Teambook – Que Perdió

Índice

Estructuras de Datos

Sparse Table

```
1.  const int tam = 1000010;
2.  const int logTam = 21;
3.
4.  int logTable[tam];
5.  int sparseTable[logTam][tam];
6.  int a[tam], n; // a[] es el arreglo original. n es el tamaño del arreglo.
7.
8.  void makeSparseTable()
```

```
{
  logTable[0] = logTable[1] = 1;

  for (int i = 2; i <= n; i++)
  {
    logTable[i] = logTable[ i >> 1 ] + 1;
  }

  for (int i = 0; i < n; i++)
  {
    sparseTable[0][i] = i;
  }

  for (int k = 1; (1 << k) < n; k++)
  {
    for (int i = 0; i + (1 << k) <= n; i++)
    {
      int x = sparseTable[k-1][i];
      int y = sparseTable[k-1][i - (1 << (k-1))];
      sparseTable[k][i] = a[x] <= a[y] ? x : y;
    }
  }
}

int queryST(int i, int j)
{
  int k = logTable[j - i];
  int x = sparseTable[k][i];
  int y = sparseTable[k][j - (1 << k) + 1];
  return a[x] <= a[y] ? x : y;
}
```

Segment Tree Iterativo

```
const int N = 1e5;  // limit for array size
int n;  // array size
int t[2 * N];

void build() {  // build the tree
  for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1];
}

void modify(int p, int value) {  // set value at position p
  for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1];
}

int query(int l, int r) {  // sum on interval [l, r)
  int res = 0;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) res += t[l++];
    if (r&1) res += t[--r];
  }
  return res;
}
```

Fenwick Tree

```
#define clr(a,h)    memset(a,(h),sizeof(a))

int BIT[tam];

void update(int pos, int val)
{
  pos++;
  while(pos < 200010)
  {
    BIT[pos] += val;
    pos += (pos & -pos);
  }
}

int query(int pos)
{
  pos++;
  int res = 0;
  while(pos > 0)
  {
    res += BIT[pos];
    pos -= (pos & -pos);
  }
  return res;
}

int main()
{
  clr(BIT,0);
  for(int i = n - 1; i >=0; i--)
  {
```

```
    inv +=query(a[i]);
    update(a[i],1);
  }
}
```

Fenwick Tree 2D

```
#include <iostream>

using namespace std;

const int tam = 1000;

int BIT[tam][tam];
int n, m;

void update(int row, int col, int val)
{
  row++; col++;
  for (int i = row; i <= n; i += (i & -i))
  {
    for (int j = col; j <= m; j += (j & -j))
    {
      BIT[i][j] += val;
    }
  }
}

int query(int row, int col)
{
  int res = 0;
  row++; col++;
  for (int i = row; i > 0; i -= (i & -i))
  {
    for (int j = col; j > 0; j -= (j & -j))
    {
      res += BIT[i][j];
    }
  }
  return res;
}
```

Segment Tree persistente

```
struct node{
  ptr iz;
  ptr der;
  int val;   //0.o
  int numero;
  node(){
    numero=-1;
    val=0;
  }
};
node nodos[tam];int cnodos=0;
node NUL;
ptr getnode()
{
  nodos[cnodos].iz=nodos[cnodos].der=&NUL;
  //if (cnodos>=tam)
    //tle(); no
  return &nodos[cnodos++];
}
void clr(){
  NUL.iz=NUL.der=&NUL;
}

void insertar(ptr nuevo,ptr antnodo,int iz,int der,int pos,int numero)
{
  if (iz==der)
  {

    (*nuevo).val=(*antnodo).val+1;
    (*nuevo).numero=numero;
    return;
  }
  int mid=(iz+der)/2;
  if (pos<=mid)
  {
    (*nuevo).der=(*antnodo).der;
    (*nuevo).iz=getnode();
    insertar((*nuevo).iz,(*antnodo).iz,iz,mid,pos,numero);
  }
  else
  {
    (*nuevo).iz=(*antnodo).iz;
    (*nuevo).der=getnode();
    insertar((*nuevo).der,(*antnodo).der,mid+1,der,pos,numero);
  }
```

```
    (*nuevo).val=(*(*nuevo).iz).val+(*(*nuevo).der).val;
}

int query(ptr nodoa,ptr nodob,ptr resta1,ptr resta2,int kth,int iz,int de
r)
{
   if (iz==der)
   {
   return iz;// numero
   }
   int valiz=(*(*nodoa).iz).val+(*(*nodob).iz).val-(*(*resta1).iz).val-
(*(*resta2).iz).val;
   int mid=(iz+der)/2;
   if (kth>valiz)
   {
      query((*nodoa).der,(*nodob).der,(*resta1).der,(*resta2).der,kth-
valiz,mid+1,der);//kth-valiz ***
   }
   else
   {
      query((*nodoa).iz,(*nodob).iz,(*resta1).iz,(*resta2).iz,kth,iz,mid);
   }
}
```

