sig[pos][s[i]-'A'] != 0){
            pos = sig[pos][s[i]-'A'];
            if (term[pos]){
                for (int j = 0; j < indice[pos].size(); j++)
                    printf("%d ", indice[pos][j]);
                printf("\n");
            }
        }
        else {
            if (pos != 0) i--;
            pos = T[pos];
        }
    }
}

int main (){
    char t[MAXS]; char texto[MAXT];
    int N;
    scanf("%d", &N);
    for (int i = 0; i < MAX; i++) indice[i].clear();
    cnt = 1;
    memset (sig, 0, sizeof (sig));
    for (int i = 0; i < N; i++){
        scanf("%s", t);
        add (t, i);
    }
    aho();
    scanf("%s", texto);
    busca (texto);
    return 0;
}
```

111 String - Hash

```
#include<iostream>
#include<stack>
#include<queue>
#include<cstdio>
#include<algorithm>
#include<vector>
#include<set>
#include<string>
#include<cstring>
#include<map>
#include<numeric>
#include<sstream>
#include<cmath>
using namespace std;

typedef pair<int, int> pii;
typedef long long ll;
typedef long double ld;
typedef unsigned long long Hash;
#define maxn 1000010

Hash CC;
Hash C[maxn];
Hash B;
Hash h[maxn], poww[maxn];
char s[maxn];
int n;

inline int V (char c){
    return c-'a';
}
```

```
void pre (){
    h[0] = 0ULL;
    for (int i = 1; i <= n; i++) {
                h[i] = h[i-1]*B+V(s[i-1]);
        }
    poww[0] = 1ULL;
    for (int i = 1; i <= n; i++) {
                poww[i] = poww[i-1]*B;
        }
    C[0] = CC;
    for (int i = 1; i <= n; i++) {
                C[i] = C[i-1]*CC;
        }
}

Hash calcula (int a, int b){
    return h[b]-h[a]*poww[b-a]+C[b-a];
}

int main (){
    CC = 5831ULL;
    B = 33ULL;
    scanf("%s", s);
    n = strlen(s);
    pre();

    while (1){
        int a, b; scanf("%d %d", &a, &b);
        cout << calcula (a, b) << endl;
    }

    return 0;
}
```

112 String - Knuth Morris Pratt

```
vector<int> KMP(string S, string K) {
    vector<int> T(K.size() + 1, -1);
    vector<int> matches;

    if(K.size() == 0) {
        matches.push_back(0);
        return matches;
    }
    for(int i = 1; i <= K.size(); i++) {
        int pos = T[i - 1];
        while(pos != -1 && K[pos] != K[i - 1]) pos = T[pos];
        T[i] = pos + 1;
    }

    int sp = 0;
    int kp = 0;
    while(sp < S.size()) {
        while(kp != -1 && (kp == K.size() || K[kp] != S[sp])) kp = T[kp];
        kp++;
        sp++;
        if(kp == K.size()) matches.push_back(sp - K.size());
    }

    return matches;
}
```

113 String - Manacher Algorithm

```
//Manacher Algorithm (Longest Palindromic Substring)
string preProcess(string s) {
  int n = s.length();
  if (n == 0) return "^$";
  string ret = "^";
  for (int i = 0; i < n; i++)
    ret += "#" + s.substr(i, 1);

  ret += "#$";
  return ret;
}

vector<int> manacher(string s) {
  string T = preProcess(s);
  int n = T.length();
  vector<int> P(n);

  int C = 0, R = 0;
  for (int i = 1; i < n-1; i++) {
```

```
        int i_mirror = 2*C-i;

        P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;

        while (T[i + 1 + P[i]] == T[i - 1 - P[i]]) {
               P[i]++;
        }

        if (i + P[i] > R) {
               C = i;
               R = i + P[i];
        }
  }

  int maxLen = 0;
  int centerIndex = 0;
  for (int i = 1; i < n-1; i++) {
        if (P[i] > maxLen) {
               maxLen = P[i];
               centerIndex = i;
        }
  }
  //to return actual longets substring
  //  return s.substr((centerIndex - 1 - maxLen)/2, maxLen);
  // P[i] is the length of the largest palindrome centered at i
  return P;
}
```

114 String - Minimal Lexicografical Rotation O(n)

```
string min_lex (string s){
    int n = s.size();
    s = s + s;
    int mini = 0, p = 1, l = 0;

    while(p < n && mini + l + 1 < n)
        if(s[mini + l] == s[p + l])
            l++;
        else if(s[mini + l] < s[p + l]){
            p = p + l + 1;
            l = 0;
        }
        else if(s[mini + l] > s[p + l]){
            mini = max(mini + l + 1, p);
            p = mini + 1;
            l = 0;
        }
    s = s.substr(mini, n);
    return s;
}
```

115 String - Smallest Inclusive String

