SUFFIX ARRAY

```
char s[MAX];
int SA[MAX],wa[MAX], wb[MAX], we[MAX], wv[MAX];

void Sufix_Array(char *cad,int *SA,int N){
  N++;
  int i, j, p, *x = wa, *y = wb, range = 256;
  memset(we, 0, range * sizeof(int));
  for (i = 0; i < N; i++)     we[ x[i] = cad[i] ]++;
  for (i = 1; i < range; i++)  we[i] += we[ i-1 ];
  for (i = N - 1; i >= 0; i--)   SA[ --we[ x[i] ] ] = i;
  for (j = p = 1; p < N; j <<= 1, range = p){
    for (p = 0, i = N - j; i < N; y[p++] = i , i++) ;
    for (i = 0; i < N; i++)
      if (SA[i] >= j) y[p++] = SA[i] - j;
    for (i = 0; i < N; i++)    wv[i] = x[ y[i] ];
    memset(we, 0, range * sizeof(int));
    for (i = 0; i< N; i++)     we[ wv[i] ]++;
    for (i = 1; i < range; i++) we[i] += we[i-1];
    for (i = N-1; i >= 0; i--)  SA[--we[wv[i]]] = y[i];
    swap(x, y);  x[SA[0]] = 0;
    for (p = i = 1; i < N; i++)
      if(y[SA[i]] == y[SA[i-1]] && y[SA[i]+j] == y[SA[i-1]+j])
         x[SA[i]] = p - 1; else x[SA[i]] = p++;
  }
  N--;
}

int rank[MAX], LCP [MAX];

void findLCP(char *cad, int *SA, int N){
  int i, j, k;
  for (i = 1; i <= N; i++)
    rank[ SA[i] ] = i;
  for (k = i = 0; i < N; LCP [rank[i++]] = k)
    for (k ? k-- : 0,j =SA[rank[i]-1]; cad[i +k] ==cad[j +k];
k++);
}
```

MANACHER

```
void manacher(char *s, int *r){
  int i,j,k=0;
  for(i = 0, j = 0; i < 2*n; i+=k, j=max(j-k,0)){
    while(i-j>=0 && i+j+1<2*n && s[(i-j)/2]==s[(i+j+1)/2]) ++j;
    r[i]=j;
    for(k=1; i>=k && r[i]>=k && r[i-k]!=r[i]-k; ++k)
     r[i+k] = min(r[i-k],r[i]-k);
  }
}

bool isPalindrome(int a, int b){
  int med=(a+b)/2;
  if((b-a)%2==0) return rad[med*2]>=(b-a+1);
  return rad[med*2+1]>=(b-a+1);
}
```

KMP

```
void preKMP(int *F) {
  int i, j;
  for (i = 1; i < M; ++i) {
    j = F[i - 1];
    while (j > -1 && B[j + 1] != B[i]) j = F[j];
    if (B[j + 1] == B[i]) F[i] = j + 1;
  }
}

void KMP() {
  int i, j = -1;
  int F[MAX]; //M-size of pattern N-size of text
  fill(F, F + M, -1);
  preKMP(F);
  for (i = 0; i < N; ++i) {
    while (j > -1 && B[j + 1] != A[i]) j = F[j];
    if (B[j + 1] == A[i]) ++j;
    if (j == (M-1)) pos[can++] = i - M + 1;
  }
}
```

TRIE

```
int tree[MAXNODE][27];
int main(){
  for(j = 'A'; j <= 'Z'; ++j) tree[0][j-'A']=-1;
  nodos=0;
  for(i = 0; i < N; ++i){
   scanf("%s", cad); p = 0;
   for(j = 0; cad[j]; ++j){
    if(tree[p][cad[j]-'A']==-1){
      tree[p][cad[j]-'A']=++nodos;
      for(k = 'A'; k <= 'Z'; ++k)
       tree[nodos][k-'A']=-1;
    }
    p = tree[p][cad[j]-'A'];
   }
  }
}
```

Z - ALGORITHM

```
#define mxn 1000100
int Z[mxn]; char S[mxn]; int n;
void zAlg(){
  int i,g,f;
  Z[g = 0] = n;
  for(i = 1; i < n; ++i) {
    if(i < g && Z[i-f] != (g-i)) Z[i] = min(Z[i-f],g-i);
    else {
      g = max(g,f=i);
      while(g < n && S[g] == S[g-f]) ++g;
      Z[i] = g - f;
    }
  }
}
```

RABIN KARP

1. Convertimos el patrón (s) en un numero H en base B (B>=alfabeto) $H = \sum_{i=0}^{m-1} s[i] * B^{m-1-i}$
2. Convertir cada substring de texto de longitud m

$H_i = (H_{i-1} - S[i-1] * B^{m-1}) * B + S[i+m-1]$

EDIT DISTANCE

```
int edit_distace(char str1[], char str2[]) {
  int m = strlen(str1), n = strlen(str2), mat[m + 1][n + 1]={0};
  for (i = 1; i <= m; i++) mat[i][0] = i;
  for (i = 1; i <= n; i++) mat[0][i] = i;
  for (j = 1; j <= n; j++) {
    for (i = 1; i <= m; i++) {
      if (str1[i - 1] == str2[j - 1]) mat[i][j] = mat[i - 1][j - 1]; // no hacer nada
      else
       mat[i][j] = 1 + min( min( mat[i - 1][j], mat[i][j - 1]), mat[i - 1][j - 1] );
            //Eliminar, Insertar, Sustituir
      if (i > 1 && j > 1 && str1[i - 2] == str2[j - 1]
                         && str1[i - 1] == str2[j - 2])
        mat[i][j] = min(mat[i][j], 1 + mat[i - 2][j - 2]); // intercambiar
    }
  }
  return mat[m][n];
}
```

AHO-CORASICK

```
const int alfa = 27;

int Convertir(char x){
  if (x > 'Z') return x - 'a' + 1;
  return 26 + x - 'A' + 1;
}

struct PMA {
  PMA *suf;
  PMA *next[ alfa ];
  int accept;
  PMA() {
    accept=-1;
    suf=0;fill(next, next + alfa , (PMA*) 0 );
  }
} TR[4000004];

int tt;
```

```
PMA *NEW(){
  if(tt == 4000000) tt=0; TR[tt] = PMA();
  return &TR[tt++];
}

char p[5010][5010];
int father[5010];

PMA *buildPMA(int size) {
  PMA *root = NEW();
  for ( int i = 0 ; i < size; ++i) {
    PMA *t = root;
    for ( int j = 0 ; p[i][j]; ++j) {
      int c = Convertir(p[i][j]);
      if (t->next[c] == NULL ) t->next[c] = NEW();
      t = t->next[c];
    }
    if(t->accept == -1) t->accept = i;
    father[i] = t->accept;
  }
  queue<PMA*> Q;
  for ( int c = 1 ; c < alfa ; ++c) {
    if (root->next[c]) {
      root->next[c]->next[ 0 ] = root;
      Q.push(root->next[c]);
    } else root->next[c] = root;
  }

while (!Q.empty()){
    PMA *t = Q.front(); Q.pop();
    for ( int c = 1 ; c < alfa ; ++c)
      if (t->next[c]) {
        Q.push(t->next[c]);
        PMA *r = t->next[ 0 ];
        while (!r->next[c]) r = r->next[ 0 ];
        t->next[c]->next[ 0 ] = r->next[c];
        if (t->next[c]->next[0]->accept!=-1)
          t->next[c]->suf = t->next[c]->next[0];
        else
          t->next[c]->suf = t->next[c]->next[0]->suf;
      }
  }
  return root;
}
```

```
int esta[5010],n;

void  match( char *t, PMA *v) {
  for ( int i = 0 ; i < n; ++i) {
    int c = Convertir(t[i]);
    while (!v->next[c]) v = v->next[0];
    v = v->next[c];
    if(v->accept != -1) esta[v->accept]++;
    for (PMA *u = v->suf; u; u = u->suf)
      if(u->accept != -1) esta[u->accept]++;
  }
}
```

EXPRESIONES REGULARES

```
Pattern pattern = Pattern.compile("<REGEX>");
Matcher matcher = pattern.matcher("<INPUT>");
while(matcher.find())
  System.out.println ( matcher.group() + " " + matcher.start()+"
"+matcher.end() );
if (matcher.matches()){}
[abc] -> a, b, or c (simple class)
[^abc] -> Any character except a, b, or c (negation)
[a-zA-Z]  -> a hasta z or A hasta Z, inclusive (range)
[a-d[m-p]] -> a hasta d, or m hasta p: [a-dm-p] (union)
[a-z&&[def]] ->  d, e, or f (intersection)
[a-z&&[^bc]] ->  a through z, except for b and c:
[ad-z] (subtraction)
[a-z&&[^m-p]] -> a through z, and not m through p:
[a-lq-z](subtraction)
.  -> Any character
\d -> A digit: [0-9]
\D -> A non-digit: [^0-9]
\s -> A whitespace character: [ \t\n\x0B\f\r]
\S -> A non-whitespace character: [^\s]
\w -> A word character: [a-zA-Z_0-9]
\W -> A non-word character: [^\w]
\p{Punct} -> One of !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
\p{Lower} -> A lower-case alphabetic character: [a-z]
\p{Upper} -> An upper-case alphabetic character:[A-Z]
\p{Alpha} -> An alphabetic character
\p{Digit} -> A decimal digit: [0-9]
\p{Alnum}  -> An alphanumeric character: [\p{Alpha}\p{Digit}]
\p{XDigit}  -> A hexadecimal digit: [0-9a-fA-F]
\p{Space}  -> A whitespace character: [\t\n\x0B\f\r]
```

```
X?     -> X, once or not at all
X*     -> X, zero or more times
X+     -> X, one or more times
X{n}   -> X, exactly n times
X{n,}  -> X, at least n times
X{n,m}-> X, at least n but not more than m times
X|Y    ->  Either X or Y
```

EVALUADOR DE EXPRESIONES

```
import javax.script.*;
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine motor = manager.getEngineByName("js");
motor.put("x", 5);
motor.eval("(x+5)");
```

BELLMAN FORD

```
struct Edge{ int u,v,w; } edge[10005];
bool bellman_ford(int source, int edges){
  for(int i = 1; i <= n+1; ++i) H[i] = INF;
  H[source] = 0;
  for(int i = 0; i < n; ++i)
    for(int j = 0; j < edges; ++j)
      if(H[ edge[j].v ] > H[ edge[j].u ] + edge[j].w)
        H[ edge[j].v ] = H[ edge[j].u ] + edge[j].w;
  for(int i = 0; i < edges; ++i)
     if(H[ edge[i].v ] > H[ edge[i].u ] + edge[i].w)
       return true; // ciclo negativo detectado
  return false;
}
```