Treap

```
#include <bits/stdc++.h>

using namespace std;
struct node{
   int key, pri, siz;
   node *l, *r;
};
typedef node* pnode;
int sz(pnode t)
{
   return (t?t->siz:0);
}
void up_sz(pnode t)
{
   if(t) t->siz = sz(t->l) + 1 + sz(t->r);
}
void split(pnode t, pnode &l, pnode &r, int val)
{
   if(!t) r = l = NULL;
   else if(t->key <= val) split(t->r, t->r, r, val), l = t;
   else split(t->l, l, t->r, val), r = l;
   up_sz(t);
}
void merge(pnode &t, pnode l, pnode r)
{
   if(!l || !r) t=(l?l:r);
   else if(l->pri >= r->pri) merge(l->r, l->r, r), t = l;
   else merge(r->l, l, r->l),t=r;
   up_sz(t);
}
void insert(pnode &t, pnode it)
{
   if(!t) t = it;
   else if(it->pri > t->pri) split(t, it->l, it->r, it->key), t = it;
   else insert((t->key <= it->key?t->l:t->r), it);
   up_sz(t);
}
void erase(pnode &t, int val)
{
   if(!t) return;
   else if(t->key == val) {pnode temp = t; merge(t, t->l, t-
>r); free(temp);}
   else erase((t->key <= val?t->l:t->r),val);
   up_sz(t);
}
pnode init(int val)
{
   pnode ret = (pnode)malloc(sizeof(node));
   ret->key = val, ret->siz = 1, ret -> pri = rand(), ret->l = ret-
>r = NULL;
   return ret;
}
```