```
//Menor string que contem duas strings S1 e S2 como subsequencia

char S1[MAXS], S2[MAXS];
int dp[MAXS][MAXS];

memset(dp, 0, sizeof(dp));

for (i = 1; i <= N; i++) {
    for (j = 1; j <= M; j++) {
        if (S1[i - 1] == S2[j - 1]) {
            dp[i][j] = dp[i - 1][j - 1] + 1;
        } else {
           dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
}

string track(int i, int j) {
    if (i == 0 && j == 0) {
        return "";
    } else if (i == 0 && j > 0) {
        return track(i, j - 1) + S2[j - 1];
    } else if (i > 0 && j == 0) {
        return track(i - 1, j) + S1[i - 1];
    } else {
        if (S1[i - 1] == S2[j - 1]) {
            return track(i - 1, j - 1) + S1[i - 1];
        } else {
            if (dp[i][j - 1] > dp[i - 1][j]) {
                return track(i, j - 1) + S2[j - 1];
```

```
            } else {
                return track(i - 1, j) + S1[i - 1];
            }
        }
    }
}
```

116 String - String Period

```
//Find string period
int stringPeriod(string arg) {
        int ori_len = (int) arg.size();
        arg = arg + arg;

        vector<int> prefix = KMP(arg);
        int ans = (int) arg.size();

        for (int i = 0; i < (int) prefix.size(); i++) {
                if (prefix[i] >= ori_len) {
                        ans = i - prefix[i];
                        break;
                }
        }
        return ans;
}
```

117 String - Suffix Array

```
//Suffix Array O(n log n) and LCP in O(n)
//Better Implementation

const int MAXN = 100005;

// Begins Suffix Arrays implementation
// O(n log n) - Manber and Myers algorithm

//Usage:
// Fill str with the characters of the string.
// Call SuffixSort(n), where n is the length of the string stored in str.
// That's it!

//Output:
// pos = The suffix array. Contains the n suffixes of str sorted in
    lexicographical order.
//      Each suffix is represented as a single integer (the position of
    str where it starts).
// rnk = The inverse of the suffix array. rnk[i] = the index of the
    suffix str[i..n)
//       in the pos array. (In other words, pos[i] = k <==> rnk[k] = i)
//       With this array, you can compare two suffixes in O(1): Suffix
    str[i..n) is smaller
//       than str[j..n) if and only if rnk[i] < rnk[j]

int str[MAXN]; //input
int rnk[MAXN], pos[MAXN]; //output
int cnt[MAXN], nxt[MAXN]; //internal
bool bh[MAXN], b2h[MAXN];

bool smaller_first_char(int a, int b){
    return str[a] < str[b];
}

void SuffixSort(int n){
 //sort suffixes according to their first character
    for (int i=0; i<n; ++i){
        pos[i] = i;
    }
    sort(pos, pos + n, smaller_first_char);
 //{pos contains the list of suffixes sorted by their first character}

    for (int i=0; i<n; ++i){
        bh[i] = i == 0 || str[pos[i]] != str[pos[i-1]];
        b2h[i] = false;
    }

    for (int h = 1; h < n; h <<= 1){
   //{bh[i] == false if the first h characters of pos[i-1] == the first h
        characters of pos[i]}
        int buckets = 0;
        for (int i=0, j; i < n; i = j){
            j = i + 1;
            while (j < n && !bh[j]) j++;
            nxt[i] = j;
            buckets++;
```

```
        }
        if (buckets == n) break; // We are done! Lucky bastards!
    //{suffixes are separted in buckets containing strings starting with
        the same h characters}

        for (int i = 0; i < n; i = nxt[i]){
            cnt[i] = 0;
            for (int j = i; j < nxt[i]; ++j){
                rnk[pos[j]] = i;
            }
        }

        cnt[rnk[n - h]]++;
        b2h[rnk[n - h]] = true;

        for (int i = 0; i < n; i = nxt[i]) {
            for (int j = i; j < nxt[i]; ++j) {
                int s = pos[j] - h;
                if (s >= 0){
                    int head = rnk[s];
                    rnk[s] = head + cnt[head]++;
                    b2h[rnk[s]] = true;
                }
            }
            for (int j = i; j < nxt[i]; ++j){
                int s = pos[j] - h;
                if (s >= 0 && b2h[rnk[s]]) {
                    for (int k = rnk[s]+1; !bh[k] && b2h[k]; k++) {
                        b2h[k] = false;
                    }
                }
            }
        }
        for (int i=0; i<n; ++i) {
            pos[rnk[i]] = i;
            bh[i] |= b2h[i];
        }
    }
    for (int i=0; i<n; ++i) {
        rnk[pos[i]] = i;
    }
}
// End of suffix array algorithm
```

