PreCode:

```
#include <bits/stdc++.h>
#define in freopen("input.txt", "r", stdin);
#define out freopen("output.txt", "w", stdout);
#define clr(arr, key) memset(arr, key, sizeof arr)  #define pb push_back
#define mp(a, b) make_pair(a, b)    #define infinity (1 << 28)
#define LL long long    #define pii pair <int, int>
#define PI acos(-1)     #define gcd(a, b) __gcd(a, b)
#define CF ios_base::sync_with_stdio(0);cin.tie(0);
#define lcm(a, b) ((a)*((b)/gcd(a,b)))
#define all(v) v.begin(), v.end()
#define no_of_ones __builtin_popcount // __builtin_popcountll for LL
#define SZ(v) (int)(v.size())    #define eps 10e-7
//int col[8] = {0, 1, 1, 1, 0, -1, -1, -1};
//int row[8] = {1, 1, 0, -1, -1, -1, 0, 1};
//int col[4] = {1, 0, -1, 0};
//int row[4] = {0, 1, 0, -1};
//int months[13] = {0, ,31,28,31,30,31,30,31,31,30,31,30,31};
//int X[]={1,1,2,2,-1,-1,-2,-2};//knight move//
//int Y[]={2,-2,1,-1,2,-2,1,-1};//knight move//
using namespace std;
struct point{int x, y; point () {} point(int a, int b) {x = a, y = b;}};
template <class T> T sqr(T a){return a * a;}
template <class T> T power(T n, T p) { T res = 1; for(int i = 0; i < p; i++)
res *= n; return res;}
template <class T> double getdist(T a, T b){return sqrt((a.x - b.x) * (a.x -
b.x) + (a.y - b.y) * (a.y - b.y));}    // distance between a and b
template <class T> T extract(string s, T ret) {stringstream ss(s); ss >> ret;
return ret;}    // extract words or numbers from a line
template <class T> string tostring(T n) {stringstream ss; ss << n;
 return ss.str();}    // convert a number to string
LL bigmod(LL B,LL P,LL M){LL R=1; while(P>0)
{if(P%2==1){R=(R*B)%M;}P/=2;B=(B*B)%M;} return R;}
**********************************************************
```

// **Generates the divisors:**

```
  for(i=2;i<=maxd;i++) if(facts[i].size()=0)
      for(j=i;j<=maxd;j+=i) facts[j].pb(i);
**********************************************************
```

2-SAT: //0 based

```
VI adj[2*sz]; //2*sz for true and false argument(only adj should be cleared)
int col[2*sz],low[2*sz],tim[2*sz],timer;
int group_id[2*sz],components;//components=number of components,
group_id = which node belongs to which node
bool ans[sz]; //boolean assignment ans
stack<int>S;
void scc(int u);
int TarjanSCC(int n) //n=nodes (some change may be required here)
{
   int i;    timer=components=0;     clr(col,0);
   while(!S.empty()) S.pop();
   fr(i,0,n-1) if(col[i]==0) scc(i);
   return components;
}
//double nodes needed normally
bool TwoSAT(int n) //n=nodes (some change may be required here)
{
   TarjanSCC(n);     int i;
   for(i=0;i<n;i+=2)
   {
      if(group_id[i]==group_id[i+1]) return false;
      if(group_id[i]<group_id[i+1]) //Checking who is lower in Topological
sort
         ans[i/2]=true;
      else ans[i/2]=false;
   }
   return true;
}
void add(int ina,int inb) { adj[ina].pb(inb); }
int complement(int n) { return n^1; }
void initialize(int n)
{
   for(int i=0;i<n;i++)
      adj[i].clear();
}
int main()
{
   int n, m, i, u, v;
   while(~scanf("%d %d", &n, &m))
   {
      initialize(n<<1);
      fr(i,0,m-1)
      {
         scanf("%d %d", &u, &v);
         if(u>0) u = 2*u-2;
         else u = -2*u-1;
         if(v>0) v = 2*v-2;
         else v = -2*v-1;
         add(complement(u),v);
         add(complement(v),u);
      }
      if(TwoSAT(n<<1)) puts("YES");
      else puts("NO");
   }
   return 0;
}
```

****

Aho-Corasick:

```
struct tt{
   int par, child[26], dep;
   vector<int>str;
};
tt T[250010]; // size = total number of pattern stings * length per string
char words[502][502];   int sz1;
char str[1000010]; // main string
void init(int lim)
{
   for(int i=0;i<=lim;i++)
   {
      T[i].par=0;
      T[i].dep=0;
      memset(T[i].child, 0, sizeof T[i].child);
      T[i].str.clear();
   }
   sz1=1;
}
void build(int n)
{
   int i, j, last, len;
   char ch;
   for(i=0;i<n;i++)
   {
      last=0;
      len = strlen(words[i]);
      for(j=0;j<len;j++)
      {
         ch = words[i][j] - 'a';
         if(T[last].child[ch]==0)
            T[last].child[ch]=sz1++;
         T[T[last].child[ch]].dep = T[last].dep + 1;
         last = T[last].child[ch];
      }
      T[last].str.pb(i);
   }
   queue<int>Q;
   for(i=0;i<26;i++)
   {
      if(T[0].child[i])
      {
         Q.push(T[0].child[i]);
         T[T[0].child[i]].par = 0;
      }
   }
   int u, v, k;
   while(!Q.empty())  // implementing kmp in the trie tree with kind of bfs
   {
      u = Q.front(); Q.pop();
      for(i=0;i<26;i++)
      {
         if(T[u].child[i])
         {
            v = T[u].child[i];
            k = T[u].par;
            while(k>0 && T[k].child[i]==0)
               k = T[k].par;
            T[v].par = T[k].child[i];
            Q.push(v);
         }
      }
   }
}
int freq[250000], ans[505];
void search() // this function will take a string as main input and find the
frequency of pattern strings in this string
{
    int i, j, k, len, u, v, cur = 0;    char ch;
    len = strlen(str);
    memset(freq, 0, sizeof freq);
    for(i=0;i<len;i++)
    {
        ch = str[i] - 'a';
        if(T[cur].child[ch]==0)
        {
            k = T[cur].par;
            while(k>0 && T[k].child[ch]==0)
                k = T[k].par;
            cur = T[k].child[ch];
        }
        else
            cur = T[cur].child[ch];
        freq[cur]++; // ei node ei frequency pabe
    }
    vector<pii>store;
    for(i=0;i<sz1;i++)
        store.pb(MP(T[i].dep, i));
    sort(store.rbegin(), store.rend());
    for(i=0;i<sz1;i++)
    {
        v = store[i].second;
        freq[T[v].par]+=freq[v]; // parent gulake cummulatively frequency gula
die dea
    }
    for(i=1;i<sz1;i++)
    {
        if(SZ(T[i].str))
        {
            for(j=0;j<SZ(T[i].str);j++)
                ans[T[i].str[j]] = freq[i];
        }
    }
}
```