HLD

```
#include <bits/stdc++.h>
#define clr(a,h)          memset(a, (h), sizeof(a))

using namespace std;

/*
Cambiar:
```

```
8.      - Valor que se guarda en cada nodo en HLD
9.      - Operador en Segment Tree
10.     - Operador en queryUp y query
11. Inicializar:
12.     - root para LCA
13.     - chainNo = 0 para HLD
14.     - Llenar chainHead[] con -1 para HLD
15.     - actPosInBase = 0 para Segment Tree
16. */
17.
18. const int tam = 10010;
19. const int Log2Tam = 15;
20.
21. int nodeCost[tam];
22.
23. vi g[tam];
24. //HLD
25. int chainNo, chainPos[tam], chainIdx[tam], chainHead[ta
    m], chainSize[tam];
26. //DFS
27. int dp[tam][Log2Tam], depth[tam], subTreeSize[tam];
28. int n, m, root;
29. //SegTree
30. int segTree[tam*4], posInBase[tam], baseArray[tam], act
    PosInBase;
31. int segTreeQ[tam*4];
32.
33. void DFS(int v, int p, int d)
34. {
35.     dp[v][0] = p;
36.     depth[v] = d;
37.     subTreeSize[v] = 1;
38.     for (int i = 0; i < g[v].size(); i++)
39.     {
40.         int u = g[v][i];
41.         if (u == p) continue;
42.         DFS(u, v, d+1);
43.         subTreeSize[v] += subTreeSize[u];
44.     }
45. }
46.
47. void HLD(int v, int p)
48. {
49.     if (chainHead[chainNo] == -
    1) chainHead[chainNo] = v;
50.     chainIdx[v] = chainNo;
51.     chainPos[v] = chainSize[chainNo];
52.     chainSize[chainNo]++;
53.     posInBase[v] = actPosInBase;
54.     baseArray[actPosInBase++] = nodeCost[v]; // Aqui gu
    ardamos lo que queremos trabajar
55.                                                 // con el
    Segment Tree
56.
57.     int bestChild = -1, maxSize = -1;
58.     for (int i = 0; i < g[v].size(); i++)
59.     {
60.         int u = g[v][i];
61.         if (u == p) continue;
62.         if (subTreeSize[u] > maxSize)
63.         {
64.             maxSize = subTreeSize[u];
65.             bestChild = u;
66.         }
67.     }
68.
69.     if (bestChild != -1) HLD(bestChild, v);
70.
71.     for (int i = 0; i < g[v].size(); i++)
72.     {
73.         int u = g[v][i];
74.         if (u == p || u == bestChild) continue;
75.         chainNo++;
76.         HLD(u, v);
77.     }
78. }
79.
80. void initLCA()
81. {
82.     clr(dp, -1);
83.     DFS(root, -1, 0);
84.     return;
85.     for (int pot = 1; pot < Log2Tam; pot++)
86.     {
87.         for (int v = 0; v < n; v++)
88.         {
89.             if (dp[v][pot-1] == -1) continue;
```

```
            dp[v][pot] = dp[ dp[v][pot-1] ][pot-1];
        }
    }
}

int LCA(int a, int b)
{
    // b siempre debajo o al mismo nivel que a
    if (depth[a] > depth[b]) swap(a, b);

    int diff = depth[b] - depth[a];
    for (int pot = Log2Tam-1; pot >= 0; pot--)
    {
        if ( ( diff & (1 << pot) ) )
        {
            b = dp[b][pot];
        }
    }
    if (a == b) return a;
    for (int pot = Log2Tam-1; pot >= 0; pot--)
    {
        if (dp[a][pot] != dp[b][pot])
        {
            a = dp[a][pot];
            b = dp[b][pot];
        }
    }
    int lca = dp[a][0];
    return lca;
}

void initSegTree(int b, int e, int nodo)
{
    int L = 2 * nodo + 1, R = L + 1, mid = (b + e) / 2
;
    if (b == e)
    {
        segTree[nodo] = baseArray[b];
        return;
    }
    initSegTree(b, mid, L);
    initSegTree(mid+1, e, R);

    // Funcion del Segment Tree que queremos hacer sob
re el arbol
    segTree[nodo] = segTree[L] > segTree[R] ? segTree[
L] : segTree[R];
}

void updateSegTree(int b, int e, int nodo, int pos, in
t val)
{
    int L = 2 * nodo + 1, R = L + 1, mid = (b + e) / 2
;
    if (b == e)
    {
        segTree[nodo] = val;
        return;
    }
    if (pos <= mid) updateSegTree(b, mid, L, pos, val)
;
    else updateSegTree(mid+1, e, R, pos, val);

    segTree[nodo] = segTree[L] > segTree[R] ? segTree[
L] : segTree[R];
}

int querySegTree(int b, int e, int nodo, int i, int j)

{
    int L = 2 * nodo + 1, R = L + 1, mid = (b + e) / 2
;
    if (i <= b && e <= j) return segTree[nodo];
    if (j <= mid) return querySegTree(b, mid, L, i, j)
;
    else if (i > mid) return querySegTree(mid+1, e, R,
 i, j);
    else
    {
        int r1 = querySegTree(b, mid, L, i, j);
        int r2 = querySegTree(mid+1, e, R, i, j);
        return r1 > r2 ? r1 : r2;
    }
}

int queryUp(int u, int v)
{
```

```
    int uChain, vChain = chainIdx[v], ans = -1;
    while (true)
    {
        uChain = chainIdx[u];
        if (uChain == vChain)
        {
            if (u == v) break;
            int queryAns = querySegTree(0, actPosInBase-1, 0, posInBase[v]+1, posInBase[u]);
            if (queryAns > ans)
            {
                ans = queryAns;
            }
            break;
        }
        int queryAns = querySegTree(0, actPosInBase-1, 0, posInBase[ chainHead[uChain] ], posInBase[u]);
        if (queryAns > ans) ans = queryAns;
        u = chainHead[uChain];
        u = dp[u][0];
    }
    return ans;
}

int query(int a, int b)
{
    int lca = LCA(a, b);
    int r1, r2;
    r1 = queryUp(a, lca);
    r2 = queryUp(b, lca);
    return r1 > r2 ? r1 : r2;
}
```