```
// Begin of the O(n) longest common prefix algorithm
int lcp[MAXN];
// lcp[i] = length of the longest common prefix of suffix pos[i] and
    suffix pos[i-1]
// lcp[0] = 0
void getLcp(int n){
    for (int i=0; i<n; ++i) rnk[pos[i]] = i;
    lcp[0] = 0;
    for (int i=0, h=0; i<n; ++i){
        if (rnk[i] > 0){
            int j = pos[rnk[i]-1];
            while (i + h < n && j + h < n && str[i+h] == str[j+h]) h++;
            lcp[rnk[i]] = h;
            if (h > 0) h--;
        }
    }
}
// End of the longest common prefix algorithm

int N = (int) S.size();

for (int i = 0; i < N; i++) {
        str[i] = S[i];
}

SuffixSort(N);
getLcp(N);
```

118 String - Z Function

```
//Z-Function O(n) => Z[i] = biggest prefix of a substring starting from i
    which is as a prefix of s
vector<int> z_function (string s) {
        int n = (int) s.length();
        vector<int> z (n);
        for (int i=1, l=0, r=0; i<n; ++i) {
                if (i <= r) {
                        z[i] = min (r-i+1, z[i-l]);
                }
                while (i+z[i] < n && s[z[i]] == s[i+z[i]]) {
                        ++z[i];
                }
```

```
            if (i+z[i]-1 > r) {
                l = i;
                r = i+z[i]-1;
            }
        }
        return z;
}
```

119 Tree - Centroid Decomposition

```
set<int> tree[MAXN];
int height[MAXN];
int subSize[MAXN];
int treeSize;
int centroidParent[MAXN];
multiset<int> minDist[MAXN];

void dfsCounting(int node, int p) {
    subSize[node] = 1;
    treeSize += 1;

    for (auto& next : tree[node]) {
        if (next == p) continue;
        dfsCounting(next, node);

        subSize[node] += subSize[next];
    }
}

int dfsCentroid(int node, int p) {
    for (auto& next : tree[node]) {
        if (next == p) continue;

        if (2 * subSize[next] >= treeSize) {
            return dfsCentroid(next, node);
        }
    }

    return node;
}

void centroid(int node, int last_centroid) {
    treeSize = 0;
    dfsCounting(node, node);

    int curr_centroid = dfsCentroid(node, node);

    //cout << curr_centroid << " " << last_centroid << endl;
    if (node == last_centroid) {
        centroidParent[curr_centroid] = curr_centroid;
    } else {
        centroidParent[curr_centroid] = last_centroid;
    }

    for (auto& next : tree[curr_centroid]) {
        tree[next].erase(curr_centroid);
        centroid(next, curr_centroid);
    }

    tree[curr_centroid].clear();
}

void update(int node) {
    int base = node;

    while (1) {
        minDist[node].insert(getDist(node, base));
        if (node == centroidParent[node]) {
            break;
        }
        node = centroidParent[node];
    }
}

int query(int node) {
    int ans = INF;
    int curr = node;

    while (1) {
        int curr_dist = getDist(node, curr);

        if (minDist[curr].size() > 0) {
            ans = min(ans, curr_dist + *minDist[curr].begin());
        }

        if (curr == centroidParent[curr]) {
            break;
```

```
        }
        curr = centroidParent[curr];
    }

    return ans;
}
```

120 Tree - Heavy Light Decomposition - Queries on Subtrees

```
int max_color[MAXN], sz[MAXN];
ll sum, bigger, ans[MAXN];
vector<vector<int> > graph(MAXN);
int n, color[MAXN];
bool heavy[MAXN];


//get the size of each subtree
void getSz(int u, int parent){
    sz[u] = 1;
    for(int i = 0; i < (int) graph[u].size(); i++){
        int v = graph[u][i];
        if(v == parent) continue;
        getSz(v, u);
        sz[u] += sz[v];
    }
}


//just update the asnwer
void reval(int col){
    max_color[col]++;
    if(bigger < max_color[col]){
        bigger = max_color[col];
        sum = col;
    }else if(bigger == max_color[col]){
        sum += col;
    }
}


//Goes through the light childs and add all nodes to the answer
void add(int u, int parent){
    reval(color[u]);
    for(int i = 0; i < (int)graph[u].size(); i++){
        int v = graph[u][i];
        if(!heavy[v] && v != parent){
            add(v, u);
        }
    }
}


//remove from the answer all nodes in the subtree (only light nodes)
void remove(int u, int parent){
    max_color[color[u]]--;
    for(int i = 0; i < (int)graph[u].size(); i++){
        int v = graph[u][i];
        if(!heavy[v] && v != parent){
            remove(v, u);
        }
    }
}

//This dfs asnwer all querys of the type:
//How many noes in subtree of V are black.. or something like that
void dfs(int u, int parent, bool keep){
    int bigChild = -1, cnt = -1;
    for(int i = 0; i < (int)graph[u].size(); i++){
        int v = graph[u][i];
        if(sz[v] > cnt && v != parent){
            cnt = sz[v];
            bigChild = v;
        }
    }
    for(int i = 0; i < (int)graph[u].size(); i++){
        int v = graph[u][i];
        if(v != parent && v != bigChild){//goes down the light nodes
            dfs(v, u, 0);
        }
    }
    if(bigChild != -1){//goes down the heavy node only once
        dfs(bigChild, u, 1);
        heavy[bigChild] = 1;
    }
    add(u, parent);
    ans[u] = sum;
```