****

Articulatin Point & Bridge & BCC:

```
vector <int> adj[SZ];
int discover[SZ], bedge[SZ], discovery_time;
bool arti[SZ];
pair <int, int> pr, e, cur;
vector <pair <int, int> > bridges;
stack <pair <int, int> > s;
void dfs(int node, int from)
{
    arti[node] = false;
    discover[node] = bedge[node] = discovery_time++;
    int i, connected = adj[node].size(), to, child = 0;
    for(i = 0; i < connected; i++)
    {
        to = adj[node][i];
        if(to == from) continue;
        ///for bcc
        if(!discover[to])
        {
            s.push(make_pair(node, to));
            dfs(to, node);
            bedge[node] = min(bedge[node], bedge[to]);
            if(bedge[to] >= discover[node])
            {
                bcc++;    cur = make_pair(node, to);
                do {
                    e = s.top(); s.pop();
                } while(e != cur);
            }
        }
        else if(discover[node] > discover[to])
        {
            s.push(make_pair(node, to));
            bedge[node] = min(discover[to], bedge[node]);
        }///for bcc
        if(!discover[to])
        {
            dfs(to, node);
            child++;
            bedge[node] = min(bedge[node], bedge[to]);
            ///for point
            if(bedge[to] >= discover[node] && from != -1)
                arti[node] = true; ///for point
            ///for bridges
            if(bedge[to] > discover[node])
            {
                if(node < to)
                    pr = make_pair(node, to);
                else
                    pr = make_pair(to, node);
                bridges.pub(pr);
            }///for bridges
        }
        else if(discover[node] > discover[to])
            bedge[node] = min(discover[to], bedge[node]);
    }
    if(from == -1 && child >= 2)
        arti[node] = true;///for point only
}
**********************************************************
```

Bellman-Ford:

```
struct edge{
    int u, v, w;
    edge();
    edge(int a, int b, int c){ u = a; v = b; w = c;}};
vector <edge> graph;
vector <int> adj[MAX];
int dist[MAX], n, m;
///graph is a vector of edges
bool belford(void)
{
    for(i = 1; i <= n; i++) dist[i] = infinity;    dist[0] = 0;
    int i, j, u, v, w;
    for(i = 1; i < n; i++)
    {
        for(j = 0; j < m; j++)
        {
            u = graph[j].u;
            v = graph[j].v;
            w = graph[j].w;
            if(dist[u] + w < dist[v])   dist [v] = dist[u] + w;
        }
    }
    for(j = 0; j < m; j++)
    {
        u = graph[j].u;
        v = graph[j].v;
        w = graph[j].w;
        if(dist[u] + w < dist[v])  return true;
    }
    return false;
}
**********************************************************
```

BigInt:

```
string add(string a,string b); //add any two string
string multiply(string a,string b); //multiply between a and b
string multiply(string a,int k);  //multiply between a and int k
string substract(string a,string b);  // substract from a to b(a always >=b)
string divide(string a,string b); //divide return a/b
string divide(string a,int k);    //divide return a/k
string mod(string a,string b);  //Modulus of divide a%b
int    mod(string a,int k);     //Modulus of divide a%k
string cut_leading_zero(string a);  //leading zero cut 001 -> 1
int compare(string a,string b); //(1 means a>b) (-1 means a<b) (0 means a=b)
string power(string s, int a);        //Calculate powerer s^a
string GCD(string a,string b);     //Calculate GCD between a and b
string LCM(string a,string b);     //Calculate LCM between a and b
string LCM(string a,string b) {
    return divide(multiply(a,b),GCD(a,b));
}
string GCD(string a,string b) {
    return (b=="0")?a:GCD(b,mod(a,b));
}
string power(string B,int P) {
    string R="1";
    while(P>0)
    {
        if(P%2==1)
            R=multiply(R,B);
        P/=2;
        B=multiply(B,B);
    }
    return R;
}
string multiply(string a,string b) {
    int i,j,multi,carry;
    string ans,temp;
    ans="0";
    for(j = SZ(b)-1; j >= 0; j--)
    {
        temp="";
        carry=0;
        for(i = SZ(a)-1; i >= 0; i--)
        {
            multi=(a[i]-'0')*(b[j]-'0')+carry;
            temp+=(multi%10+'0');
            carry=multi/10;
        }
        if(carry)   temp+=(carry+'0');
        reverse(all(temp));
        temp+=string(SZ(b)-j-1,'0');
        ans=add(ans,temp);
    }
    ans=cut_leading_zero(ans);
    return ans;
}
string multiply(string a,int k)
{
    string ans;
    int i,sum,carry=0;

    for(i = SZ(a)-1; i >= 0; i--)
    {
        sum=(a[i]-'0')*k+carry;
        carry=sum/10;
        ans+=(sum%10)+'0';
    }
    while(carry)    {ans+=(carry%10)+'0';carry/=10;}
    reverse(all(ans));
    ans=cut_leading_zero(ans);
    return ans;
}
string add(string a,string b)
{
    int carry=0,i;
    string ans;
    if(SZ(a)>SZ(b)) b=string(SZ(a)-SZ(b),'0')+b;
    if(SZ(b)>SZ(a)) a=string(SZ(b)-SZ(a),'0')+a;
    ans.resize(SZ(a));
    for(i = SZ(a)-1; i >= 0; i--)
    {
        int sum=carry+a[i]+b[i]-96;
        ans[i]=(char)(sum%10+'0');
        carry=sum/10;
    }
    if(carry)   ans.insert(0,string(1,carry+'0'));
    ans=cut_leading_zero(ans);
    return ans;
}
string substract(string a,string b)
{
    int borrow=0,i,sub;
    string ans;
    if(SZ(b)<SZ(a)) b=string(SZ(a)-SZ(b),'0')+b;
    for(i = SZ(a)-1; i >= 0; i--)
    {
        sub=a[i]-b[i]-borrow;
        if(sub<0)
        {
            sub+=10;
            borrow=1;
        }
        else borrow=0;
        ans+=sub+'0';
    }
    reverse(all(ans));
    ans=cut_leading_zero(ans);
    return ans;
}
string divide(string a,string b)
{
    string mod,temp,ans="0";
    int i,j;
    for(i = 0; i < SZ(a); i++)
    {
        mod+=a[i];
        mod=cut_leading_zero(mod);
        for(j = 0; j < 10; j++)
        {
            temp=multiply(b,j);
            if(compare(temp,mod)==1)
                break;
        }
        temp=multiply(b,j-1);
        mod=substract(mod,temp);
        ans+=(j-1)+'0';
    }
    mod=cut_leading_zero(mod);
    ans=cut_leading_zero(ans);
    return ans;
}
string divide(string a,int k)
{
    int i,sum=0;
    string ans="0";
    for(i = 0; i < SZ(a); i++)
    {
        sum=(sum*10+(a[i]-'0'));
        ans+=(sum/k)+'0';
        sum=sum%k;
    }
    ans=cut_leading_zero(ans);
    return ans;
}
string mod(string a,string b)
{
    string mod,temp,ans="0";
    int i,j;
    for(i = 0; i < SZ(a); i++)
    {
        mod+=a[i];
        mod=cut_leading_zero(mod);
        for(j = 0; j < 10; j++)
        {
            temp=multiply(b,j);
            if(compare(temp,mod)==1)
                break;
        }
        temp=multiply(b,j-1);
        mod=substract(mod,temp);
        ans+=(j-1)+'0';
    }
    mod=cut_leading_zero(mod);
    ans=cut_leading_zero(ans);
    return mod;
}
int mod(string a,int k)
{
    int i,sum=0;

    for(i = 0; i < SZ(a); i++)
        sum=(sum*10+(a[i]-'0'))%k;
    return sum;
}

int compare(string a,string b)
{
    int i;
    a=cut_leading_zero(a);
    b=cut_leading_zero(b);
    if(SZ(a)>SZ(b)) return 1;   //bigger
    if(SZ(a)<SZ(b)) return -1;  //smaller
    for(i = 0; i < SZ(a); i++)
        if(a[i]>b[i])   return 1;   //bigger
        else if(a[i]<b[i])  return -1; //smaller
    return 0;   //equal
}
string cut_leading_zero(string a)
{
    string s;    int i;
    if(a[0]!='0')   return a;
    for(i = 0; i < SZ(a)-1; i++)  if(a[i]!='0')   break;
    for(;i < SZ(a); i++)  s+=a[i];
    return s;
}
```