DININC

```
int A,V,fuente,dest,id,oo = 10000001;
int cap[MAXA],flow[MAXA],ady[MAXA],next[MAXA],last[MAXV];
int now[MAXA],level[MAXV],Q[MAXV];

void ini(){
  A = id = fuente = 0; dest = V;
  memset(last,-1,sizeof(last));
}

void Add_Edge(int u, int v, int c){
  cap[A] = c; flow[A] = 0; ady[A] = v;
  next[A] = last[u]; last[u] = A++;
}
```

```
bool BFS(int s, int t){
  int i,u,v,ent,sal;
  memset(level,-1,sizeof(level));
  ent = sal = level[s] = 0;
  Q[ent++] = s;
  while(sal < ent && level[t] == -1){
    u = Q[sal++];
    for(i = last[u]; i != -1; i = next[i]){
      v = ady[i];
      if(level[v] == -1 && flow[i] < cap[i]){
        level[v] = level[u] + 1;
        Q[ent++] = v;
      }
    }
  }
  return level[t] != -1;
}

int DFS(int v, int flujo){
  if(v == dest) return flujo;
  for(int f,i = now[v]; i != -1; now[v] = i = next[i]){
    if (level[ady[i]] > level[v] && flow[i] < cap[i]){
      if( (f = DFS(ady[i], min(flujo ,cap[i] - flow[i]))) ){
        flow[i]   += f;
        flow[i^1] -= f;
        return f;
      }
    }
  }
  return 0;
}

long long int maxFlow(){
  long long res, flujo = 0;
  while(BFS(fuente,dest)){
    for(int i = 0; i < V; i++)
      now[i] = last[i];
    while((res = DFS(fuente,oo))>0)
      flujo += res;
  }
  return flujo;
}
```

MAX FLOW MIN COST

```
int e, u[MAXA], v[MAXA], w[MAXA], cap[MAXA], last[MAXV],
next[MAXA];
void addEdge(int a, int b, int co, int ca){
  u[e] = a; v[e] = b; w[e] = co; cap[e] = ca;
  next[e] = last[a]; last[a] = e++;
  u[e] = b; v[e] = a; w[e] = -co; cap[e] = 0;
  next[e] = last[b]; last[b] = e++;
}
int pre[MAXV], d[MAXV];
bool inq[MAXV];
queue<int> q;
void init(){
  e = 0;
  memset(last, -1, sizeof(last));
}

bool spac(int s, int t, int cnt){
  int x,y, i;
  memset(pre, -1, sizeof(pre));
  memset(inq, 0, sizeof(inq));
  for(i = 0; i <= cnt; ++i) d[i] = INF;
  d[s] = 0; inq[s] = true; q.push(s);
  while(!q.empty()){
    x = q.front(); q.pop(); inq[x] = false;
    for(i = last[x]; i != -1; i = next[i]){
      y = v[i];
      if(cap[i] != 0 && d[y] > d[x] + w[i]){
        d[y] = d[x] + w[i];  pre[y] = i;
        if(!inq[y]){
          inq[y] = true;
          q.push(y);
        }
      }
    }
  }
  return d[t] != INF;
}

int cost, flow;
void mfmc(int s, int t, int cnt){
  cost = flow = 0; int i, tmp;
  while(spac(s, t, cnt)){
    tmp = INF;
    for(i = pre[t]; i != -1; i = pre[u[i]])
      tmp = min(tmp, cap[i]);
    for(i = pre[t]; i != -1; i = pre[u[i]]){
      cap[i] -= tmp; cap[i^1] += tmp;
      cost += w[i];
    }
    flow += tmp;
  }
}
```

TARJAN ARTICULATION POINTS AND BRIDGES

```
int ndfs, vdfs[MAX], low[MAX];

bool apoint[MAX];
// puentes si y solo si vdfs[u] < low[v]
void tarjan(int u, int p = -1){
  int v;
  vdfs[u] = low[u] = ndfs++;
  for(int i = G[u].size()-1; i >= 0; --i){
    v = G[u][i];
    if(!vdfs[v]){
      tarjan(v, u);
      low[u] = min(low[u], low[v]);
      if((vdfs[u] == 1 && vdfs[v] > 2) || (vdfs[u] != 1 &&
vdfs[u] <= low[v]))
        apoint[u] = true;
    } else if(v != p) low[u] = min(low[u], vdfs[v]);
  }
}

void BCC(int n){
  for(int i = 0; i < n; ++i) vdfs[i] = apoint[i] = 0;
  for(int i = 0; i < n; ++i)
    if(!vdfs[i]){
      ndfs = 1;
      tarjan(i);
    }
}
```

TARJAN SCC

```
vector<int> G[MAX];
int comp[MAX], index[MAX], lowlink[MAX], stack[MAX], top, ndfs,
cmp;
bool visited[MAX], parent[MAX];
void Tarjan(int u){
  index[u] = lowlink[u] = ndfs++;
  stack[top++] = u; visited[u] = 1;
  for(int i = 0; i < G[u].size(); ++i){
    int w = G[u][i];
    if(!index[w]){
      Tarjan(w);
      lowlink[u] = min(lowlink[u], lowlink[w]);
    } else if(visited[w])
      lowlink[u] = min(lowlink[u], index[w]);
  }
  if(index[u] == lowlink[u]){
    int w;
    do{
      w = stack[--top];
      comp[w] = cmp;
      visited[w] = 0;
    }while(index[u] != index[w]);
    cmp++;
  }
}
void SCC(){
  top = ndfs = cmp = 1;
  for(int i = 0; i < n; ++i) index[i] = visited[i] = 0;
  for(int i = 0; i < n; ++i) if(!index[i]) Tarjan(i);
}
```

HOPCROFT-KARP

```
const int MAXV = 1001;
const int MAXV1 = 2*MAXV;
int N,M;
vector<int> ady[MAXV];
int D[MAXV1],Mx[MAXV], My[MAXV];

bool BFS(){
  int u, v, i, e;
  queue<int> cola;
  bool f = 0;
  for (i = 0; i < N+M; i++) D[i] = 0;
  for (i = 0; i < N; i++) if (Mx[i] == -1) cola.push(i);
  while (!cola.empty()){
    u = cola.front(); cola.pop();
    for (e = ady[u].size()-1; e >= 0; e--) {
      v = ady[u][e];
      if (D[v + N]) continue;
      D[v + N] = D[u] + 1;
      if (My[v] != -1){
        D[My[v]] = D[v + N] + 1;
        cola.push(My[v]);
      }else f = 1;
    }
  }
  return f;
}

int DFS(int u){
  for (int v, e = ady[u].size()-1; e >=0; e--){
    v = ady[u][e];
    if (D[v+N] != D[u]+1) continue;
    D[v+N] = 0;
    if (My[v] == -1 || DFS(My[v])){
      Mx[u] = v;  My[v] = u;  return 1;
    }
  }
  return 0;
}

int Hopcroft_Karp(){
  int i, flow = 0;
  for (i = max(N,M); i >=0; i--) Mx[i] = My[i] = -1;
  while (BFS())
    for (i = 0; i < N; i++)
      if (Mx[i] == -1 && DFS(i)) ++flow;
  return flow;
}
```

HUNGARIAN

```
int N,A[MAXN+1][MAXN+1],p,q, oo;
int fx[MAXN+1],fy[MAXN+1],x[MAXN+1],y[MAXN+1];

int hng(int oo){
  memset(fx,0,sizeof(fx)); memset(fy,0,sizeof(fy));
  memset(x,-1,sizeof(x)); memset(y,-1,sizeof(y));
  for(int i = 0; i < N; ++i)
   for(int j = 0; j < N; ++j) fx[i] = max(fx[i],A[i][j]);
  for(int i = 0; i < N; ){
    vector<int> t(N,-1), s(N+1,i);
    for(p = q = 0; p <= q && x[i]<0; ++p)
     for(int k = s[p], j = 0; j < N && x[i]<0; ++j)
       if (fx[k]+fy[j]==A[k][j] && t[j]<0){
         s[++q]=y[j]; t[j]=k;
         if(s[q]<0)
           for(p=j; p>=0; j=p)
             y[j]=k=t[j], p=x[k], x[k]=j;
       }
    if (x[i]<0){
      int d = oo;
      for(int k = 0; k < q+1; ++k)
       for(int j = 0; j < N; ++j)
        if(t[j]<0) d=min(d,fx[s[k]]+fy[j]-A[s[k]][j]);
      for(int j = 0; j < N; ++j) fy[j]+=(t[j]<0?0:d);
      for(int k = 0; k < q+1; ++k) fx[s[k]]-=d;
    }else ++i;
  }

  int ret = 0;
  for(int i = 0; i < N; ++i) ret += A[i][x[i]];
  return ret;
}
```

HUNGARIAN EXTENDIDO

```
int cost[1000][1000], r[1000][1000];
int work[1000], task[1000], n, m;
int lx[1000], ly[1000], vx[1000], vy[1000];
int slack[1000], slackx[1000], Enl[1000];
int t, i, j, k, u, bot, delta, ans;
bool found;

int HungarianExtendend() {
  for (u = 0; u < n; u++)
    while (work[u]) {
      for (i = 0; i < n; i++) {
        vx[i] = 0;
        Enl[i] = -1;
      }
      for (i = 0; i < m; i++) {
        vy[i] = 0;
        slack[i] = lx[u] + ly[i] - cost[u][i];
        slackx[i] = u;
      }
      vx[u] = 1;
      while (1) {
        delta = 0x7fffffff;
        found = false;
        for (i = 0; i < m; i++)
          if (!vy[i]) {
            delta = min(slack[i], delta);
            if (slack[i] == 0) {
              vy[i] = 1;
              if (task[i]) {
                bot = min(work[u], task[i]);
                for (j = slackx[i]; Enl[j] != -1;
                     j = slackx[Enl[j]])
                  bot = min(bot, r[j][Enl[j]]);
                work[u] -= bot; task[i] -= bot;
                for (j = i; j != -1; j = Enl[slackx[j]]) {
                  r[slackx[j]][j] += bot;
                  if (Enl[slackx[j]] != -1)
                    r[slackx[j]][Enl[slackx[j]]] -= bot;
                }
                found = true;
              } else {
                for (j = 0; j < n; j++)
                  if (!vx[j] && r[j][i]) {
                    Enl[j] = i; vx[j] = 1;
                    for (k = 0; k < m; k++)
                      if (!vy[k] && slack[k] > lx[j] +
                                        ly[k] - cost[j][k]) {
                        slack[k] = lx[j] + ly[k] - cost[j][k];
                        slackx[k] = j;
                      }
                  }
              }
              break;
```

```
          }
        }
      if (found) break;
      if (delta) {
        for (i = 0; i < n; i++) if (vx[i]) lx[i] -= delta;
        for (i = 0; i < m; i++)
          if (vy[i]) ly[i] += delta;
          else slack[i] -= delta;
      }
    }
  }
  ans = 0;
  for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      ans -= r[i][j] * cost[i][j];
  return ans;
}

int main() {
  scanf("%d%d", &n, &m);
  //n cantidad de tipos de obrero m cantidad de tareas
  for (i = 0; i < n; i++) {
    scanf("%d", &work[i]);
    // cantidad de cada uno de los obreros
    lx[i] = -0x80000000;
  }
  for (i = 0; i < m; i++) {
    scanf("%d", &task[i]);
    // cantidad de cada uno de las tareas
    ly[i] = 0;
  }
  for (i = 0; i < n; i++)
    for (j = 0; j < m; j++) {
      scanf("%d", &cost[i][j]);
      r[i][j] = 0;
      cost[i][j] = -cost[i][j]; // matriz de costos
      lx[i] = max(cost[i][j], lx[i]);
    }
  printf("%d\n", HungarianExtendend());
}
```