```
    if(bigChild != -1){
        heavy[bigChild] = 0;
    }
    if(keep == 0){
        remove(u, parent);
        bigger = 0;
        sum = 0;
    }
}

// on main:
getSz(1,-1);
dfs(1, -1, 0);
```

121 Tree - Kosaraju Algorithm

```
//ga -> Regular Adjacency List
//gb -> Transposed Adjacency List

void dfs1(int x) {
    used[x] = 1;
    for(int b = 0; b < g[x].size(); b++) {
        if(!used[g[x][b]]) dfs1(g[x][b]);
    }
        order.push_back(x);
}

void dfs2(int x) {
    used[x] = 1;
        comoponent.insert(x);
    for(int b = 0; b < gr[x].size(); b++) {
        if(!used[gr[x][b]]) dfs2(gr[x][b]);
    }
}

//Topological Sort
for (int i = 1; i <= n; i++) if(!used[i]) dfs1(i);

//Get components
for(int i = 0; i < order.size(); i++) {
    int v = order[i];
        if(!used[v]) {
                dfs2(v);
        ans++;
        component.clear();
        }
}
```

122 Tree - LCA with DP

```
vector<pair<int, Int> > tree[MAXN];
int lca[MAXN][MAX_LOG];
Int dist[MAXN];
int height[MAXN];

void dfs(int node, Int curr_cost, int p) {
    dist[node] = curr_cost;
    lca[node][0] = p;

    for (int i = 1; i < MAX_LOG; i++) {
        lca[node][i] = lca[lca[node][i - 1]][i - 1];
    }


    for (int i = 0; i < (int) tree[node].size(); i++) {
        int next = tree[node][i].first;
        Int cost = tree[node][i].second;

        if (next == p) continue;

        height[next] = height[node] + 1;

        dfs(next, curr_cost + cost, node);
    }
}

int getLca(int p, int q) {
    if (height[q] > height[p]) {
        swap(p, q);
    }

    for (int i = MAX_LOG - 1; i >= 0; i--) {
        if (height[p] - (1 << i) >= height[q]) {
            p = lca[p][i];
```

```
        }
    }

    if (p == q) return p;

    for (int i = MAX_LOG - 1; i >= 0; i--) {
        if (lca[p][i] != -1 && lca[p][i] != lca[q][i]) {
            p = lca[p][i];
            q = lca[q][i];
        }
    }
    return lca[p][0];
}

Int getCost(int p, int q) {
    return dist[p] + dist[q] - 2 * dist[getLca(p, q)];
}
```

123 Tree - LCA with Segment Tree

```
//LCA using segment tree
int H[MAXN], L[MAXN << 1], E[MAXN << 1], vis[MAXN], tree[MAXN * 8],
    path[MAXN << 1];
vector<vector<pair<int, int> > > g(MAXN);

void dfs(int x, int depth){
       vis[x] = 1;//visited
       if(H[x] == -1) H[x] = idx;//mark first time the i'th node is
           visited
       L[idx] = depth;//when you visit a node you should mark the the
           depth you have found it.
       E[idx++] = x;//the i'th recursion, global variable
       for(int i = 0; i < g[x].size(); i++){
              int next = g[x][i].first;
              if(!vis[next]){
                     path[next] = x;
                     dfs(next, depth+1);
                     L[idx] = depth;
                     E[idx++] = x;
              }
       }
}
```

```
//NlogN build the segtree and minimize the height of the I'th visited node
void build(int node, int l, int r){
       if(l > r) return;
       if(l == r){
              tree[node] = l;
       }else{
              int mid = (l+r) >> 1;
              build(node*2, l, mid);
              build(node*2+1, mid+1, r);
              int A = tree[node*2];
              int B = tree[node*2+1];
              if(L[A] <= L[B]){
                     tree[node] = A;
              }else{
                     tree[node] = B;
              }
       }
}

//Get the vertex with the minimum height, then it will be the LCA of A
    and B.
int rmq(int node, int l, int r, int ra, int rb){
       if(l > rb || r < ra){
              return -1;
       }else if(l >= ra && r <= rb){
              return tree[node];
       }else{
              int mid = (l+r) >> 1;
              int q1 = rmq(node*2, l, mid, ra, rb);
              int q2 = rmq(node*2+1, mid+1, r, ra, rb);
              if(q1 == -1){
                     return q2;
              }else if(q2 == -1){
                     return q1;
              }else{
                     if(L[q1] <= L[q2]){
                            return q1;
                     }else{
                            return q2;
                     }
              }
       }
}
```

```
idx = 0;
for(int i = 0; i <= n; i++){
       g[i].clear();
       H[i] = -1;
       L[i] = E[i] = vis[i] = 0;
       path[i] = -1;
 }
dfs(0,0);
build(1, 0, 2*n-1);
for(int i = 0; i < k; i++){
       scanf("%d%d", &u, &v);
       u--;
       v--;
       int goFrom = H[u];
       int goTo = H[v];
       if(goFrom > goTo){
              swap(goFrom, goTo);
       }
       int lcaAB = E[rmq(1, 0, 2*n-1, goFrom, goTo)]; //is the LCA of A
           and B;
 }
```

124 Tree - Lowest Common Ancestor