****

BIT:

```
void insert(int x, int v) {
    while(x <= n)
    {
        bit[x] += v;
        x += x & -x;
    }
}
int readRes(int x) {
    int ret = 0;
    while(x > 0)
    {
        ret += bit[x];
        x -= x & -x;
    }
    return ret;
}
```

****

BPM:

```
int par[MAX];
bool col[MAX];
int MAX_BMP(int n) // finds maximum possible bipartite matching
{
    int ret = 0, i;
    clr(par, -1);
    for(i = 0; i < n; i++)
    {
        clr(col, 0);
        if(dfs(i)) ret++;
    }
    return ret;
}
int dfs(int u)
{
    if(col[u])
        return false;
    col[u] = true;
    for(int i = 0; i < SZ(adj[u]); i++)
    {
        int v = adj[u][i];
        if(par[v] == -1 || dfs(par[v]))
        {
            par[v] = u;
            return true;
        }
    }
    return false;
}
```

****

Closest Pair:

```
point arr[MAX], sortedY[MAX];
bool flag[MAX];
bool compareX(const point &a, const point &b){
    return a.x < b.x;
}
bool compareY(const point &a, const point &b){
    return a.y < b.y;
}
double closest_pair(point X[], point Y[], int n) {
    double left_call, right_call, mindist;
    if(n == 1) return infinity;
    if(n == 2)
        return getdist(X[0], X[1]);
    int n1, n2, ns, j, m = n / 2, i;
    point xL[m + 1], xR[m + 1], yL[m + 1], yR[m + 1], Xm = X[m - 1],
yS[n];
    for(i = 0; i < m; i++)
    {
        xL[i] = X[i];
        flag[X[i].i] = 0;
    }
    for(; i < n; i++)
    {
        xR[i - m] = X[i];
        flag[X[i].i] = 1;
    }
    for(i = n2 = n1 = 0; i < n; i++)
    {
        if(!flag[Y[i].i]) yL[n1++] = Y[i];
        else yR[n2++] = Y[i];
    }
    left_call = closest_pair(xL, yL, n1);
    right_call = closest_pair(xR, yR, n2);
    mindist = min(left_call, right_call);
    for(i = ns = 0; i < n; i++)
```

```
        if(sqr(Y[i].x - Xm.x) < mindist)
            yS[ns++] = Y[i];
    for(i = 0; i < ns; i++)
        for(j = i + 1; j < ns && sqr(yS[j].y - yS[i].y) < mindist; j++)
            mindist = min(mindist, getdist(yS[i], yS[j]));
    return mindist;
}
int main()
{
    int n, i;  double ans;
    while(scanf("%d", &n) == 1 && n)
    {
        ans = infinity;
        for(i = 0; i < n; i++)
        {
            scanf("%lf %lf", &arr[i].x, &arr[i].y);
            arr[i].i = i;
            sortedY[i] = arr[i];
        }
        sort(arr, arr + n, compareX);
        sort(sortedY, sortedY + n, compareY);
        ans = closest_pair(arr, sortedY, n);
        ans = sqrt(ans);
        if(ans - 10000.0 > 1e-7)
            printf("INFINITY\n");
        else
            printf("%.4lf\n", ans);
    }
    return 0;
}
```

****

Convex Hull:

```
point hull[SZ], O;
int check(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); }
bool comp_angle(const point &a, const point &b)
{
    if(check(O, a, b) > 0)
        return true;
    else if(check(O, a, b) == 0)
        return getdist(O, a) < getdist(O, b);
        return false;
}
int next_to_top(stack <int> &S) {
    int p = S.top();
    S.pop();
    int ret = S.top();
    S.push(p);
    return ret;
}
vector <point> make_hull(point convex[], int n)
{
    stack <int> S;
    vector <point> hull;
    int i, p = 0;
    for(i = 1; i < n; i++)
    {
        if(convex[p].y > convex[i].y)
            p = i;
        else if(convex[p].y == convex[i].y && convex[p].x > convex[i].x)
            p = i;
    }
    swap(convex[0], convex[p]);
    O = convex[0];
    sort(convex + 1, convex + n, comp_angle);
    S.push(0), S.push(1), S.push(2);
    for(i = 3; i < n; i++)    {
        while(S.size() > 1 && check(convex[next_to_top(S)], convex[S.top()],
convex[i]) < 0)            S.pop();
        S.push(i);
    }
    n = S.size();
    for(i = n - 1; i >= 0; i--) {
        hull.push_back(convex[i]);    S.pop();
    }
    return hull;  }
```

****

ConvexHullTrick:

```
int pointer; //Keeps track of the best line from previous query
vector<long long> M; //Holds the slopes of the lines in the envelope, aka
M[i]
```

```
vector<long long> B; //Holds the y-intercepts of the lines in the envelope,
aka C[i]
//Returns true if either line l1 or line l3 is always better than line l2
bool bad(int l1,int l2,int l3)
{/*
        intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
        intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
        set the former greater than the latter, and cross-multiply to
        eliminate division*/
        return 1.0 * (B[l3]-B[l1])*(M[l1]-M[l2])< 1.0 * (B[l2]-
B[l1])*(M[l1]-M[l3]); // must check overflow
}
//Adds a new line (with lowest slope) to the structure
void add(long long m,long long b)
{
        //First, let's add it to the end
        M.push_back(m);
        B.push_back(b);
//If the penultimate is now made irrelevant between the antepenultimate
        //and the ultimate, remove it. Repeat as many times as necessary
        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 we removed what was the best line for the previous query, then the
        //newly inserted line is now the best for that query
        if (pointer>=M.size())
                pointer=M.size()-1;
        //Any better line must be to the right, since query values are
        //non-decreasing
        while (pointer<M.size()-1&&
          M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
                pointer++;
        return M[pointer]*x+B[pointer];
}
```

```
LL a[MAX], b[MAX], dp[MAX];
int main()
{
   int n, i;
   cin >> n;
   for(i = 1; i <= n; i++)
      cin >> a[i];
   for(i = 1; i <= n; i++)
      cin >> b[i];
   add(b[1], 0);
   for(i = 2; i <= n; i++)
   {
      dp[i] = query(a[i]);
      add(b[i], dp[i]);
   }
   cout << dp[n] << endl;
   return 0;
}
************************************************************
```