HEAVY-LIGHT

```
struct propagacion{
  int sum, expande;
  bool p;
};

int N, M;
vector<int> G[MAXN+1];
int P[MAXN+1][17], T[MAXN+1], L[MAXN+1], H[MAXN+1], ntiras;
int pos[MAXN+1], cad[MAXN+1], join[MAXN+1];
bool mark[MAXN+1];
vector<int> tiras[MAXN+1];
vector<propagacion> tree[MAXN+1];

void dfs(int v){
  H[v] = 1; mark[v] = 1;
  int i, M = G[v].size(), ady;
  for(i = 0; i < M; ++i){
   ady = G[v][i];
   if(!mark[ady]){
     T[ady] = v; L[ady] = L[v] + 1;
     dfs(ady);
     H[v] += H[ady];
   }
  }
}

void process3(){
  int i, j;
  for(i = 1; i <= N; ++i)
   for(j = 0; 1 << j <= N; ++j) P[i][j] = -1;

  for(i = 2; i <= N; ++i) P[i][0] = T[i];

   for(j = 1; 1 << j <= N; ++j)
    for(i = 2; i <= N; ++i)
     if(P[i][j-1] != -1)
      P[i][j] = P[P[i][j-1]][j-1];
}

int LCA(int p, int q){
  int i,log;
  if(L[p] < L[q]) swap(p,q);

  for(log = 1; 1 << log <= L[p]; ++log);
  log--;
```

```
  for(i = log; i >= 0; --i)
   if(L[p]-(1<<i) >= L[q]) p = P[p][i];

  if(p==q) return p;

  for(i = log; i >= 0; --i)
   if(P[p][i] != -1 && P[p][i] != P[q][i])
     p = P[p][i], q = P[q][i];

  return T[p];
}
void descomponerTree(int padre, int v){
  tiras[ntiras].push_back(v);
  pos[v] = tiras[ntiras].size()-1;
  cad[v] = ntiras;
  int i, M = G[v].size(), mayor=-1, son, ady;
  for(i = 0; i < M; ++i){
    ady = G[v][i];
    if(H[ady] > mayor && ady != padre){
      mayor = H[ady];
      son = ady;
    }
  }
  if(mayor!=-1){
    descomponerTree(v,son);
    for(i = 0; i < M; ++i){
      ady = G[v][i];
      if(ady != padre && ady != son){
        join[++ntiras] = v;
        descomponerTree(v,ady);
      }
    }
  }
}
int a, b, who;
void lazy(int ini, int fin, int node){
  int aux = (ini+fin)/2;
  if(tree[who][node].p){
    tree[who][2*node].sum += tree[who][node].expande*(aux-
ini+1);
    tree[who][2*node].expande += tree[who][node].expande;
    tree[who][2*node].p = 1;
    tree[who][2*node+1].sum += tree[who][node].expande*(fin-
aux);
    tree[who][2*node+1].expande += tree[who][node].expande;
    tree[who][2*node+1].p = 1;

    tree[who][node].expande = 0;
    tree[who][node].p = 0;
  }
}

void update(int ini, int fin, int node){
  if(ini > b || fin < a) return;
  if(ini >= a && fin <= b){
    tree[who][node].sum += (fin-ini+1);
    tree[who][node].expande++;
    tree[who][node].p = 1;
    return ;
  }
  lazy(ini, fin, node);
  update(ini,(ini+fin)/2,2*node);
  update((ini+fin)/2+1,fin,2*node+1);
  tree[who][node].sum = tree[who][2*node].sum +
tree[who][2*node+1].sum;
}

int query(int ini, int fin, int node){
  if(ini > b || fin < a) return 0;
  if(ini >= a && fin <= b) return tree[who][node].sum;
  lazy(ini,fin,node);
  int s1 = query(ini,(ini+fin)/2,2*node);
  int s2 = query((ini+fin)/2+1,fin,2*node+1);
  return s1 + s2;
}

void findUpdate(int lca, int v){
  int t;
  if(cad[lca]==cad[v]){
    a = pos[lca] + 1; b = pos[v];
    who = cad[lca];
    t = tiras[cad[lca]].size();
    if(a <= b) update(0,t-1,1);
  }else{
    a = 0; b = pos[v]; who = cad[v];
    t = tiras[cad[v]].size();
```

```
    update(0,t-1,1);
    findUpdate(lca,join[cad[v]]);
  }
}


int findQuery(int lca, int v){
  int t;
  if(cad[lca]==cad[v]){
    a = pos[lca]+1; b = pos[v];
    who = cad[lca];
    if(a>b) return 0;
    t = tiras[cad[lca]].size();
    return query(0,t-1,1);
  }else{
    a = 0; b = pos[v];
    who = cad[v];
    t = tiras[cad[v]].size();
    return query(0,t-1,1) + findQuery(lca,join[cad[v]]);;
  }
}

int main(){
  int i, u, v, t;
  scanf("%d%d", &N, &M);
  for(i = 1; i < N; ++i){
    scanf("%d%d",&u,&v);
    G[u].push_back(v);
    G[v].push_back(u);
  }

  dfs(1);
  process3();
  descomponerTree(-1,1);

  for(i = 0; i <= ntiras; ++i){
    t = tiras[i].size();
    tree[i] = vector<propagacion>(4*t+1);
  }
  char ch[2]; int lca;
  for(i = 0; i < M; ++i){
    scanf("%s %d %d", ch, &u, &v);
    lca = LCA(u,v);
    if(ch[0] == 'P'){
      findUpdate(lca,u);
      findUpdate(lca,v);
    }else{
      int suma1 = findQuery(lca,u);
      int suma2 = findQuery(lca,v);
      printf("%d\n",suma1+suma2);
    }
  }
}
```

RMQ

```
int A[MAXN], RMQ[MAXN][MAXLOG];
void PREPROCESSED_RMQ() {
  memset(RMQ, 0, sizeof(RMQ));
  for (int i = 0; i < N; i++) RMQ[i][0] = i;
  for (int j = 1; (1 << j) <= N; j++)
    for (int i = 0; i + (1 << j) - 1 < N; i++)
      if (A[RMQ[i][j - 1]] < A[RMQ[i + (1 << (j - 1))][j - 1]])
        RMQ[i][j] = RMQ[i][j - 1];
      else
        RMQ[i][j] = RMQ[i + (1 << (j - 1))][j - 1];
}
int CALCULE_RMQ(int i, int j) {
  int k = (int) (log((double) (j - i + 1)) / log(2.0));
  if (A[RMQ[i][k]] < A[RMQ[j - (1 << k) + 1][k]])
    return A[RMQ[i][k]];
  else
    return A[RMQ[j - (1 << k) + 1][k]];
}
```

DISJOIN-SET

```
int get(int x){
  if(x != cmp[x]) cmp[x] = get(cmp[x]);
  return cmp[x];
}
void join(int x, int y){
  x = get(x); y = get(y);
  cmp[x] = y;
}
```

JOSEPHUS

```
int josephus(int n, int k) {
  int f = 0;
  for (int i = 0; i < n; i++) f = (f + k) % (i + 1);
  return f + 1;
}
```

MILLER RABIN
```
ll multiply(ll a, ll b, ll mod) {
  ll rx = 0, sx = 0;
  int bx;
  for (bx = 0; b >> bx > 0; ++bx) {
    sx += (bx) ? sx : a;
    if (sx >= mod) sx -= mod;
    rx += ((b >> bx) & 1) ? sx : 0;
    if (rx >= mod) rx -= mod;
  }
  return rx;
}
ll modpow(ll a, ll b, ll mod) {
  ll rx = 1, sx = 0; int bx;
  for (bx = 0; b >> bx > 0; ++bx) {
    sx = (bx) ? multiply(sx, sx, mod) : a;
    rx = ((b >> bx) & 1) ? multiply(rx, sx, mod) : rx;
  }
  return rx;
}
ll f(ll x, ll mod) {
  ll rx = multiply(x, x, mod) + 123;
  while (rx >= mod) rx -= mod;
  return rx ? rx : 2;
}
bool miller_rabin(ll n, ll iter) {
  ll m = n - 1, b = 2, z;
  int i, j, a = 0;
  while (!(m & 1)) {
    m >>= 1;
    ++a;
  }
  for (int i = 0; i < iter; ++i) {
    j = 0; z = modpow(b, m, n);
    while (!((j == 0 && z == 1) || z == n - 1)) {
      if ((j > 0 && z == 1) || ++j == a) return false;
      z = modpow(z, 2, n);
    }
    b = f(b, n);
  }
  return true;
}
bool is_prime(ll n) {
  return ((n==2) || ((n>1) && (n & 1) && miller_rabin(n, 1)));
}
```

MINIMA CANTIDAD DE PASOS DE UN CABALLO
```
ll dist(ll x1, ll y1, ll x2, ll y2) {
  ll dx = mod(x2 - x1);
  ll dy = mod(y2 - y1);
  ll lb = (dx + 1) / 2;
  lb = max(lb, (dy + 1) / 2);
  lb = max(lb, (dx + dy + 2) / 3);
  while ((lb % 2) != (dx + dy) % 2) lb++;
  if (mod(dx) == 1 && !dy) return 3;
  if (mod(dy) == 1 && !dx) return 3;
  if (mod(dx) == 2 && mod(dy) == 2) return 4;
  return lb;
}
```