```
struct LCA{

    LCA(){
        build();
    }

    void build(){
        int base = 1;
        int pot = 0;
        for(int i = 0; i < 2*MAXN; i++){
            if(i >= base * 2){
                pot++;
                base *= 2;
            }
            pre[i] = pot;
            dp[i][0] = i;
        }
        base = 2;
        pot = 1;
        while(base <= 2*n){
            for(int i = 0; i + base / 2 < 2*n; i++){
                int before = base / 2;
                if(L[dp[i][pot-1]] < L[dp[i + before][pot-1]]){
                    dp[i][pot] = dp[i][pot-1];
                }else{
                    dp[i][pot] = dp[i + before][pot-1];
                }
            }
            base *= 2;
            pot++;
        }
    }

    int getLca(int u, int v){
        int l = H[u];
        int r = H[v];
        if(l > r){
            swap(l,r);
        }
        int len = r-l+1;
        if(len == 1){
            return E[dp[r][0]];
        }else{
            int base = (1 << pre[len]);
            int pot = pre[len];
            if(L[dp[l][pot]] < L[dp[r-base+1][pot]]){
                return E[dp[l][pot]];
            }else{
                return E[dp[r-base+1][pot]];
            }
        }
    }

};

void dfs(int x, int depth){
    vis[x] = 1;
    if(H[x] == -1) H[x] = idx;
    L[idx] = depth;
    E[idx++] = x;
    for(int i = 0; i < g[x].size(); i++){
        int next = g[x][i].first;
        int cost = g[x][i].second;
```

```
        if(!vis[next]){
            dfs(next, depth+1);
            L[idx] = depth;
            E[idx++] = x;
        }
    }
}
```

125 Tree - Order Statistics Tree - STL

```
//Order statistics tree inside STL
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

int main(){
        ordered_set<int> s;
        s.insert(1);
        s.insert(3);
        cout << s.order_of_key(2) << endl; // the number of elements in
            the s less than 2
        cout << *s.find_by_order(0) << endl; // print the 0-th smallest
            number in s(0-based)
}
```

126 Tree - Splay Tree

```
//Splay Tree
template< typename T, typename Comp = std::less< T > >
class splay_tree {
private:
  Comp comp;
  unsigned long p_size;
  struct node {
    node *left, *right;
    node *parent;
    T key;
    node( const T& init = T( ) ) : left( 0 ), right( 0 ), parent( 0 ),
        key( init ) { }
  } *root;

  void left_rotate( node *x ) {
    node *y = x->right;
    x->right = y->left;
    if( y->left ) y->left->parent = x;
    y->parent = x->parent;
    if( !x->parent ) root = y;
    else if( x == x->parent->left ) x->parent->left = y;
    else x->parent->right = y;
    y->left = x;
    x->parent = y;
  }

  void right_rotate( node *x ) {
    node *y = x->left;
    x->left = y->right;
    if( y->right ) y->right->parent = x;
    y->parent = x->parent;
    if( !x->parent ) root = y;
    else if( x == x->parent->left ) x->parent->left = y;
    else x->parent->right = y;
    y->right = x;
    x->parent = y;
  }

  void splay( node *x ) {
    while( x->parent ) {
      if( !x->parent->parent ) {
        if( x->parent->left == x ) right_rotate( x->parent );
        else left_rotate( x->parent );
      } else if( x->parent->left == x && x->parent->parent->left ==
          x->parent ) {
        right_rotate( x->parent->parent );
        right_rotate( x->parent );
      } else if( x->parent->right == x && x->parent->parent->right ==
          x->parent ) {
        left_rotate( x->parent->parent );
        left_rotate( x->parent );
```

```
    } else if( x->parent->left == x && x->parent->parent->right ==
        x->parent ) {
      right_rotate( x->parent );
      left_rotate( x->parent );
    } else {
      left_rotate( x->parent );
      right_rotate( x->parent );
    }
  }

  void replace( node *u, node *v ) {
    if( !u->parent ) root = v;
    else if( u == u->parent->left ) u->parent->left = v;
    else u->parent->right = v;
    if( v ) v->parent = u->parent;
  }

  node* subtree_minimum( node *u ) {
    while( u->left ) u = u->left;
    return u;
  }

  node* subtree_maximum( node *u ) {
    while( u->right ) u = u->right;
    return u;
  }
public:
  splay_tree( ) : root( 0 ), p_size( 0 ) { }

  void insert( const T &key ) {
    node *z = root;
    node *p = 0;

    while( z ) {
      p = z;
      if( comp( z->key, key ) ) z = z->right;
      else z = z->left;
    }

    z = new node( key );
    z->parent = p;

    if( !p ) root = z;
    else if( comp( p->key, z->key ) ) p->right = z;
    else p->left = z;