Grafos

LCA

```
#include <bits/stdc++.h>
#define clr(a,h)        memset(a, (h), sizeof(a))

using namespace std;

const int tam = 1010; // Max Nodos
const int Log2Tam = (log(tam)/log(2)) + 3;

vector< vi > g;
int dp[tam][Log2Tam], depth[tam];
int n, m, root; // nodos, aristas, raiz del arbol

void initDFS(int v, int p, int d)
{
  dp[v][0] = p; // padre inmediato
  depth[v] = d;
  for (int u : g[v])
  {
    if (u == p) continue;
    initDFS(u, v, d+1);
  }
}

void initLCA()
{
  clr(dp, -1);
  initDFS(root, -1, 0);
  for (int pot = 1; pot < Log2Tam; pot++)
  {
    for (int v = 0; v < n; v++)
    {
      if (dp[v][pot-1] == -1) continue;
      dp[v][pot] = dp[ dp[v][pot-1] ][pot-1];
    }
  }
}

int LCA(int a, int b)
{
  // b siempre debajo o al mismo nivel que a
  if (depth[a] > depth[b]) swap(a, b);

  int diff = depth[b] - depth[a];
  for (int pot = Log2Tam-1; pot >= 0; pot--)
  {
    if ( ( diff & (1 << pot) ) )
    {
      b = dp[b][pot];
    }
  }
  if (a == b) return a;
```

```
    for (int pot = Log2Tam-1; pot >= 0; pot--)
    {
      if (dp[a][pot] != dp[b][pot])
      {
        a = dp[a][pot];
        b = dp[b][pot];
      }
    }
    int lca = dp[a][0];
    return lca;
}

int main()
{
    /*
      Iniciar el arbol en el grafo g
      Asignar la raiz en la variable root

      Llamar initLCA() luego de crear el arbol
      LCA(a, b) devuelve el LCA de los nodos a y b
    */
    return 0;
}
```

Centroid Decomposition

```
int n;
vector<int> grafo[tam];
int hijos[tam],padre[tam];bool marcado[tam];
void dfshijos(int num,int pad)
{
    //<<num<<endl;
    hijos[num]=1;
    padre[num]=pad;
    int num2;
    for (int i = 0; i < grafo[num].size(); ++i)
    {
      num2=grafo[num][i];
      if (num2==pad|| marcado[num2]==true)continue;
      dfshijos(num2,num);
      hijos[num]+=hijos[num2];
    }
}
void operaciones(int num,int compsize);
void descomponer(int num)
{
    dfshijos(num,num);
    queue<int> cola;
    cola.push(num);
    int maxx,minn,centroide,num2,auxnum;
    minn=hijos[num];centroide=num;
    int compsize=0;
    while(!cola.empty())
    {   compsize++;
      auxnum=cola.front();cola.pop();
      maxx=hijos[num]-hijos[auxnum];
      for (int i = 0; i < grafo[auxnum].size(); ++i)
      {
        num2=grafo[auxnum][i];
        if (padre[auxnum]==num2|| marcado[num2]==1)continue;
        maxx=max(maxx,hijos[num2]);
        cola.push(num2);
      }
      if (minn>maxx)
      {
        minn=maxx;
        centroide=auxnum;
      }
    }
    //cout<<centroide<<" "<<compsize<<endl; para revisar
    operaciones(centroide,compsize);
    marcado[centroide]=1;
    for (int i = 0; i < grafo[centroide].size(); ++i)
    {
      num2=grafo[centroide][i];
      if (marcado[num2]==1)continue;
      descomponer(num2);
    }
}
// memset(marcado,false,sizeof marcado);
```