DATE -> 0:SUNDAY, 1:MONDAY, ...
```
int zeller(int y, int m, int d){
  if (m<3) --y, m+=12;
  return (y+y/4-y/100+y/400+(13*m+8)/5+d)%7;
}
```

CRIBA N*Log(N)
```
#define MAXN 10000001
#define MAXL (MAXN / (8*sizeof(int)))
int cases, n, hprimes = 2, bit[MAXL + 1];
int  primes[MAXN] = { 2, 3 }, pos[2] = {4, 2 };
bool isOn(int pos) {
 int seg = pos / (8 * sizeof(int));
 int off = pos % (8 * sizeof(int));
 return bit[seg] & (1 << off);
}
void set(int pos) {
 int seg = pos / (8 * sizeof(int));
 int off = pos % (8 * sizeof(int));
 bit[seg] |= (1 << off);
}
void criba() {
 bool p = 1;
 for (int i = 5; i < MAXN; i += pos[p], p = !p)
  if (!isOn(i)) {
   primes[hprimes++] = i;
```

```
   if (i < 10000001 / i)
    for (int j = i * i; j < MAXN; j += (i << 1)) set(j);
  }
}
```

EUCLIDES EXTENDIDO

```
ll GCDext(ll a, ll b, ll& x, ll& y){
  ll g = a; x = 1; y = 0;
  if(b != 0){
    g = GCDext(b, a%b, y, x);
    y -= (a/b)*x;
  }
  return g;
}
```

```
a*x + b*y = c, x = inv si c = 1 y b = mod
bool GCDdiofanto(ll a,ll b,ll c,ll &x,ll &y){
  int r = GCDext(a,b,x,y);
  if (c%r != 0) return false;
  x *= c/r; y *= c/r;
  return true;
}
```

```
Inverso multiplicativo a*inv == 1 (mod m)
bool invMult(ll a,ll m,ll &inv) {
  ll x, y, r;
  r = GCDext(a, m, x, y);
  if (r!=1) return false;
  inv = x;
  if (inv<0) inv += m;
  return true;
}
```

```
a*x == b (mod n)
bool MLE(ll a,ll b,ll n,ll &x){
  ll d, xx, y;
  d = extGcd(a,n,xx,y);
  if (b%d) return false;
  x = ((xx*(b/d))%n+n)%n;
  return true;
}
```

```
TRC x==r[i] (mod m[i])
bool TRC(vector<ll> r,vector<ll> m,ll &x,ll &M){
  int n=sz(r);
  ll inv; x=0; M=1;
  for(int i =0; i < n; i++) M*=m[i];
  for(int i =0; i < n; i++){
    if(!invMult(M/m[i],m[i],inv)) return false;
    x+=r[i]*(M/m[i])*inv;
  }
  x=(x%M);
  return true;
}
```

EULER'S TOTIENT FUNCTION (PHI(N))

```
ll Euler_Totient_Function(ll n){
   ll ans = n;
   for(ll i=2;i*i<= n;i++){
      if(n %i==0) ans -= ans/i;
      while(n%i==0) n/=i;
   }
   if(n>1) ans -=ans/n;
   return ans;
}
```

EULER'S TOTIENT THEOREM

a, n coprime, then a^phi(n) == 1 (mod n)

TEOREMA DE WILSON

p primo, (p-1)! == -1 mod(p).

FERMAT'S LITTLE THEOREM

p prime, a coprime to p, we have a^p = a (mod p)

DISCRETE LOGARITHM THEOREM

g raiz primitiva de Zn, entonces g^x == g^y (mod n) se cumple si y solo si se cumple x == y mod(phi(n)).
g es una raiz primitiva mod n si las potencias de g modulo n van por todos los coprimos de n.
La raiz primitiva existe si n = 2, 4, p^k o 2*p^k donde p es un primo impar.
Para comprbar que g es una raiz primitiva de n solo tenemos q comprobar que g^d != 1 mod(n) para todo primo p que divide a phi(n), d = phi(n)/p.

CANTIDAD DE DIGITOS DE N!

```
(ll)floor((log(2*M_PI*n)/2+n*(log(n)-1))/log(10))+1);
```

PROBABILIDAD

P(E1|E2)=P(E1^E2)/P(E2)

TEOREMA DE BAYES

P(E1|E2)=P(E1)*P(E2|E1)/P(E2)

BERNOULLI

Una prueba de Bernoulli es aquella que puede terner 2 resultados exito o fallo. Si la probabilidad de exito de una prueba de Bernoulli es p, la probabilidad de q ocurran k exitos en una secuencia de n eventos idependientes es: C(n,k)*(p^k)*(1-p)^(n-k).

STIRLING NUMBER OF THE SECOND KIND

{m,n} = (1/n!)*∑(-1)^(n-k)*C(n,k)*(k^m)

CLASSICAL OCCUPANCY PROBLEM

En una urna con m bolas numeradas de 1 a m. Suponga que extraemos n bolas una por una, con reemplazamientos. La probabilidad de que hayan sido extraídas exactamente t bolas diferentes es:
P1(m,n,t) = {n,t}*(m^(t))/(m^n).

PROBLEMA DEL CUMPLEANNO

En una urna con m bolas numeradas de 1 a m. Suponga que extraemos n bolas una por una, con reemplazamientos. La probabilidad de que haya una coincidencia es:
P2(m,n) = 1-P1(m,n,n) = 1-(m^(n))/(m^n)
≈ 1-exp(-(n*n)/(2*m)). exp(x) = e^x.
Si sacamos n1 bolas de una urna y n2 bolas de otra con reemplazo, la probabilidad de coincidencia es:
P3(m,n1,n2) = 1-(1/m^(n1+n2))*
∑(m^(t1+t2)*{n1,t1}*{n2,t2}) ≈ 1-exp(-(n*n)/m).
Si sacamos n1 bolas de una urna y n2 bolas de otra sin reemplazo, la probabilidad de coincidencia es:
P4(m,n1,n2) = 1 - (m^((n1+n2)))/(m^(n1)+m^(n2)).
Si sacamos n1 bolas de una urna con remplazo y n2 bolas de otra sin reemplazo, la probabilidad de coincidencia es:
P5(m,n1,n2) = 1-(1-n2/m)^n1.

GAUSS 1-0

```
int r, c, n;
int mat[MAXR * MAXC + 1][MAXR * MAXC + 1];
//coef 1-n y result en 0
int index_[MAXR * MAXC + 1];
int x[MAXR * MAXC + 1], X;
void S(int pos, int num) {
  if (num == X) return;
  if (pos == 0) {
    if (num < X) X = num;
    return;
  }
  if (!mat[index_[pos]][pos]) {
    x[pos] = 1; S(pos - 1, num + 1);
    x[pos] = 0; S(pos - 1, num);
    return;
  }
  int left = 0;
  for (int j = n; j > pos; j--)
    if (mat[index_[pos]][j]) left ^= x[j];
  x[pos] = mat[index_[pos]][0] ^ left;
  x[pos] ? S(pos - 1, num + 1) : S(pos - 1, num);
}
void G() {
  for (int i = 1; i <= n; i++) index_[i] = i;
  bool no = false;
  for (int i = 1; i <= n && !no; i++) {
    for (int j = i; j <= n; j++)
      if (mat[index_[j]][i]) {
        int aux = index_[i];
        index_[i] = index_[j];
        index_[j] = aux;
        break;
      }
    if (!mat[index_[i]][i]) continue;
    for (int j = i + 1; j <= n; j++) {
      if (mat[index_[j]][i]) {
        for (int k = i; k <= n; k++)
          mat[index_[j]][k] ^= mat[index_[i]][k];
        mat[index_[j]][0] ^= mat[index_[i]][0];
        if (mat[index_[j]][0]) {
          int left = 0;
          for (int k = i + 1; k <= n && !left; k++)
            left += mat[index_[j]][k];
          if (!left) {
            no = true;
            break;
          }
        }
      }
    }
  }
  if (no) {
```

```
    puts("No solution");
    return;
  }
  X = n + 1;
  S(n, 0);
  printf("You have to tap %d tiles.\n", X);
}
```

GAUSS A – coef B – termino independiente

```
vector<vector<int> > A; vector<double> B;
int n;
vector<double> V; vector<int> I;
void gauss(){
  for(int i=0; i < n; i++){
    int p=i;
    for (int j=i+1; j<n; j++)
      if (abs(A[j][i])>abs(A[p][i])) p=j;
      swap(A[p], A[i]); I.pb(p);
      //If A[i][i]==0, LU decomposition failed.
      for (int j = i+1; j<n; j++){
        A[j][i]/=A[i][i];
        for (int k = i+1; k<n; k++) A[j][k]-=A[i][k]*A[j][i];
        V.pb(A[j][i]);
      }
  }
  for (int i=n-1; i>= 0; i--){
    for (int j=i+1; j<n; ++j) V.pb(A[i][j]);
    V.pb(A[i][i]);
  }
}

void sol(){
  int k = 0;
  for(int i = 0; i < n; i++){
    swap(B[I[i]],B[i]);
    for (int j=i+1; j<n; j++) B[j]-=B[i]*V[k++];
  }
  for (int i=n-1; i>=0; i--){
    for (int j=i+1; j<n; j++) B[i]-=B[j]*V[k++];
    B[i]/=V[k++];
  }
}
```

DETERMINANTE

```
double det(){
  double d=1;
  for(int i=0; i < n; i++){
    int p=i;
    for (int j=i+1; j<n; ++j)
      if (abs(A[j][i]) > abs(A[p][i])) p=j;
    swap(A[pivot],A[i]);
    d*=A[i][i]*( i != p ? -1 : 1 );
    if (abs(A[i][i]) < eps) break;
    for (int j=i+1; j < n; j++)
      for (int k=n-1; k >= i; k--)
        A[j][k] -= A[i][k]*A[j][i] / A[i][i];
  }
  return d;
}
```