    splay( z );
    p_size++;
  }

  node* find( const T &key ) {
    node *z = root;
    while( z ) {
      if( comp( z->key, key ) ) z = z->right;
      else if( comp( key, z->key ) ) z = z->left;
      else return z;
    }
    return 0;
  }

  void erase( const T &key ) {
    node *z = find( key );
    if( !z ) return;

    splay( z );

    if( !z->left ) replace( z, z->right );
    else if( !z->right ) replace( z, z->left );
    else {
      node *y = subtree_minimum( z->right );
      if( y->parent != z ) {
        replace( y, y->right );
        y->right = z->right;
        y->right->parent = y;
      }
      replace( z, y );
      y->left = z->left;
      y->left->parent = y;
    }

    delete z;
    p_size--;
  }

  const T& minimum( ) { return subtree_minimum( root )->key; }
  const T& maximum( ) { return subtree_maximum( root )->key; }

  bool empty( ) const { return root == 0; }
  unsigned long size( ) const { return p_size; }
```

```
};
```

127 Tree - Tree Center

```
void addEdge(int U_, int V_){
    graph[U_].push_back(V_);
    graph[V_].push_back(U_);
    deg[U_]++;
    deg[V_]++;
}

vector<int> findCenter(){
    queue<int> q;
    //pushing the leaves
    for(int i = 0; i < n; i++){
        dist[i] = 0;
        if(deg[i] == 1){
            q.push(i);
        }
    }
    int further = 0;
    while(!q.empty()){
        int top = q.front(); q.pop();
        for(int i = 0; i < graph[top].size(); i++){
            int next = graph[top][i];
            deg[next]--;
            if(deg[next] == 1){
                q.push(next);
                dist[next] = dist[top] + 1;
                further = max(further, dist[next]);
            }
        }
    }
    vector<int> ans;
    //all reachable nodes with the maximum distance, belong to the center
    for(int i = 0; i < n; i++){
        if(dist[i] == further){
            ans.push_back(i);
        }
    }
    return ans;
}
```

128 Tree -Heavy Light Decomposition

```
vector<vector<pair<int,int> > > g(MAXN);
int cnt[MAXN], prev[MAXN], chainNode[MAXN], chainHead[MAXN],
    posInChain[MAXN], base[MAXN], level[MAXN], chainIdx, idxSegTree;
int H[MAXN], L[MAXN << 1], E[MAXN << 1], idx;

struct LCA{
    int tree[MAXN * 8];
    LCA(int root, int n){
        build(1, 0, 2*n-1);
    }

    //NlogN build the segtree and minimize the height of the I'th visited
        node
    void build(int node, int l, int r){
        if(l > r) return;
        if(l == r){
            tree[node] = l;
        }else{
            int mid = (l+r) >> 1;
            build(node*2, l, mid);
            build(node*2+1, mid+1, r);
            int A = tree[node*2];
            int B = tree[node*2+1];
            if(L[A] <= L[B]){
                tree[node] = A;
            }else{
                tree[node] = B;
            }
        }
    }

    //Get the vertex with the minimum height, then it will be the LCA of
        A and B.
    int rmq(int node, int l, int r, int ra, int rb){
        if(l > rb || r < ra){
            return -1;
        }else if(l >= ra && r <= rb){
            return tree[node];
        }else{
            int mid = (l+r) >> 1;
            int q1 = rmq(node*2, l, mid, ra, rb);
            int q2 = rmq(node*2+1, mid+1, r, ra, rb);
```

```
            if(q1 == -1){
                return q2;
            }else if(q2 == -1){
                return q1;
            }else{
                if(L[q1] <= L[q2]){
                    return q1;
                }else{
                    return q2;
                }
            }
        }
    }

    int getLCA(int u, int v, int n){
        int goFrom = H[u];
        int goTo = H[v];
        if(goFrom > goTo){
            swap(goFrom, goTo);
        }
        return E[rmq(1, 0, 2*n-1, goFrom, goTo)]; //is the LCA of A and B;
    }
};

struct SegTree{

    int tree[MAXN*4];

    SegTree(){
        memset(tree,0,sizeof(tree));
    }


    void build(int node, int l, int r){
        if(l > r) return;
        if(l == r){
            tree[node] = l;
        }else{
            int mid = (l+r) >> 1;
            build(node*2, l, mid);
            build(node*2+1, mid+1, r);
            int A = tree[node*2];
            int B = tree[node*2+1];
            tree[node] = base[A] > base[B] ? A : B;
        }
    }

    int rmq(int node, int l, int r, int ra, int rb){
        if(l > rb || r < ra){
            return -1;
        }else if(l >= ra && r <= rb){
            return tree[node];
        }else{
            int mid = (l+r) >> 1;
            int q1 = rmq(node*2, l, mid, ra, rb);
            int q2 = rmq(node*2+1, mid+1, r, ra, rb);
            if(q1 == -1){
                return q2;
            }else if(q2 == -1){
                return q1;
            }else{
                return base[q1] > base[q2] ? q1 : q2;
            }
        }
    }

    void update(int node, int l, int r, int pos, int value) {
        if (l > r) return;
        if (l == r) {
            base[pos] = value;
        } else {
            int m = (l + r) >> 1;
            if (pos <= m) {
                update(2 * node, l, m, pos, value);
            } else {
                update(2 * node + 1, m + 1, r, pos, value);
            }
            tree[node] = base[tree[2 * node]] > base[tree[2 * node + 1]] ?
                tree[2 * node] : tree[2 * node + 1];
        }
    }
};