Dinic’s Flow:

```
// Running time: O(|V|^2 |E|)  OUTPUT: - maximum flow value
const int INF = 2000000000;
struct Edge{
   int from, to, cap, flow, index;
   Edge(int from, int to, int cap, int flow, int index) :
      from(from), to(to), cap(cap), flow(flow), index(index) {}
};
struct Dinic{
   int N;
   vector <vector<Edge> > G;
   vector <Edge *> dad;
   vector<int> Q;
   Dinic(int N) : N(N), G(N), dad(N), Q(N) {}
   void AddEdge(int from, int to, int cap)
   {
      G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
      if (from == to) G[from].back().index++;
      G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
   }
   long long BlockingFlow(int s, int t)
   {
    fill(dad.begin(), dad.end(), (Edge *) NULL);
    dad[s] = &G[0][0] - 1;     int head = 0, tail = 0;
    Q[tail++] = s;
    while (head < tail)   {
        int x = Q[head++];
        for (int i = 0; i < G[x].size(); i++)
        {
            Edge &e = G[x][i];
            if (!dad[e.to] && e.cap - e.flow > 0)
            {
                dad[e.to] = &G[x][i];
                Q[tail++] = e.to;
            }    }     }
    if (!dad[t]) return 0;        long long totflow = 0;
    for (int i = 0; i < G[t].size(); i++)
    {
        Edge *start = &G[G[t][i].to][G[t][i].index];
        int amt = INF;
        for (Edge *e = start; amt && e != dad[s]; e = dad[e->from])
        {
            if (!e){ amt = 0; break; }
            amt = min(amt, e->cap - e->flow);
        }
        if (amt == 0) continue;
        for (Edge *e = start; amt && e != dad[s]; e = dad[e->from])
        {
            e->flow += amt;
            G[e->to][e->index].flow -= amt;
        }
        totflow += amt;
    }
    return totflow;
}
long long GetMaxFlow(int s, int t) // source, sink
{
    long long totflow = 0;
    while (long long flow = BlockingFlow(s, t))
        totflow += flow;
    return totflow;    } };
```

****

Diophantine Equation:

```
// computes x and y such that ax + by = c; on failure, x = y =-1
void linear_diophantine(int a, int b, int c, int &x, int &y) {
  int d = gcd(a,b);
  if (c%d)
    x = y = -1;
 else {
    x = c/d * mod_inverse(a/d, b/d);
    y = (c-a*x)/b;
  }  }
```

Divisor finding function:

```
int find_divisor(int n)
{
    int i,ans=1,count=1;
    while(n%2==0) {
        n/=2;
        count++;  }
    ans*=count;
    for(i=3; i*i<=n; i+=2)  {   count=1;
        while(n%i==0)
        {
            n/=i;
            count++;
        }
        ans*=count;
    }
    if(n>1)
        ans*=2;
    return ans;
}
```

****

Euler's Formula:

If G is a connected plane graph with v vertices, e edges, and f faces, then
$v - e + f = 1 +$ number of connected components.

****

E-GCD:

```
typedef pair<int, int> pii;
#define x first
#define y second
pii extendedEuclid(int a, int b)   // returns x, y | ax + by = gcd(a,b)
{
    if(b == 0) return pii(1, 0);
    else   {
        pii d = extendedEuclid(b, a % b);
        return pii(d. y, d. x - d. y * (a / b));
    } }
***********************************************************
```

Gauss Elemination:

```
void gauss( int N, long double mat[NN][NN] ) {
    int i, j, k;
    for (i = 0; i < N; i++)    {
        k = i;
        for (j = i+1; j < N; j++) if (fabs(mat[j][i]) > fabs(mat[k][i])) k = j;
        if (k != i) for (j = 0; j <= N; j++) swap(mat[k][j], mat[i][j]);
        for (j = i+1; j <= N; j++) mat[i][j] /= mat[i][i];
        mat[i][i] = 1;
        for (k = 0; k < N; k++) if( k != i )   {
                long double t = mat[k][i];
                if (t == 0.0L) continue;
                for (j = i; j <= N; j++) mat[k][j] -= t * mat[i][j];
                mat[k][i] = 0.0L;
            }    }    }
***********************************************************
```

Geometry:

```
double INF = 1e100;
double EPS = 1e-12;
struct PT {
  double x, y; PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c,   y*c  ); }
  PT operator / (double c)     const { return PT(x/c,   y/c  ); }
};
double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}
// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}
// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}
// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
    double a, double b, double c, double d){
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}
// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}
// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}
// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}
// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c,
c+RotateCW90(a-c));
}
// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  }
```

```
  return c;
}
// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}
// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}
// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}
// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}
double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}
// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}
```

****

Josephus:

```
int dp[SZ][SZ];
int find_survivour(int n, int k){
    if(n == 1)  return 1;
    if(dp[n][k])   return dp[n][k];
    return dp[n][k] = ((find_survivour(n - 1, k) + k - 1) % n) + 1;
}
```

****

KMP:

```
int match[MAX];
void compute_match_array(string pat) {
    int m = SZ(pat), len = 0, i;    match[0] = 0, i = 1;
    // calculate match[i] for i = 1 to m - 1
    while(i < m)   {
        if(pat[i] == pat[len]){  len++;   match[i] = len;   i++; }
        else   {
            if(len != 0) len = match[len - 1];
            else {
                match[i] = 0;  i++;  }
        }     }    }
```

****

Kth Best Shortest Path:

```
int m, n, deg[MM], source, sink, K, val[MM][12];
struct edge{
    int v, w;
} adj[MM][500];
struct info{    int v, w, k;
    bool operator < ( const info &b ) const {
        return w > b.w;
    }  };
priority_queue < info, vector <info> > Q;
void kthBestShortestPath()  {
    int i, j;    info u, v;
    for( i = 0; i < n; i++ ) for( j = 0; j < K; j++ ) val[i][j] = inf;
    u.v = source;    u.k = 0;    u.w = 0;    Q.push(u);
    while( !Q.empty() )    {
        u = Q.top();       Q.pop();
        for( i = 0; i < deg[u.v]; i++ )        {
            v.v = adj[u.v][i].v;
            int cost = adj[u.v][i].w + u.w;
            for( v.k = u.k; v.k < K; v.k++ )
            {
                if( cost == inf ) break;
                if( val[v.v][v.k] > cost )
                {
                    swap( cost, val[v.v][v.k] );
```

```
               v.w = val[v.v][v.k];
               Q.push(v);
               break;
            }
         }
         for( v.k++; v.k < K; v.k++ )
         {
            if( cost == inf ) break;
            if( val[v.v][v.k] > cost ) swap( cost, val[v.v][v.k] );
         }
      }
   }
}
```