GEOMETRIA 2D

```
const double EPS = 1e-8;
const double oo = 1e12;
//Tipo Datos
typedef complex<double> P; //Punto
//Operadores
bool operator <(const P &a, const P &b) {
 return real(a) != real(b) ? real(a) < real(b) : imag(a) <
imag(b);
}
struct L: public vector<P> { //Linea
 L(const P & a, const P & b) {
  push_back(a);
  push_back(b);
 }
};
//Metodos
double cross(P a, P b) {
 return imag(conj(a) * b);
}
double dot(P a, P b) {
 return real(conj(a) * b);
}
//Orientacion de 3 puntos
int ccw(P a, P b, P c) {
 b -= a;
 c -= a;
```

```
 if (cross(b, c) > 0)
  return +1; //counter clockwise
 if (cross(b, c) < 0)
  return -1; //clockwise
 if (dot(b, c) < 0)
  return +2; //c - a - b line
 if (norm(b) < norm(c))
  return -2; //a - b - c line
 return 0;
}
//Interseccion de 2 rectas
bool intersectLL(L l, L m) {
 return abs(cross(l[1] - l[0], m[1] - m[0])) > EPS || //non-
parallel
   abs(cross(l[1] - l[0], m[0] - l[0])) < EPS; //same-line
}
//Interseccion recta y segmento
bool intersectLS(L l, L s) {
 return cross(l[1] - l[0], s[0] - l[0]) * //s[0] is left of l
   cross(l[1] - l[0], s[1] - l[0]); //s[1] is right of l
}
//Interseccion recta y punto
bool intersectLP(L l, P p) {
 return abs(cross(l[1] - p, l[0] - p)) < EPS;
}
//Interseccion de 2 segmentos
bool intersectSS(L s, L t) {
 if (abs(s[0] - t[0]) < EPS || abs(s[0] - t[1]) < EPS ||
abs(s[1] - t[0])
   < EPS || abs(s[1] - t[1]) < EPS)
  return 1; // same point
 return ccw(s[0], s[1], t[0]) * ccw(s[0], s[1], t[1]) <= 0 &&
ccw(t[0],
   t[1], s[0]) * ccw(t[0], t[1], s[1]) <= 0;
}
//Interseccion segmento y punto
bool intersectSP(L s, P p) {
      //desigualdad triangular
   return abs(s[0] - p) + abs(s[1] - p) - abs(s[1] - s[0]) <
EPS;
}
//Proyeccion punto recta
P projection(L l, P p) {
 double t = dot(p - l[0], l[0] - l[1]) / norm(l[0] - l[1]);
 return l[0] + t * (l[0] - l[1]);
}
//Refleccion punto recta
P reflection(L l, P p) {
 return p + (P(2, 0) * (projection(l, p) - p));
}
//Distancia recta punto
double distanceLP(L l, P p) {
 return abs(p - projection(l, p));
}
//Distancia recta recta
double distanceLL(L l, L m) {
 return intersectLL(l, m) ? 0 : distanceLP(l, m[0]);
}
//Distancia recta segmento
double distanceLS(L l, L s) {
 if (intersectLS(l, s))
  return 0;
 return min(distanceLP(l, s[0]), distanceLP(l, s[1]));
}
//Distancia segmento punto
double distanceSP(L s, P p) {
 const P r = projection(s, p);
 if (intersectSP(s, r)) return abs(r - p);
 return min(abs(s[0] - p), abs(s[1] - p));
}
//distancia segmento segmento
double distanceSS(L s, L t) {
 if (intersectSS(s, t)) return 0;
 return min(min(distanceSP(s, t[0]), distanceSP(s, t[1])),
min(distanceSP(t,
   s[0]), distanceSP(t, s[1])));
}
//Punto interseccion recta recta
P crosspoint(L l, L m) {
 double A = cross(l[1] - l[0], m[1] - m[0]);
 double B = cross(l[1] - l[0], l[1] - m[0]);
 if (abs(A) < EPS && abs(B) < EPS)
  return m[0]; //Same line
 if (abs(A) < EPS)
  return P(0, 0); //parallels
 return m[0] + B / A * (m[1] - m[0]);
}
```

```
//Centro de circunferencia dado 3 puntos
P circunferenceCenter(P a, P b, P c) {
 P x = 1.0 / conj(b - a), y = 1.0 / conj(c - a);
 return (y - x) / (conj(x) * y - x * conj(y)) + a;
}
```

ROTOTAR POLIGONO

```
P rotar(double x, double y, double degree){
 return P(x*cos(pi*degree) - y*sin(pi*degree),
            x*sin(pi*degree) + y*cos(pi*degree));
}
/* El angulo entre dos puntos geograficos */
double angle(double lai, double laf, double loi, double lof){
  double lamda = fabs(lon2 - lon1) * M_PI/180;
  lat1 *= M_PI/180; lon1 *= M_PI/180;
  lat2 *= M_PI/180; lon2 *= M_PI/180;
  return acos( sin(lat1)*sin(lat2) +
cos(lat1)*cos(lat2)*cos(lamda) );
}
double dist(double r, double lai, double laf, double loi, double
lof){
  return r*angle(lai,laf,loi,lof);
}
```

GEOMETRY 3D

```
//Geometry 3D
struct P3 {
  double x, y, z;
  P3(double x, double y, double z = 0) :
    x(X), y(Y), z(Z) {}
};
struct V3 {
  double x, y, z;
  V3(double x, double x, double z) :
    x(x), y(y), z(z) {}
  V3(P3 p) { x = p.x;  y = p.y;  z = p.z; }
  V3(P3 p, P3 q){ x = q.x - p.x; y = q.y - p.y; z = q.z - p.z; }
};
P3 operator+(const P3 &p, const V3 &v) {
  return P3(p.x + v.x, p.y + v.y, p.z + v.z);
}
P3 operator+(const P3 &p, const P3 &q) {
  return P3(p.x + q.x, p.y + q.y, p.z + q.z);
}
P3 operator-(const P3 &p, const V3 &v) {
  return P3(p.x - v.x, p.y - v.y, p.z - v.z);
}
P3 operator-(const P3 &p, const P3 &q) {
  return P3(p.x - q.x, p.y - q.y, p.z - q.z);
}
V3 operator +(const V3 &u, const V3 &v) {
  return V3(u.x + v.x, u.y + v.y, u.z + v.z);
}
V3 operator-(const V3 &u, const V3 &v) {
  return V3(u.x - v.x, u.y - v.y, u.z - v.z);
}
V3 operator*(const double &a, const V3 &v) {
  return V3(a * v.x, a * v.y, a * v.z);
}
double dot(const V3 u, const V3 v) {
  return u.x * v.x + u.y * v.y + u.z * v.z;
}
V3 cross(const V3 u, const V3 v) {
  return V3(u.y * v.z - u.z * v.y, u.z * v.x - u.x * v.z, u.x *
v.y - u.y * v.x);
}

double norma(const V3 v) {
  return sqrt(dot(v, v));
}
struct recta {
  P3 a, b;
  recta(P3 A, P3 B) :
    a(A), b(B) { }
  recta(P3 P, V3 V) :
    a(P) { b = P + V; }
};
struct semirecta {
  P3 a, b;
  semirecta(P3 A, P3 B) :
    a(A), b(B) { }
  semirecta(P3 P, V3 V) :
    a(P) { b = P + V; }
};
struct segmento {
  P3 a, b;
  segmento(P3 A, P3 B) : a(A), b(B) { } };
```

```
struct triangulo {
  P3 a, b, c;
  triangulo(P3 A, P3 B, P3 C) :
    a(A), b(B), c(C) { }
};
double distancia(const P3 a, const P3 b) {
  return norma(V3(a, b));
}
double distancia(const P3 p, const recta r) {
  V3 v(r.a, r.b), w(r.a, p);
  return norma(cross(v, w)) / norma(v);
}
double distancia(P3 p, semirecta s) {
  V3 v(s.a, s.b), w(s.a, p);
  if (dot(v, w) <= 0) return distancia(p, s.a);
  return distancia(p, recta(s.a, s.b));
}
double distancia(P3 p, segmento s) {
  V3 v(s.a, s.b), w(s.a, p);
  double c1 = dot(v, w), c2 = dot(v, v);
  if (c1 <= 0) return distancia(p, s.a);
  if (c2 <= c1) return distancia(p, s.b);
  return distancia(p, s.a + (c1 / c2) * v);
}
double distancia(recta r, recta s) {
  V3 u(r.a, r.b), v(s.a, s.b), w(r.a, s.a);
  double a = dot(u, u), b = dot(u, v), c = dot(v, v),
            d = dot(u, w), e = dot(v, w);
  double D = a * c - b * b, sc, tc;
  if (D < EPS) sc = 0, tc = (b > c) ? d / b : e / c;
  else sc = (b * e - c * d) / D, tc = (a * e - b * d) / D;
  V3 dP = w + (sc * u) - (tc * v);
  return norma(dP);
}
double distancia(segmento r, segmento s) {
  V3 u(r.a, r.b), v(s.a, s.b), w(s.a, r.a);
  double a = dot(u, u), b = dot(u, v), c = dot(v, v),
           d = dot(u, w), e = dot(v, w);
  double D = a * c - b * b;
  double sc, sN, sD = D;
  double tc, tN, tD = D;
  if (D < EPS) { sN = 0; sD = 1; tN = e; tD = c;
  } else {
    sN = (b*e - c*d);
    tN = (a * e - b * d);
    if (sN < 0) { sN = 0; tN = e; tD = c;
    } else if (sN > sD) {
      sN = sD; tN = e + b; tD = c;
    }
  }
  if (tN < 0) {
    tN = 0;
    if (-d < 0) { sN = 0;
    } else if (-d > a) { sN = sD;
    } else { sN = -d; sD = a; }
  } else if (tN > tD) {
    tN = tD;
    if ((-d + b) < 0) { sN = 0;
    } else if (-d + b > a) { sN = sD;
    } else { sN = -d + b; sD = a; }
  }
  sc = fabs(sN) < EPS ? 0 : sN / sD;
  tc = fabs(tN) < EPS ? 0 : tN / tD;
  V3 dP = w + (sc * u) - (tc * v);
  return norma(dP);
}
V3 projecao(V3 u, V3 v) {
  return (dot(v, u) / dot(u, u)) * u;
}
bool between(P3 a, P3 b, P3 p) {
  return dot(V3(p - a), V3(p - b)) < EPS;
}
double linedist(P3 a, P3 b, P3 p) {
  P3 proj = a + projecao(V3(a, b), V3(a, p));
  if (between(a, b, proj)) {
    return norma(V3(proj, p));
    return min(norma(V3(a, p)), norma(V3(b, p)));
  }
  double distancia(P3 p, triangulo T) {
    V3 X(T.a, T.b), Y(T.a, T.c), P(T.a, p);
    V3 PP = P - projecao(cross(X, Y), P);
    P3 PPP = T.a + PP;
    V3 R1 = cross(V3(T.a, T.b), V3(T.a, PPP));
    V3 R2 = cross(V3(T.b, T.c), V3(T.b, PPP));
    V3 R3 = cross(V3(T.c, T.a), V3(T.c, PPP));
    if (dot(R1, R2) > -EPS && dot(R2, R3) > -EPS && dot(R1, R3)
> -EPS) {
      return norma(V3(PPP, p));
```

```
      return min(linedist(T.a,T.b,p), min(linedist(T.b, T.c, p),
          linedist(T.c, T.a, p)));
    }
  }
}
```