//Decompose the tree into chains
void HLD(int node, int cost, int parent){
    if(chainHead[chainIdx] == -1){
        chainHead[chainIdx] = node;
    }
    chainNode[node] = chainIdx;
    posInChain[node] = idxSegTree;
```

```
    base[idxSegTree++] = cost;
    int nodeHeavy = -1, nextCost;
    //seeking the special child (the one with most childs on the subtrees)
    for(int i = 0; i < g[node].size(); i++){
        int next = g[node][i].first;
        if(next != parent && (nodeHeavy == -1 || cnt[next] >
            cnt[nodeHeavy])){
            nodeHeavy = next;
            nextCost = g[node][i].second;
        }
    }
    if(nodeHeavy > -1){
        //expanding the current chain
        HLD(nodeHeavy, nextCost, node);
    }

    for(int i = 0; i < g[node].size(); i++){
        int next = g[node][i].first;
        if(next != nodeHeavy && next != parent){
            chainIdx++;
            HLD(next, g[node][i].second, node);
        }
    }

}

void dfsCnt(int node, int parent, int depth = 0){
    if(H[node] == -1) H[node] = idx;//mark first time the i'th node is
        visited
    L[idx] = depth;//when you visit a node you should mark the the depth
        you have found it.
    E[idx++] = node;//the i'th recursion, global variable
    level[node] = depth;
    cnt[node] = 1;
    for(int i = 0; i < g[node].size(); i++){
        int next = g[node][i].first;
        if(next != parent){
            prev[next] = node;
            dfsCnt(next, node, depth + 1);
            cnt[node] += cnt[next];
            L[idx] = depth;
            E[idx++] = node;
        }
    }
}
```

```
int walkChain(int U, int V, SegTree &q, int n){
    if(U == V) return 0;
    int ans = 0;
    while(chainNode[U] != chainNode[V]){
        int Left = posInChain[chainHead[chainNode[U]]];
        int Right = posInChain[U];
        int val = base[q.rmq(1, 0, n-1, Left, Right)];
        if(val > ans) ans = val;
        U = prev[chainHead[chainNode[U]]];
    }
    if(U == V) return ans;
    int val = base[q.rmq(1, 0, n-1, posInChain[V]+1, posInChain[U])];
    if(val > ans) ans = val;
    return ans;
}

int getMax(int U, int V, LCA &ref, SegTree &q, int n){
    int lca = ref.getLCA(U, V, n),a=0,b=0;
    if(lca != U)
        a = walkChain(U, lca, q, n);
    if(lca != V)
        b = walkChain(V, lca, q, n);
    return max(a,b);
}

void update(int a, int b, int c, SegTree &q, int n){
    if(level[a] < level[b]){//update b
        q.update(1,0,n-1,posInChain[b], c);
    }else{//update a
        q.update(1,0,n-1,posInChain[a], c);
    }
}
void add(int a, int b, int c){
    g[a].push_back(make_pair(b,c));
    g[b].push_back(make_pair(a,c));
}

int n, t, from[MAXN], to[MAXN], cost[MAXN], A, B;
char TYPE[20];

int main(void){
    scanf("%d", &t);
    while(t--){
        scanf("%d", &n);
```

```
        chainIdx = idxSegTree = idx = 0;
        for(int i = 0; i <= n; i++){
            cnt[i] = prev[i] = chainNode[i] = base[i] = level[i] = 0;
            chainHead[i] = posInChain[i] = H[i] = -1;
            g[i].clear();
        }
        memset(L,0,sizeof(L));
        memset(E,0,sizeof(E));
        for(int i = 0; i < n - 1; i++){
            scanf("%d%d%d", &from[i], &to[i], &cost[i]);
            from[i]--;
            to[i]--;
            add(from[i], to[i], cost[i]);
        }
        dfsCnt(0,-1);
        LCA lca(0,n);
        HLD(0,-1, -1);
        SegTree query;
        query.build(1,0,n-1);
        while(1){
            scanf("%s", TYPE);
            if(TYPE[0] == 'D') break;
            scanf("%d%d", &A, &B);
            A--;
            if(TYPE[0] == 'Q'){
                B--;
                printf("%d\n", getMax(A, B, lca, query, n));
            }else if(TYPE[0] == 'C'){
                update(from[A], to[A], B, query, n);
            }
        }
    }
    return 0;
}
```

129 Tree Isomorphism