****

LCA:

```
int level[MAX], pwr[MAX][log2(MAX) + 2];
vector <int> adj[MAX];
queue <int> Q;
void bfs(void)
{
   clr(level, 0);
   while(!Q.empty())
   {
      int u = Q.front();
      Q.pop();
      int elements = adj[u].size();
      for(int i = 0; i < elements; i++)
      {
         int v = adj[u][i];
         if(level[v]) continue;
         level[v] = level[u] + 1;
         pwr[v][0] = u;
         Q.push(v);
      }
   }
}
void process(int n)
{
   int h, i, lev;
   h = log2(n) + 1;
   for(lev = 1; lev <= h; lev++)
   {
      for(i = 1; i <= n; i++)
      {
         if(pwr[i][lev - 1] != -1)
            pwr[i][lev] = pwr[pwr[i][lev - 1]][lev - 1];
      }
   }
}
int query(int high, int low)
{
   if(level[low] < level[high]) swap(low, high);
   int h, i, diff;
   h = log2(level[low]) + 1;
   diff = level[low] - level[high];
   for(i = 0; i <= h; i++)
      if(diff & (1 << i))
         low = pwr[low][i];
   if(low == high) return low;
   for(i = h; i >= 0; i--)
   {
      if(pwr[low][i] != -1 && pwr[low][i] != pwr[high][i])
      {
         low = pwr[low][i];
         high = pwr[high][i];
      }
   }
   return pwr[low][0];
}
```

****

LIS:

```
int sequence[100], I[101], L[100], lislength;

int input(void)
{
   int n, i;
   scanf("%d", &n);
   for(i = 0; i < n; i++)
      scanf("%d", &sequence[i]);
   I[0] = -infinity;
   for(i = 1; i <= n; i++)
```

```
        I[i] = infinity;
    return n;
}
int lis(int n)
{
    int i, low, high, mid;
    lislength = 0;
    for(i = 0; i < n; i++)
    {
        low = 0, high = lislength;
        while(low <= high)
        {
            mid = low + high >> 1;
            if(I[mid] < sequence[i])
                low = mid + 1;
            else
                high = mid - 1;
        }
        I[low] = sequence[i];
        L[i] = low;
        if(lislength < low)
            lislength = low;
    }
    return lislength;
}
void printseq(void)
{
    int pos, i, n, j, arr[lislength], val = lislength;
    for(i = 0; i < 10; i++)
        if(L[i] == lislength)  {
            pos = i;      arr[val - 1] = sequence[pos];  val--;  break;    }
    for(i = pos; i >= 0; i--) {
        if(L[i] == val && sequence[pos] > sequence[i])
        {
            arr[val - 1] = sequence[i];
            val--;
            pos = i;
        }    }  }
```

****

Manacher’s longest palindrome:

```
string s, t;
char str[1000005];
void prepare_string()
{
    int i;
    t = "^#";
    for(i = 0; i < SZ(s); i++)
        t += s[i], t += "#";
    t += "$";
}
int manacher()
{
    prepare_string();
    int P[SZ(t)], c = 0, r = 0, i, i_mirror, n = SZ(t) - 1;
    for(i = 1; i < n; i++)
    {
        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];
        }
    }
    return *max_element(P + 1, P + n);
}
int main()
{
    int kase = 1;
    while(scanf(" %s", str) && str[0] != 'E')
    {
        s = str;
        printf("Case %d: %d\n", kase++, manacher());
    }
    return 0;
}
```

****

MatExpo:

```
struct matrix
{
    LL x[5][5];
};
matrix base, zero;
matrix matmult(matrix &a, matrix &b, int n)//m*n and n*r matrix  //1 based
{
    matrix ret;   int i,j,k;
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)   {    ret.x[i][j]=0;
            for(k = 1; k <= n; k++)
                ret.x[i][j] = (ret.x[i][j] + (a.x[i][k] * b.x[k][j]) % mod) % mod;
            ret.x[i][j]%=mod;        }
    return ret;   }
matrix mat_expo(matrix b, long long p, int n) //have to pass dimension - n
{
    if(!p) return b;
    matrix xx = zero;    int i;
    for(i = 1; i <= n; i++)  xx.x[i][i] = 1;
    matrix power = b;
    while(p)    {
        if((p & 1) == 1) xx = matmult(xx, power, n);
        power = matmult(power, power, n);
        p /= 2;    }
    return xx;  }
```

****

Modular Inverse:

```
int modularInverse(int a, int n) {
pii ret = extendedEuclid(a, n);
return ((ret. x % n) + n) % n;
}
```

****

Modular Linear Equation Solver:

```
//Input – a, b, n; Output – all x in a vector; ax = b (mod n)
//EGCD returns x, y, d; ax + by = d, d = gcd(a,b);
vector <int> modularEqnSolver( int a, int b, int n )
{
    Euclid t = egcd( a, n );
    vector <int> r;
    if( b % t.d ) return r;
    int x = ( b / t.d * t.x ) % n;
    if( x < 0 ) x += n;
    for( int i = 0; i < t.d; i++ ) r.push_back( ( x + i * n / t.d ) % n );
    return r;
}
```

****

Histogram:

```
int hist[MAX];     stack <int> st;
int get_max_rec(int n) {
    int i = 0, res = 0, tem, top;
    while(i < n)    {
        if(st.empty() || hist[st.top()] <= hist[i])      st.push(i++);
        else        {
            top = st.top();
            st.pop();
            tem = hist[top] * (st.empty() ? i : i - st.top() - 1);
            res = max(tem, res);
        }    }
    while(!st.empty()) {
        top = st.top();   st.pop();
        tem = hist[top] * (st.empty()? i : i - st.top() - 1);
        res = max(tem, res);
    }
    return res;
}
```

****

Max sum 2D:

```
///need to test
int arr[SZ][SZ];
int max_sum(int n, int r)
{
    int i, j, m = 0, sum = 0;
    for(i = 1; i <= n; i++)
        {
            for(j = 1; j <= r; j++)
                    arr[i][j] += arr[i][j - 1];
        }
    for(int c1 = 1; c1 <= r; c1++)
        {
            for(int c2 = c1; c2 <= r; c2++)
                {
                    sum = 0;
                    for(int r = 1; r <= n; r++)
```

```
                        {
                            sum += (arr[r][c2] - arr[r][c1 - 1]);
                            if(sum < 0)
                                sum = 0;
                            else if(sum > m)
                                m = sum;  }    }    }
    return m;  }
```