NÚMERO CICLOMÁTICO

```
M : cantidad de Aristas
N : # de vértices
P : # de componentes conexas.
NC =  M - N + P   cantidad de ciclos.
```

NÚMERO DE ESTABILIDAD INTERNA

Un conjunto de vértices se dice que es interiormente estable si dos vértices cualesquiera del conjunto no son adyacentes. El mayor subconjunto interiormente estable de un grafo es conocido como número de estabilidad interna. Lo designaremos por I. En todo grafo se cumple la siguiente relación:

```
   I(G) * NC(G) = Total de vértices de la red.
```

KD-TREE

```
#define MAX 10005
struct point{ double x, y; };
struct cmpX{
  bool operator() (const point &a, const point &b){
    return a.x < b.x;
  }
};
struct cmpY{
  bool operator() (const point &a, const point &b){
    return a.y < b.y;
  }
};
struct KDtree{
  point boundary;
  KDtree *left;
  KDtree *right;
  int d; // dimension 0: left/right split 1: up/down split
  int cantNodes;
  point UpLeft;  //estos puntos
  point UpRight; //me delimitan
  point DownLeft; //el rectangulo
  point DownRight; //que cubre ese borde
};
int N,Q;
point P[MAX];
KDtree *root;
double xmin,xmax,ymin,ymax;

KDtree *build(int a, int b, int depth){
  if(a <= b){
    KDtree *node = new KDtree();
    node->d = depth % 2;
    if(node->d==0) sort(P+a,P+a+(b-a+1),cmpX());
    else sort(P+a,P+a+(b-a+1),cmpY());
    int med = a + (b-a+1)/2;
    node->boundary = P[med];
    node->left = build(a,med-1,depth+1);
    node->right = build(med+1,b,depth+1);
    node->cantNodes = (b-a+1);
    return node;
  }
  return NULL;
}

bool isInside(point target, double radio, KDtree *node ){
  int ok=0;
  if(Euclidean_Dist(target,node->DownLeft)<=radio) ok++;
  if(Euclidean_Dist(target,node->DownRight)<=radio) ok++;
  if(Euclidean_Dist(target,node->UpLeft)<=radio) ok++;
  if(Euclidean_Dist(target,node->UpRight)<=radio) ok++;
  return ok==4;
}

int findPoints(point target, double radio, KDtree *node){
  int cant = 0;
  if(isInside(target,radio,node))return node->cantNodes;;
  if( Euclidean_Dist(target,node->boundary) <= radio ) cant++;
  double spacing = target.x - node->boundary.x;
  if(node->d==1) spacing = target.y - node->boundary.y;
  KDtree *rightSide = (spacing < 0) ? node->left : node->right;
  KDtree *otherSide = (spacing < 0) ? node->right : node->left;
  if(rightSide != NULL)
    cant += findPoints(target,radio,rightSide);
  if(otherSide != NULL && abs(spacing) <= radio)
    cant += findPoints(target,radio,otherSide);
  return cant;
}
```

```
void recorre(KDtree *node){
  if(node->left!=NULL){
    if(node->d == 0){
      node->left->DownLeft = node->DownLeft;
      node->left->DownRight.x = node->boundary.x;
      node->left->DownRight.y = node->DownRight.y;
      node->left->UpLeft = node->UpLeft;
      node->left->UpRight.x = node->boundary.x;
      node->left->UpRight.y = node->UpRight.y;
    }else{
      node->left->DownLeft = node->DownLeft;
      node->left->DownRight = node->DownRight;
      node->left->UpLeft.x = node->UpLeft.x;
      node->left->UpLeft.y = node->boundary.y;
      node->left->UpRight.x = node->UpRight.x;
      node->left->UpRight.y = node->boundary.y;
    }
    recorre(node->left);
  }

  if(node->right != NULL){
    if(node->d == 0){
      node->right->DownLeft.x = node->boundary.x;
      node->right->DownLeft.y = node->DownLeft.y;
      node->right->DownRight = node->DownRight;
      node->right->UpLeft.x = node->boundary.x;
      node->right->UpLeft.y = node->UpRight.y;
      node->right->UpRight = node->UpRight;
    }else{
      node->right->DownLeft.x = node->DownLeft.x;
      node->right->DownLeft.y = node->boundary.y;
      node->right->DownRight.x = node->DownRight.x;
      node->right->DownRight.y = node->boundary.y;
      node->right->UpLeft = node->UpLeft;
      node->right->UpRight = node->UpRight;
    }
    recorre(node->right);
  }
}

int main() {
  xmin=ymin=xmax=ymax=-1;
  scanf("%d%d",&N,&Q);
  for(int i = 0; i < N; ++i){
    scanf("%lf%lf",&P[i].x, &P[i].y);
    if(xmin==-1 || xmin>P[i].x) xmin=P[i].x;
    if(xmax==-1 || xmax<P[i].x) xmax=P[i].x;
    if(ymin==-1 || ymin>P[i].y) ymin=P[i].y;
    if(ymax==-1 || ymax<P[i].y) ymax=P[i].y;
  }
  root = build(0,N-1,0);
  root->DownLeft.x = xmin;
  root->DownLeft.y = ymin;
  root->DownRight.x = xmax;
  root->DownRight.y = ymin;
  root->UpLeft.x = xmin;
  root->UpLeft.y = ymax;
  root->UpRight.x = xmax;
  root->UpRight.y = ymax;
  recorre(root);
  for(int i = 0; i < Q; ++i){
    double x, y, r;
    scanf("%lf%lf%lf",&x,&y,&r);
    point target = {x,y};
    int ans = findPoints(target,r,root);
    printf("%d\n", ans);
  }
  return 0;
}
point findClosest(point target, KDtree *node){
  point closest;
  if(target.x==node->boundary.x && target.y==node->boundary.y)
       closest.x = 1e9, closest.y = 1e9;
  else
       closest = node->boundary;
  int bestDist = ManhattanDist(closest, target);
  int spacing = target.x - node->boundary.x;
  if(node->d==1) spacing = target.y - node->boundary.y;
  KDtree *rightSide = (spacing < 0) ? node->left  : node->right;
  KDtree *otherSide = (spacing < 0) ? node->right : node->left;
  if(rightSide != NULL){
        point candidate = findClosest(target,rightSide);
        if(ManhattanDist(candidate,target) < bestDist){
              closest = candidate;
              bestDist = ManhattanDist(closest,target);
        }
  }
  if(otherSide != NULL && abs(spacing) < bestDist){
```

```
        point candidate = findClosest(target, otherSide);
        if(ManhattanDist(candidate,target) < bestDist){
              closest = candidate;
              bestDist = ManhattanDist(target, closest);
        }
  }
  return closest;
}
```

CONVEX HULL

```
struct point {
  double x, y;
  inline bool operator<(const point b) const {
    return x < b.x || (x == b.x && y < b.y);
  }
} Ptos[MAX], up[MAX / 2 + 1], down[MAX / 2 + 1];

double cross(point & a, point & b, point & c) {
  return a.x*(b.y - c.y) + b.x*(c.y - a.y) + c.x*(a.y - b.y);
}
int convexHull(int n, point *P) {
  sort(P, P + n);
  int c1 = 0, c2 = 0, d = 0;
  point p1 = up[c1++] = down[c2++] = P[0], p2 = P[n - 1];
  for (int i = 1; i < n; i++) {
    if (i == n - 1 || cross(p1, P[i], p2) < 0) {
      while (c1>=2 && cross(up[c1 - 2], up[c1 - 1], P[i]) >= 0)
        c1--;
      up[c1++] = P[i];
    }
    if (i == n - 1 || cross(p1, P[i], p2) > 0) {
      while (c2>=2 && cross(down[c2-2], down[c2-1], P[i]) <= 0)
        c2--;
      down[c2++] = P[i];
    }
  }
  for (int i = 0; i < c1 - 1; i++) P[d++] = up[i];
  for (int i = c2 - 1; i > 0; i--) P[d++] = down[i];
  return d;
}
```

NÚMEROS DE STIRLING

Consideremos un conjunto con n elementos, cuántos conjuntos de k subconjuntos podemos formar que excluyan el elemento vacío y que la unión de ellos, nos da el conjunto original: S(n, k) = S(n − 1, k) + kS(n − 1, k)

NÚMEROS EULERIANOS

Sea p = {a1,a2,...an}, deseamos conocer todas las permutaciones que cumplen la relación ai < ai+1 k veces: Sean {1234} y una permutación {2341}, esta cumple la propiedad 2 veces: 2<3 y 3<4. Los números eulerianos cuentan la cantidad de dichas permutaciones:

E(n, k) = k E(n-1, k) + (n-k+1) E(n-1, k-1)

PARTICIONES ENTERAS

Se quiere contar de cuántas formas se puede escribir un número entero positivo como la suma de k enteros positivos: 3 se escribe como 1+1+1, 1+2, 3, 1 como 3, 1 como 2 y 1 como 1. p(n,k) cuenta las formas de escribir n como k sumandos:

p(n, k) = p(n − 1, k − 1) + p(n − k, k)

NUMBERS THEORY

• Un número R en base N es divisible por (N-1) si y solo si la suma de sus dígitos (en decimal) es divisible por (N-1).

• If p is a prime and a !≡ 0 mod p, ap−1 ≡ 1 mod p; if a (p−1)/2 ≡ 1 mod p then there exist b such that b^2 ≡ a mod p.

• Let n be a positive integer greater than 1 and let its unique prime factorization be $p1^{e1}p2^{e2}. . .pk^{ek}$ where ei > 0 and pi is prime for all i. Then the Euler Φ function Φ(n) = n(1 − 1/p1)(1 − 1/p2) . . . (1 − 1/pk)= $\prod(pi^{ei}-pi^{ei-1})$ describes the number of positive integers co-prime to n in [1..n]. As a special case, Φ(p) = p − 1 for prime p. The number of divisors of n is ∏i(ei + 1).

• Euler's Theorem, which extends Fermat's Little Theorem: If mcd(a, n) = 1, aΦ(n) ≡ 1 mod p.