```
struct node{
    vector<int> sortedLabel;
    int label;
    int pos;
    int quem;
    node(){label = 0;}
    node( int pos_): pos(pos_){label = 0;}
    bool operator < (const node &o) const{
        return sortedLabel < o.sortedLabel;
    }
    void clear() {
        sortedLabel.clear();
        label = 0;
    }
};

vector<vector<int> > graph(MAXN);
vector<vector<node> > level(MAXN);
int n, U, V;
int deg[MAXN], dist[MAXN];
bool vis[MAXN];

void addEdge(int U_, int V_){
    graph[U_].push_back(V_);
    graph[V_].push_back(U_);
    deg[U_]++;
    deg[V_]++;
}

vector<int> findCenter(int offset){
    queue<int> q;
    //pushing the leaves
    for(int i = offset; i < n+offset; i++){
        dist[i] = 0;
        if(deg[i] == 1){
            q.push(i);
        }
    }
    int further = 0;
    while(!q.empty()){
        int top = q.front(); q.pop();
        for(int i = 0; i < graph[top].size(); i++){
            int next = graph[top][i];
            deg[next]--;
            if(deg[next] == 1){
                q.push(next);
                dist[next] = dist[top] + 1;
                further = max(further, dist[next]);
            }
        }
```

```
    }
    vector<int> ans;
    //all reachable nodes with the maximum distance, belong to the center
    for(int i = offset; i < n+offset; i++){
        if(dist[i] == further){
            ans.push_back(i);
        }
    }
    return ans;
}

int bfs(int center){
    queue<pair<int, int> > q;
    for(int i = 0; i < MAXN; i++){
        dist[i] = INF;
        vis[i] = 0;
    }
    int maxLevel = 0;
    dist[center] = 0;// or level = 0
    q.push(make_pair(center, -1));
    vis[center] = 1;
    while(!q.empty()){
        int top = q.front().first;
        int pos_parent = q.front().second;
        q.pop();
        level[dist[top]].push_back(node(pos_parent));
        for(int i = 0; i < graph[top].size(); i++){
            int next = graph[top][i];
            if(!vis[next]){
                dist[next] = dist[top] + 1;
                vis[next] = 1;
                maxLevel = max(maxLevel, dist[next]);
                q.push(make_pair(next, level[dist[top]].size() - 1));
            }
        }
    }
    return maxLevel;
}

bool rootedTreeIsomorphic(int r1, int r2){
    for(int i = 0; i < MAXN; i++) level[i].clear();
    int h1 = bfs(r1);
    int h2 = bfs(r2);
    if(h1 != h2){
        return false;
    }
    for(int i = h1-1; i >= 0; i--){
        for(int j = 0; j < level[i+1].size(); j++){
            node v = level[i+1][j];
            level[i][v.pos].sortedLabel.push_back(v.label);
        }
        for(int j = 0; j < level[i].size(); j++){
            sort(level[i][j].sortedLabel.begin(),
                level[i][j].sortedLabel.end());
        }
        sort(level[i].begin(), level[i].end());
        int cnt = 0;
        for(int j = 0; j < level[i].size(); j++){
            if(j > 0 && level[i][j].sortedLabel !=
                level[i][j-1].sortedLabel) {
                cnt++;
            }
            level[i][j].label = cnt;
        }
    }
    return level[0][0].sortedLabel == level[0][1].sortedLabel;
}

bool isIsomorphic(){
    vector<int> r2 = findCenter(n);
    vector<int> r1 = findCenter(0);
    if(r1.size() != r2.size()){
        return false;
    }else{
        if(r1.size() == 1){
            return rootedTreeIsomorphic(r1[0], r2[0]);
        }else {
            return rootedTreeIsomorphic(r1[0], r2[0]) ||
                rootedTreeIsomorphic(r1[0], r2[1]);
        }
    }
}

int main(){
    for(int i = 0; i < (n-1); i++){
        cin >> U >> V;
        U--;V--;
        addEdge(U,V);
    }
    for(int i = 0; i < (n-1); i++){
```

```
        cin >> U >> V;
        U--;V--;
        addEdge(n+U,n+V);
    }
    cout << (isIsomorphic() ? "S" : "N") << endl;
    return 0;
}
```

130 Tree-Segment Tree Conversion

```
void dfs(int node, int p) {
    ini[node] = cnt++;
    for (int i = 0; i < (int) graph[node].size(); i++) {
        int next = graph[node][i];
        if (next != p) {
            dfs(next, node);
        }
    }
}

int A, B;

for (int i = 0; i < N - 1; i++) {
    scanf("%d%d", &A, &B);
    graph[A].push_back(B);
    graph[B].push_back(A);
 }

cnt = 0;
dfs(0, -1);

for (int i = 0; i < Q; i++) {
    scanf("%d%d", &A, &B);

    if (A == 0) {
        update(1, 0, N - 1, ini[B], fin[B]);
    } else {
        printf("%d\n", query(1, 0, N - 1, ini[B], fin[B]));
    }
 }
```