****

MaxFlow:

```
int source, sink, from[MAX], visited[MAX], capacity[MAX][MAX];
vector <int> adj[MAX];

int find_path()
{
    //find augmenting path
    queue <int> Q;
    Q.push(source);
    clr(visited, 0);
    clr(from, -1);
    visited[source] = 1;
    int v, lim, i, cur;
    while(!Q.empty())
    {
        cur = Q.front();
        Q.pop();
        lim = SZ(adj[cur]);
        for(i = 0; i < lim; i++)
        {
            v = adj[cur][i];
            if(visited[v] || v == from[cur] || !capacity[cur][v])
                continue;
            Q.push(v);
            visited[v] = 1;
            from[v] = cur;
            if(v == sink)
                break;
        }
        if(i < lim)
            break;
    }
    //compute path capacity
    int path_capacity = infinity, prev;
    cur = sink;
    while(from[cur] > -1)
    {
        prev = from[cur];
        path_capacity = min(path_capacity, capacity[prev][cur]);
        cur = prev;
    }
    //update residual graph
    cur = sink;
    while(from[cur] > -1)
    {
        prev = from[cur];
        capacity[prev][cur] -= path_capacity;
        capacity[cur][prev] += path_capacity;
        cur = prev;
    }
    if(path_capacity == infinity)
        return 0;
    return path_capacity;
}

int max_flow()
{
    int result = 0, path_capacity;
    while(true)
    {
        path_capacity = find_path(); //finds augmenting path
        if(path_capacity == 0) //no augmenting path found
            break;
        else
            result += path_capacity;
    }
    return result;
}

void print_min_cut()
{
    vector <int> first_set;
    queue <int> Q;
    Q.push(source);
    clr(visited, 0);
    visited[source] = 1;
    int v, lim, i, cur, j;
    while(!Q.empty())
    {
        cur = Q.front();
        Q.pop();
        lim = SZ(adj[cur]);
        for(i = 0; i < lim; i++)
        {
            v = adj[cur][i];
            if(visited[v])
                continue;
            if(!capacity[cur][v])
            {
                if(visited[cur] == 1)
                {
                    visited[cur]++;
                    first_set.pb(cur);
                }
                continue;
            }
            Q.push(v);
            visited[v] = 1;
        }
    }
    lim = SZ(first_set);
    for(i = 0; i < lim; i++)
    {
        cur = first_set[i];
        for(j = 0; j < SZ(adj[cur]); j++)
        {
            v = adj[cur][j];
            if(!visited[v] && !capacity[cur][v])
                printf("%d %d\n", cur, v);
        }
    }
    printf("\n");
}
```

****

Max sum 1D:

```
struct info  {
    int start, en, sum;
};
info max_sum(int *data, int n)  {
    int start = 0, en = 0, tem = 0, i, sum = 0;    info ret;
    ret.start = ret.en = ret.sum = 0;
    for(i = 0; i < n; i++)  {
        sum += data[i];
        if(sum < 0)   {
            tem = i + 1;
            sum = 0;   }
        else if(sum > ret.sum)   {
            ret.sum = sum;
            ret.start = tem;
            ret.en = i;
        }    }
    return ret;  }
```

****

MinCostFlow:

```
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;
const L INF = (1LL << 60);
struct MinCostMaxFlow  {
    int N;
    VVL cap, flow, cost;
    VI found;
    VL dist, pi, width;
    VPII dad;
    MinCostMaxFlow(int N) :
        N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
        found(N), dist(N), pi(N), width(N), dad(N) {}

    void AddEdge(int from, int to, L cap, L cost)
    {
        this->cap[from][to] = cap;
    this->cost[from][to] = cost;
}
void Relax(int s, int k, L cap, L cost, int dir)
{
    L val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k])
    {
        dist[k] = val;
        dad[k] = make_pair(s, dir);
        width[k] = min(cap, width[s]);
    }
}
L Dijkstra(int s, int t)
{
    fill(found.begin(), found.end(), false);  fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;
    while (s != -1)        {
        int best = -1;
        found[s] = true;
        for (int k = 0; k < N; k++)
        {
            if (found[k]) continue;
            Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
            Relax(s, k, flow[k][s], -cost[k][s], -1);
            if (best == -1 || dist[k] < dist[best]) best = k;
        }
        s = best;
    }

    for (int k = 0; k < N; k++)
        pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
}
pair<L, L> GetMaxFlow(int s, int t)
{
    L totflow = 0, totcost = 0;
    while (L amt = Dijkstra(s, t))
    {
        totflow += amt;
            for (int x = t; x != s; x = dad[x].first)
            {
                if (dad[x].second == 1)
                {
                    flow[dad[x].first][x] += amt;
                    totcost += amt * cost[dad[x].first][x];
                }
                else
                {
                    flow[x][dad[x].first] -= amt;
                    totcost -= amt * cost[x][dad[x].first];
                }
            }
        }
        return make_pair(totflow, totcost);
    }
};
```

****

More BitMask:

```
int more_bit[10];
int get_bit(int mask , int pos) { return (mask / more_bit[pos]) % 3; }
int set_bit(int mask, int pos , int bit) {
    int tmp = (mask / more_bit[pos]) % 3;
    mask -= tmp * more_bit[pos];  mask += bit * more_bit[pos];
    return mask;  }
void init(void)  {    more_bit[0] = 3;
    for(int i = 1; i < 10; i++) more_bit[i] = 3 * more_bit[i - 1]; }
**mask is full when mask == more_bit[pos + 1] – 3
```

****

nCr (loop):

```
void calculate(){
    int i, j;
    for(i=0; i<=1000; i++){
        for(j=0; j<=i; j++){
            if(j == 0) nCr[i][j] = 1;
            else if(j == 1) nCr[i][j] = i;
            else nCr[i][j] = (nCr[i-1][j] + nCr[i-1][j-1]) % mod;
        }
    }
}
```

****

Phi function:

```
int phi (int n)
{
    int ret = n;
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
        {
            while (n % i == 0)
            {
                n /= i;
            }
            ret -= ret / i;
        }
    }
// this case will happen if n is a prime number
// in that case we won't find any prime that divides n
// that's less or equal to sqrt(n)
    if (n > 1) ret -= ret / n;
    return ret;
}
```

****

SCC:

```
int low[MAX], tim[MAX], col[MAX], no_of_component, n, timer,
group_id[MAX];
vector <int> adj[MAX], dag[MAX];
stack <int> st;
void scc(int u)
{
    low[u] = tim[u] = timer++;
    col[u] = 1;
    st.push(u);
    int i, elements = adj[u].size(), v, tem;
    for(i = 0; i < elements; i++)
    {
        v = adj[u][i];
        if(col[v] == 1)
            low[u] = min(low[u], tim[v]);
        else if(col[v] == 0)
        {
            scc(v);
            low[u] = min(low[u], low[v]);
        }
    }
    if(low[u] == tim[u])
    {
        do
        {
            tem = st.top();
            st.pop();
            group_id[tem]=no_of_component;
            col[tem] = 2;
        }
        while(tem != u);
        no_of_component++;
    }
}
void call_for_scc_check()
{
    no_of_component = timer = 0;
    clr(col, 0);
    int i;
    while(!st.empty()) st.pop();
    for(i = 0; i < n; i++)
    {
        if(col[i] == 0)
            scc(i);
    }
}

void make_new_DAG()
{
    int i,j,u,v;

    for(i = 0; i < no_of_component; i++) dag[i].clear();

    for(i = 0; i < n; i++)
    {
        for(j = 0; j < SZ(adj[i]); j++)
        {
            u=group_id[i];
            v=group_id[adj[i][j]];
            if(u!=v)
                dag[u].pb(v);
        }
    }
}
************************************************************
```