PROPIEDADES DE FIBONACCI

$F_n = F_{n-2} + F_{n-1}$	$\sum_{i\leq n} F_n = F_{n+2} - 1$
$\sum_{i\leq n} F_i^2 = F_n * F_{n+1}$	$F_n^2 - F_{n-1} * F_{n+1} = -1^n$
$F_{2n} = F_n^2 + 2F_nF_{n-1}$	$F_{2n+1} = F_{n+1}^2 + F_n^2$
$F_{n+m} = F_{m+1} * F_n + F_m * F_{n-1}$	gcd $(F_n, F_m) = F_{gcd(nm)}$ n>=3
$m \equiv 0(mod\ n) \rightarrow F_m \equiv 0(mod\ F_n)$	$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$

SUMA DE LOS DIVISORES DE UN NUMERO

$$\sum d = \prod_{i=1}^{n} \frac{p_i^{e_i+1} - 1}{p_i - 1}$$

SUMA DE LOS GRADOS DE UN GRAFO

$$\sum_{v=1}^{n} degree(v) = 2m$$

```
If G is planar then n - m + f = 2,so f <= 2n - 4; m <= 3n - 6:
Any planar graph has a vertex with degree <= 5.
```

TEORIA DE NUMEROS

```
N=p^a*q^b*r^c
CantDiv = D = (a+1)*(b+1)*(c+1)
SumaDiv = FOR(i,k)
   sum*=(prim[i]^(cant[i]+1)-1)/(prim[i]-1)
ProdDiv = P = N^(D/2)=Sqrt(N^D)
```

CANT DE PALINDROMES DE <= N DIGITOS

```
a(n) = 2 *(10^(n/2) -1) si n es par
a(n) = 11*(10^(n-1)/2)-2 si n es impar
```

GRIRAR GRILLA 45 GRADOS

```
Matriz de N x M
X = X0 + Y0
Y = X0 - Y0 + max(N,M)
```

FAST FOURIER TRANSFORM

```
//a - coeficientes del polinomio
typedef complex < double > base ;

void fft (vector <base> &a, bool invert ) {
  int n = (int) a.size();
  if ( n == 1 )  return ;
  vector < base > a0 ( n / 2 ) ,  a1 ( n / 2 ) ;
  for ( int i = 0 , j = 0 ; i < n ; i += 2 , ++ j ) {
    a0 [ j ] = a [ i ] ;
    a1 [ j ] = a [ i + 1 ] ;
  }
  fft ( a0, invert ) ;  fft ( a1, invert ) ;

  double ang = 2 * M_PI / n * ( invert ? - 1 : 1 ) ;
  base w ( 1 ) ,  wn ( cos ( ang ) , sin ( ang ) ) ;
  for ( int i = 0 ; i < n / 2 ; ++ i ) {
    a [ i ] = a0 [ i ] + w * a1 [ i ] ;
    a [ i + n / 2 ] = a0 [ i ] - w * a1 [ i ] ;
    if ( invert )
      a [ i ] /= 2 ,  a [ i + n / 2 ] /= 2 ;
    w *= wn ;
  }
}

void multiply(vector <int> &a, vector <int> &b, vector <int>
&res){
  vector <base> fa ( a.begin(), a.end() ), fb ( b.begin(),
b.end() ) ;
  int n = 1;
  while ( n < max ( a. size() , b. size() ) ) n <<= 1;
  n <<= 1 ;
  fa. resize ( n ) ,  fb. resize ( n ) ;
  fft ( fa, false ) ,  fft ( fb, false ) ;
  for ( int i = 0 ; i < n ; ++i )
    fa [ i ] *= fb [ i ] ;
  fft ( fa, true ) ;

  res. resize ( n ) ;
  for ( int i = 0 ; i < n ; ++i )
    res [ i ] = int ( fa [ i ] . real ( ) + 0.5 ) ;

  int carry = 0 ;
  for ( int i = 0 ; i < n ; ++ i ){
    res [ i ] += carry;
    carry = res [ i ] / 10 ;
    res [ i ] %= 10;
  }
}
```

DIGIT COUNT

```
void DigitCount(int n,ll *sol){
   ll aux=n, sum=0,p=1,d;
   while(aux){
     d = aux % 10, aux /= 10;
     sol[d] += ((n%p)+1);
     for(int i=0;i<d;i++) sol[i]+=p;
     for(int i=0;i<10;i++) sol[i] += sum*d;
     sol[0] -= p;
     sum = p + 10 * sum;
     p *= 10;
   }
}
```

TREAP

```
srand(time(0));
#define size(r) buff[r].ch[2]
#define hijo(r,i) buff[r].ch[i]
#define PR(r) buff[r].ch[4]
#define key(r) buff[r].ch[3]

struct Treap {
  struct Nodo {
    int ch[5];
    Nodo() {}
    Nodo(int key) {
      ch[0] = ch[1] = 0, ch[4] = rand();
      ch[2] = 1, ch[3] = key;
    }
  } buff[MAXNODES];
  int root, nodes;

 void update_size(int root) {
    size(root) = 1 + size(hijo(root,0)) + size(hijo(root,1));
  }
  void rotate(int &root, bool dir){
    int tmp = hijo(root,dir);
    hijo(root,dir) = hijo(tmp,1 - dir);
    hijo(tmp,1-dir) = root;
    update_size(root);
    update_size(tmp);
    root = tmp;
  }
void insert(int &root, int val) {
    if (root == 0){
      buff[root = ++nodes] = Nodo(val);
      return;
    }
    if (val == key(root)) return;
    bool dir = !(val < key(root));
    insert(hijo(root,dir), val);
    if (PR(root) > PR(hijo(root,dir))) rotate(root, dir);
    update_size(root);
  }
  void erase(int &root, int val) {
    if (root == 0) return;
    if (val != key(root)) {
      bool dir = !(val < key(root));
      erase(hijo(root,dir), val);
    } else {
      int L = hijo(root,0);
      int R = hijo(root,1);
      if (L)
        if (R) rotate(root, PR(L) > PR(R));
        else rotate(root, 0);
      else if (R) rotate(root, 1);
      else {
        root = 0;
        return;
      }
      erase(root, val);
    }
    update_size(root);
  }
  int countLessThan(int root, int val){
    int cant = 0;
    while (root) {
      bool dir = !(val < key(root));
      if (dir) {
        cant += size(hijo(root,0));
        if (val <= key(root)) return cant;
        cant++;
      }
      root = hijo(root,dir);
    }
    return cant;
  }
```

```
  int findKth(int root, int kth) {
    while (root) {
      int v = hijo(root,0);
      if (kth < size(v)) root = v;
      else {
        kth -= size(v) + 1;
        if (kth < 0) return key(root);
        root = hijo(root,1);
      }
    }
    return -1;
  }
};
```

CATALAN

```
C[n] => FOR(k=0,n-1) C[k] * C[n-1-k]
C[n] => Comb(2*n,n) / (n + 1)
C[n] => 2*(2*n-3)/n * C[n-1]
```

FACTORIAL MODULAR

```
int factMod (int n, int p) {
  int res = 1,i;
  while (n > 1) {
    if ((n/p) & 1) res = (res * (p-1)) % p;
    for (i=n%p; i > 1;i--) res = (res * i) % p;
    n /= p;
  }
  return res % p;
}
```

FINDING THE DEGREE OF THE DIVISOR FACTORIAL

```
int fact_pow ( int n, int k )  {
  int res =  0 ;
  while  ( n )  {
    n /= k ;
    res += n ;
  }
  return res ;
}
```

MINIMAL ENCLOSING CIRCLE

```
double distSqr(P &p1, P &p2){
   return (p1.X-p2.X)*(p1.X-p2.X) + (p1.Y-p2.Y)*(p1.Y-p2.Y);
}
bool contain(circle c,P p){
   return distSqr(c.p,p)<= c.r*c.r;
}
circle findCircle(P a,P b){
   P p( real(a+b)/2.0 , imag(a+b)/2.0);
   return circle( p, sqrt(distSqr(a,p)));
}
circle findCircle(P pa,P pb,P pc) {
   double a,b,c,x,y,r,d;
   c = sqrt(distSqr(pa , pb));
   b = sqrt(distSqr(pa , pc));
   a = sqrt(distSqr(pb , pc));
   if (b==0 || c==0 || a*a>= b*b+c*c)
      return findCircle(pb,pc);
   if (b*b >= a*a+c*c)
      return findCircle(pa,pc);
   if (c*c >= a*a+b*b)
      return findCircle(pa,pb);
   d = real(pb-pa)*imag(pc-pa);
   d = 2 * (d - imag(pb-pa)*real(pc-pa));
   x = (imag(pc-pa)*c*c-imag(pb-pa)*b*b)/d;
   y = (real(pb-pa)*b*b-real(pc-pa)*c*c)/d;
   x += real(pa), y += imag(pa);
   r= sqrt(pow(real(pa)-x,2)+ pow(imag(pa)-y,2));
   return circle(P(x,y),r);
}
P points[MAXN], R[3];
circle sed(int n,int nr){
   circle c;
   if(nr == 3) c = findCircle(R[0],R[1],R[2]);
   else if (n == 0 && nr==2) c = findCircle(R[0], R[1]);
   else if(n==1 && nr == 0) c = circle(points[0],0);
   else if(n == 1 && nr == 1) c = findCircle(R[0],points[0]);
   else{
      c = sed(n-1, nr);
      if(!contain(c,points[n-1])){
         R[nr++] = (points[n-1]);
         c = sed(n-1, nr);
      }
   }
   return c;
}
```

INDEX OF PERMUTATION
```
//normalizar a *per a partir del 1
//dif = cant de elementos distintos
ll IndexPermutation(int *per,int n,int dif){
   memset(alpha,0,sizeof(alpha));
   for(int i = 0; i < n; i++) alpha[per[i]]++;
   ll sol = 0, par;
   for(int i = 0; i < n-1; i++){
      for(int j = 1; j < per[i]; j++){
         if(!alpha[j]) continue;
            par = fact[n-i-1];
            for(int k=1;k <= dif;k++)
               par /=fact[alpha[k]-(k==j)];
            sol += par;
      }
      --alpha[per[i]];
   }
   return sol+1;
}
```

KTH PERMUTACION
```
ll fact[21]; //factorial
int N; // N grupos
char grupo[22];//caract del grupo
int cantgrupo[22], quitar;
//quitar=1; N=20;
//for(i = 0; i < N; i++) cantgrupo[i]=1,grupo[i]=i+'A';
//for(i = 0; i < N; i++) quitar *= fact[cantgrupo[i]];
//KthPermutacion(1234567889,20);

void KthPermutacion(int k, int quedan){
  if (quedan == 0) return;
  int total = fact[quedan - 1];
  int inicio = 0, fin = 0;
  for (int i = 0; i < N; i++) {
    if (cantgrupo[i] == 0) continue;
    fin += (cantgrupo[i] * total) / quitar;
    if (fin > k) {
      quitar /= cantgrupo[i]--;
      cout << grupo[i];
      KthPermutacion(k - inicio, quedan - 1);
    } else inicio = fin;
  }
}
```

LONGEST INCREASING SUBSEQUENCE
```
int LIS(int n,int *a){
  int i,l, r, c, p[MAXN], b[MAXN], m = 1;
  for (b[0]=0,i=1; i < n; i++){
    if (a[b[m-1]] < a[i]){
      p[i] = b[m-1];
      b[m++] = i;
      continue;
    }
    l = 0, r = m - 1;
    while (l < r){
      c = (l + r) / 2;
      if (a[b[c]] < a[i]) l = c + 1;
      else r = c;
    }
    if ( a[i] < a[b[l]] ) {
      p[i] = (l > 0)? b[l-1] : -1;
      b[l] = i;
    } else p[i] = -1;
  }
  return m;
}
```

METODO DE SIMPSON
```
// a,b: intervalo de integracion
// n = 1000*1000: número de pasos (ya multiplicado por 2)
double Simpson(int n, double a, double b){
      double s = 0;
      double h = (b - a) / n;
      for (int i=0; i<=n; ++i) {
        double x = a + h * i;
        s += f(x) * ((i==0 || i==n) ? 1 : ((i&1)==0) ? 2 : 4);
      }
      return s*(h/3);
}
```

AMOUNT OF SPANNING TREES IN COMPLETE GRAPH
```
T(Kn) = n^(n-2)
```

AMOUNT OF SPANNING TREES IN COMPLETE BIPARTITE GRAPH
```
T(Kp,q) = p^(q-1) * q^(p-1)
```

AMOUNT OF SPANNING TREES IN GRAPH (KIRCHHOFF'S THEOREM)

if vertex i is adjacent to vertex j in G, then Qi,j equals -m, where m is the number of edges between i and j;
Qi,i = degree(i), when counting the degree of a vertex, all loops are excluded.
Then the amount of spanning trees in the graph is equal to the determinant of Q matrix erasing the last row and column.

CONVERSION A POSTFIJA

```
bool delim ( char c ) {
  return c == ' ' ;
}

bool is_op ( char c ) {
  return c == '+' || c == '-' || c == '*' || c == '/' || c ==
'%' ;
}

int priority ( char op ) {
  return op == '+' || op == '-' ? 1 :
       op == '*' || op == '/' || op == '%' ? 2 :  - 1 ;
}

void process_op ( vector < int > & st, char op ) {
  int r = st. back ( ) ;  st. pop_back ( ) ;
  int l = st. back ( ) ;  st. pop_back ( ) ;
  switch ( op ) {
    case '+' :  st. push_back ( l + r ) ;  break ;
    case '-' :  st. push_back ( l - r ) ;  break ;
    case '*' :  st. push_back ( l * r ) ;  break ;
    case '/' :  st. push_back ( l / r ) ;  break ;
    case '%' :  st. push_back ( l % r ) ;  break ;
  }
}

int calc ( string & s ) {
  vector < int > st ;
  vector < char > op ;
  for ( size_t i = 0 ; i < s. length ( ) ; ++ i )
    if ( !delim ( s [ i ] ) )
      if ( s [ i ] == '(' )
        op. push_back ( '(' ) ;
      else if ( s [ i ] == ')' ) {
        while ( op. back () != '(' )
          process_op ( st, op. back ( ) ) ,  op. pop_back ( ) ;
        op. pop_back ( ) ;
      }
      else if ( is_op ( s [ i ] ) ) {
        char curop = s [ i ] ;
        while ( ! op. empty ( ) &&
             priority ( op. back ( ) ) >= priority ( s [ i ] ) )
          process_op ( st, op. back ( ) ) ,  op. pop_back ( ) ;
        op. push_back ( curop ) ;
      }
      else {
        string operand ;
        while ( i < s. length() && isalnum ( s [ i ] ) )
          operand += s [ i ++ ] ;
        -- i ;
        if ( isdigit ( operand [ 0 ] ) )
          st. push_back ( atoi ( operand. c_str ( ) ) ) ;
        else
          st. push_back ( get_variable_val (operand) ) ;
      }
  while ( ! op. empty ( ) )
    process_op ( st, op. back ( ) ) ,  op. pop_back ( ) ;
  return st. back ( ) ;
}
```

GRAY CODE G(n) = n ^ (n >> 1)

```
int rev_g ( int G )  {
  int n =  0 ;
  for  ( ; G ; G >>= 1 ) n ^= G ;
  return n ;
}
```

TODAS LAS MASCARAS S MENORES QUE M QUE CONTIENE SOLAMENTE LOS BITS ACTIVOS EN M

```
void sub(int m){
  int s = m ;
  while  ( s > 0 ) {
      ... You can use the s ...
      s  =  ( s - 1 )  & m ;
  }
}
```

TODAS LAS MASCARAS S MENORES QUE M QUE CONTIENE SOLAMENTE LOS BITS NO ACTIVOS EN M

```
void mask(int m){
  int k = log2(m);
  for(int s = (1<<k)-1; (s &= ~m) >= 0; s--){
    ... You can use the s ...
  }
}
```

PRIME FACTORIZATION

```
void factorization(int n){
  int p[n];
  for(int i = 2; i <= n; ++i) p[i]=i;
  for(int i = 2; i*i <= n; ++i)
   if(p[i]==i)
    for(int j = i*i; j <= n; j+=i) p[j]=i;
  vector<int> d;
  while(n!=1){
    d.push_back(p[n]);
    n/=p[n];
  }
}
```

DISCRETE LOGARITHM

```
//solve a^x = b (mod m) where a and m co-prime calcula x
int powmod ( int a, int b, int p )  {
  int res =  1 ;
  while   ( b )
    if   ( b & 1 ) res =  int  ( res * 1ll * a % p ) ,  --b ;
    else a =  int  ( a * 1ll * a % p ) , b >>=  1 ;
  return res ;
}
int solve ( int a, int b, int m )  {
  vector < pair < int , int >  > :: iterator it;
  int msq =  (int)sqrt(m + 0.0) +  1;
  int msq2 = m / msq +  ( m % msq ? 1 : 0 );
  vector < pair < int , int >  > vals ( msq2 ) ;
  for  ( int i = 1 ; i <= msq2 ;  ++i )
    vals [ i-1 ]  = make_pair ( powmod ( a, i * msq, m ) , i ) ;
  sort ( vals. begin ( ) , vals. end ( ) ) ;
  for  ( int i = 0 ; i <= msq ;  ++i )  {
    int cur = powmod ( a, i, m ) ;
    cur =  ( cur * b )  % m ;
    it = lower_bound ( vals. begin ( ) , vals. end ( ) ,
make_pair ( cur, 0 ) ) ;
    if  ( it != vals.end ( )  && it->first == cur )
      return it->second * msq - i;
  }
  return  - 1 ;
}
```

PRIMITIVE ROOTS

```
solve a^x = b (mod m) where b and m co-prime calcula a
n is either an odd prime power or twice a prime power, and in
cases , n = 1, n = 2, n = 4.
int PrimitiveRoot ( int p )  {
  vector < int > Fact ;
  int Phi = p - 1, n = Phi ;
  for  ( int i = 2 ; i * i <= n ;  ++i )
    if  ( n % i == 0 )  {
      Fact. push_back  ( i ) ;
      while  ( n % i == 0 ) n /= i ;
    }
  if  ( n > 1 ) Fact . push_back  ( n ) ;

  for  ( int res = 2 ; res <= p ;  ++res )  {
    bool OK =  true ;
    for  ( int i = 0 ; i < Fact. size ( )  && OK ;  ++ i )
      OK &= powmod ( res, Phi / Fact [ i ] , p )  !=  1 ;
    if  ( OK )   return res ;
  }
  return  - 1 ;
}
```

DISCRETE ROOT EXTRACT

```
x^k = a (mod n) solve x where n is prime
int gcd ( int a, int b ) {
  return a ? gcd ( b % a, a )  : b ;
}
void DiscreteRootExtract(int n, int k, int a){
  if  ( a == 0 )  {
      puts  ( "1 \n 0 " ) ;
      return  ;
    }
    int G = PrimitiveRoot ( n ) ;
    int sq =  ( int )  sqrt  ( n + .0 )  +  1 ;
    vector < pair < int , int >  > dec ( sq ) ;
```

```
    vector < pair < int , int >  > :: iterator IT;
    for  ( int I = 1 ; I <= sq ;  ++I )
      dec [ I - 1 ]  = make_pair ( powmod ( G, int  ( I * sq *
1ll * k %  ( n - 1 ) ) , n ) , I ) ;
    sort ( dec. begin ( ) , dec. end ( ) ) ;
    int any_ans =  - 1 ;
    for  ( int I = 0 ; I < sq ;  ++I )  {
      int My =  int  ( powmod ( G, int  ( I * 1ll * k %  ( n - 1
) ) , n )  * 1ll * a % n ) ;
      IT = lower_bound ( dec. begin ( ) , dec. end ( ) ,
make_pair ( My, 0 ) ) ;
      if  ( IT != dec.end ( )  && IT ->first == My )  {
        any_ans = IT->second * sq - I ;
        break ;
      }
    }
    if  ( any_ans ==  -1 )  {
      puts  ( "0" ) ;
      return ;
    }
    int delta =  ( n - 1 )  / gcd ( k, n - 1 ) ;
    vector < int > ans ;
    for(int cur = any_ans % delta ; cur < n - 1 ; cur += delta )
      ans. push_back  ( powmod ( G, cur, n ) ) ;
    sort ( ans. begin ( ) , ans. end ( ) ) ;
    printf  ( "%d\n" , ans. size ( ) ) ;
    for  ( int I = 0 ; I < ans. size ( ) ;  ++I )
      printf  ( "%d" , ans [ I ] ) ;
}
```

ABI 2D

```
void update(int f, int c, int k) {
  for(int i = f;i<n + 10;i+=(i&-i))
    for(int j = c;j<n + 10;j+=(j&-j))
      T[i][j] += k;
}

int query(int f, int c) {
  int sol = 0;
  for(int i = f;i;i-=(i&-i))
    for(int j = c;j;j-=(j&-j))
      sol += T[i][j];
  return sol;
}
```