SegmentTree (Lazy):

```
LL segtree[SZ * 4], beg, en, sum;
LL lazy[SZ * 4];
void lazy_update(int lef, int rig, int cur, int val)
{
    if(lazy[cur])
    {
        segtree[cur] += (rig - lef + 1) * lazy[cur];
        if(lef != rig)
        {
            lazy[cur << 1] += lazy[cur];
            lazy[(cur << 1) + 1] += lazy[cur];
        }
        lazy[cur] = 0;
    }
    if(lef > en || rig < beg)
        return;
    if(lef >= beg && rig <= en)
    {
        segtree[cur] += (rig - lef + 1) * val;
        if(rig != lef)
        {
            lazy[cur << 1] += val;
            lazy[(cur << 1) + 1] += val;
        }
        return;
    }
    lazy_update(lef, (lef + rig) >> 1, cur << 1, val);
    lazy_update(((lef + rig) >> 1) + 1, rig, (cur << 1) + 1, val);
    segtree[cur] = segtree[cur << 1] + segtree[(cur << 1) + 1];
}
LL query(int lef, int rig, int cur)
{
    if(lazy[cur])
    {
        segtree[cur] += (rig - lef + 1) * lazy[cur];
        if(lef != rig)
        {
            lazy[cur << 1] += lazy[cur];
            lazy[(cur << 1) + 1] += lazy[cur];
        }
        lazy[cur] = 0;
    }
    if(lef > en || rig < beg)
        return 0;
    if(lef >= beg && rig <= en)
        return segtree[cur];
    return query(lef, (lef + rig) >> 1, cur << 1) + query(((lef + rig) >> 1) + 1,
rig, (cur << 1) + 1);
}
************************************************************
```

Sieve (bitmask):

```
LL col[s/64+10], ma;
int seive()//1 indexed
{
    long long i,j,k;
    k=0;
    LL prev=0;
    for(i=3;i<s;i+=2)
         if(!(col[i/64]&(1LL<<(i%64))))
         {
               if((i%4)==1)
               {
                    k++;
                    ma=max(ma,i-prev);
                    prev=i;
                }
                for(j=i*i;j<s;j+=2*i)
                    col[j/64]|=(1LL<<(j%64));
         }
    return k;
}
************************************************************
```

Sieve version of finding divisor:

```
int nod[100000+5];
```

```
void Generate()
{
    nod[1]=1;
    for(int i=2; i<=100000; i++)
    {
        if(!nod[i])   //here checking i is prime or not ???
        {
            nod[i]=2;
            for(int j=i+i; j<=100000; j+=i)
            {
                if(!nod[j])nod[j]=1;
                int n=j,cnt=0;
                while(!(n%i))
                {
                    cnt++;
                    n/=i;
                }
                nod[j]*=(cnt+1);
            }
        }
    }
}
```

****

Sieve version of phi:

```
int phi[10000];
const int M=1000;
void Generate_phi()
{
    int i,j;
    phi[1]=1;
    for(i=2; i<M; i++)
    {
        if(!phi[i])
        {
            phi[i]=i-1;
            for(j=i+i; j<M; j+=I)
            {
                if(!phi[j])phi[j]=j;
                phi[j]=phi[j]/i*(i-1);
            }
```

```
        }
    }
}
```

****

Sliding Window(min):

```
void sliding_window_minimum(std::vector<int> & ARR, int K) {
  // pair<int, int> represents the pair (ARR[i], i)
  std::deque< std::pair<int, int> > window;
  for (int i = 0; i < ARR.size(); i++) {
     while (!window.empty() && window.back().first >= ARR[i])
       window.pop_back();
     window.push_back(std::make_pair(ARR[i], i));

     while(window.front().second <= i - K)
       window.pop_front();

     std::cout << (window.front().first) << ' ';
  }
}
```

****

Stable Marriage:

```
int m, n, L[MAXM][MAXW], R[MAXW][MAXM], L2R[MAXM],
R2L[MAXW], p[MAXM];
void stableMarriage()
{
    memset( R2L, -1, sizeof( R2L ) );
    memset( p, 0, sizeof( p ) );
    for( int i = 0; i < m; i++ )   // Each man proposes...
    {
        int man = i;
        while( man >= 0 )
        {
            int wom;
            while( 1 )
            {
                wom = L[man][p[man]++];
                if( R2L[wom] < 0 ‖ R[wom][man]
                        > R[wom][R2L[wom]] ) break;
            }
            int hubby = R2L[wom];
            R2L[L2R[man] = wom] = man;
            man = hubby;
        }
    }
}
```

****

Suffix Array:

```
string text;
int revSA[MAX],SA[MAX];
int cnt[MAX] , nxt[MAX];
bool bh[MAX],b2h[MAX];
int lcp[MAX];
bool cmp(int i,int j)
{
    return text[i]<text[j];
}
void sortFirstChar(int n)
{
    /// sort for the first char  ...
    for(int i =0 ; i<n ; i++)
        SA[i] = i;
    sort(SA,SA+n ,cmp);

    ///indentify the bucket ........
    for(int i=0 ; i<n ; i++)
    {
        bh[i] = (i==0  || text[SA[i]]!=text[SA[i-1]]);
        b2h[i] = false;
    }
}
int CountBucket(int n)
{
    int bucket = 0;
    for(int i =0 ,j; i<n ; i=j)
    {
        j = i+1;
        while(j<n && bh[j]==false) j++;
        nxt[i] = j;
        bucket++;
    }
    return bucket;
}
void SetRank(int n)
{
    for(int i = 0 ; i<n ; i=nxt[i])
    {
        cnt[i] = 0;
        for(int j =i ; j<nxt[i] ; j++)
        {
            revSA[SA[j]] = i;
        }
    }
}
void findNewRank(int l,int r,int step)
{
    for(int j = l  ; j<r ; j++)
    {
        int pre = SA[j] - step;
        if(pre>=0)
        {
            int head = revSA[pre];
            revSA[pre] = head+cnt[head]++;
            b2h[revSA[pre]] = true;
        }
    }
}
void findNewBucket(int l,int r,int step)
{
    for(int j = l  ; j<r ; j++)
    {
        int pre = SA[j] - step;
        if(pre>=0 && b2h[revSA[pre]])
        {
            for(int k = revSA[pre]+1 ; b2h[k] && !bh[k] ; k++) b2h[k] = false;
        }
    }
}
void buildSA(int n)
{
    ///start sorting in logn step ...
    sortFirstChar(n);
    for(int h =1 ; h<n ; h<<=1)
    {
        if(CountBucket(n)==n) break;
        SetRank(n);
        /// cause n-h suffix must be sorted
        b2h[revSA[n-h]] = true;
        cnt[revSA[n-h]]++;

        for(int i = 0 ; i<n ; i=nxt[i])
        {
            findNewRank(i,nxt[i] , h);
            findNewBucket(i , nxt[i] , h);
        }
        ///set the new sorted suffix array ...
        for(int i =0 ; i<n ; i++)
        {
            SA[revSA[i]] = i;
            bh[i] |= b2h[i]; ///new bucket ....
        }
    }
}
void buildLCP(int n)
{
    int len = 0;
    for(int i = 0 ;i<n ; i++)
        revSA[SA[i]] = i;
    for(int i =0 ; i< n ; i++)
    {
        int k = revSA[i];
        if(k==0)
        {
            lcp[k] = 0;
            continue;
        }
        int j = SA[k-1];
        while(text[i+len]==text[j+len]) len++;
        lcp[k] = len;
        if(len) len--;
    }
}
void printSA()
{
    for(int i=0;i<SZ(text);i++) printf("%d %d %d %s %d\n", i, SA[i],
revSA[SA[i]], text.substr(SA[i]).c_str(), lcp[i]);
    puts("");
//    for(int i=1;i<SZ(text);i++) printf("%d ",lcp[i]);
    puts("");
}
```

****

Trie:

```
int trie[MAX][52], cnt[MAX], last;
char str[10001];
void add(char *str)
{
    int i, id, cur = 0;
    for(i = 0; str[i]; i++)
    {
        if(islower(str[i]))
            id = str[i] - 'a' + 26;
        else
            id = str[i] - 'A';
        if(trie[cur][id] == -1)
        {
            trie[cur][id] = ++last;
            clr(trie[last], -1);
            cnt[last] = 0;
        }
        cur = trie[cur][id];
    }
    cnt[cur]++;
}
/// do clr(trie[0], -1) and last = 0 for every case

int get(char *str)
{
    int id, i, cur = 0;
    for(i = 0; str[i]; i++)
    {
        if(islower(str[i]))
            id = str[i] - 'a' + 26;
        else
            id = str[i] - 'A';
        if(trie[cur][id] == -1)
```

```
         return 0;
      cur = trie[cur][id];
   }
   return cnt[cur];
}
*********************************************************
*********************************************************
```

Formula for Arithmetic Series:

→n-th term, $x_n = a+(n-1)d$

→Summation of first n term, $S_n = n\{2a + (n-1)d\}/2$

→Summation of first n odd terms = n^2

→$1+2+3+\ldots+n = n(n+1)/2$

→$1^2+2^2+3^2+\ldots+n^2 = n(n+1)(2n+1)/6$

→$1^3+2^3+3^3+\ldots+n^3 = \{n(n+1)/2\}^2$

Formula for Geometric Series:

→n-th term, $x_n = ar^{(n-1)}$

→Summation of first n term, $S_n = a/(1-r)$, (while n tens to inf)/ $S_n = a(1-r^n)/(1-r)$, (while(r<1)/ $S_n = a(r^n-1)/(r-1)$, (while(r>1)

Formula for Permutation and Combination:

→${}^nC_r = n! / r!(n-r)!$

→${}^nP_r = n! / (n-r)!$

→${}^nP_r = n!\ {}^nC_r$

→ ${}^nC_r + {}^nC_{r-1} = {}^{n+1}C_r$

Formula for Cubic Geometry:

→Area of regular polygon = (1/2) n sin(360°/n) S^2 when n = # of sides and S = length from center to a corner

→angle = $(n-2)*180^o$

Formula for Triangle:

→Area of an equilateral triangle = $(c/2)*\sqrt{(a^2 - (c/2)^2)}$.

→Area of an isosceles triangle = $(a^2 *\sqrt{3})/4$,

→Law of cosine: $c^2=a^2+b^2 - 2ab \cos C$.

→Sin (A/2) = $\sqrt{((s-b)*(s-c)/bc)}$, Cos (A/2) = $\sqrt{(s*(s-a)/bc)}$

length of median to side c = sqrt(2*(a*a+b*b)-c*c)/2

length of bisector of angle C = sqrt(ab[(a+b)*(a+b)-c*c])/(a+b)

Formula of Straight Line:

→Slope from point (x_1, y_1) to point $(x_2, y_2) = (y_2 - y_1)/(x_2 - x_1)$

→Equation for a straight line going through point (x_1, y_1) to point (x_2, y_2): $(y - y_1) = m(x - x_1)$

→Absolute distance from point (x_1, y_1) to straight line $ax + by + c = 0$: $|((ax_1 + by_1 + c)/\sqrt{(a^2+b^2)})|$

→Angles between two straight lines $y = m_1x + c_1$ and $y = m_2x + c_2$ is $\tan^{-1}((m_1-m_2)/(1+m_1m_2))$

Formula of Circle:

→ Equation of a circle centering (h,k) with radius r: $(x-h)^2+(y-k)^2=r^2$

→ Equation of a circle having diameter from point (x_1, y_1) to point (x_2, y_2): $(x-x_1)(x-x_2) + (y-y_1)(y-y_2) = 0$

→Equation of a circle going through the point where the straight line $lx+my+n=0$ and the circle $x^2+y^2+2gx+2fy+c=0$ cross each-other: $x^2+y^2+2gx+2fy+c +k(lx+my+n)=0$

→Equation of a circle going through the point where the circle $x^2+y^2+2g_1x+2f_1y+c=0$ cross each-other: and the circle $x^2+y^2+2g_2x+2f_2y+c=0$ cross each-other: $x^2+y^2+2g_1x+2f_1y+c + k(x^2+y^2+2g_2x+2f_2y+c)=0$

→ Equation of a circle centering (-g,-f) with radius $\sqrt{(g^2+f^2-c)}$: $x^2+y^2+2gx+2fy+c=0$

Formula of Probability:

→P(an event) = # of probable event in favor / total # of probable event

→P(A U B) = P(A) + P(B) – P(A ∩ B)

→ P(A ∩ B) = P(A) * P(B) for independent event, → P(A ∩ B) = P(A) * P(B/A) for dependent event

To use bits as flag:

```
int arr[MAX/32 + 5];
#define get(x) ((arr[(x)>>5]>>((x)&31))&1)
#define set(x) arr[(x)>>5]|=(1<<((x)&31));
```

have to print the digit(s) of factorial n in the given base:

```
for(n = 1; n < MAX; n++)
   arr[n] = log10(n) + arr[n - 1];
scanf("%d %d", &n, &b);
cout << (int) (arr[n] / log10(b) + 1) << endl;
```

Dijkstra:

```
struct pq{ int cost,node;
   bool operator<(const pq &b)const
   {
      return cost>b.cost; // Min Priority Queue(b is curret)
   }};
```