MinCost-MaxFlow

```
struct Edge
{
    int from, to, capacity, cost;
};

```

```
vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<int> m(n, 2);
    deque<int> q;
    q.push_back(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop_front();
        m[u] = 0;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (m[v] == 2) {
                    m[v] = 1;
                    q.push_back(v);
                } else if (m[v] == 0) {
                    m[v] = 1;
                    q.push_front(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

DP

SOS DP

```
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
    dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
    for(int i = 0;i < N; ++i){
        if(mask & (1<<i))
            dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
        else
```

```
            dp[mask][i] = dp[mask][i-1];
        }
        F[mask] = dp[mask][N-1];
    }
    //memory optimized, super easy to code.
    for(int i = 0; i<(1<<N); ++i)
        F[i] = A[i];
    for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
        if(mask & (1<<i))
            F[mask] += F[mask^(1<<i)];
    }
```

Geometría

Convex Hull - Graham's Scan

```
struct point
{
    double x,y;
    point() {x=y=0;}
    point(double X, double Y): x(X), y(Y) {}
};
double dist(point a, point b)
{
    return hypot(a.x-b.x,a.y-b.y);
}
point tovec(point a, point b)
{
    return point(b.x-a.x, b.y-a.y);
}
double cross(point a, point b)
{
    return a.x*b.y - a.y*b.x;
}
bool coolinear(point a, point b, point c)
{
    return fabs(cross(tovec(a,b), tovec(a,c))) < EPS;
}
bool ccw(point a, point b, point c)
{
    return cross(tovec(a,b),tovec(a,c)) >= 0;
}
point pivot;
bool anComp(point a, point b)
{
    if(coolinear(pivot, a, b))
        return dist(pivot, a) < dist(pivot, b);
    a = tovec(pivot, a), b = tovec(pivot, b);
    return atan2(a.y,a.x) < atan2(b.y,b.x);
}
vector<point> ch(vector<point> p)
{
    int n = p.size(), i , j;
    if(n<=3)
    {
        if(p[0].x!=p[n-1].x||p[0].y!=p[n-1].y) p.pb(p[0]);
        return p;
    }
    int in=0;
    for(int i = 0; i< n; i++)
        if(p[i].y < p[in].y || p[i].x < p[in].x && p[i].y == p[in].y)
            in = i;
    pivot = p[in];
    for(int i = 0; i<n;i++)
    {
        cout<<p[i].x<<' '<<p[i].y<<endl;
    }
    cout<<endl;
    swap(p[0],p[in]);
    cout<<pivot.x<<'#'<<pivot.y<<endl;
    sort(++p.begin(), p.end(),anComp);
    for(int i = 0; i<n;i++)
    {
        cout<<p[i].x<<' '<<p[i].y<<endl;
    }
    cout<<endl;
    i = 2;
    vector<point> s = {p[n-1], p[0], p[1]};
    while(i < n && s.size()>0)
    {
        j = s.size();
        if(ccw(s[j-2],s[j-1],p[i]) || j == 2)
        {
            s.pb(p[i]);
            i++;
        }
        else
            s.pop_back();
```

```
    }
    return s;
}
```

Convex Hull – Monotone Chain

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,b) for(int i = a; i <= b; ++i)
#define invrep(i,b,a) for(int i = b; i >= a; --i)
typedef long long int ll;
// ------------------------------------------------
// Convex Hull: Andrew's Montone Chain Algorithm
// ------------------------------------------------

struct Point {
    ll x, y;
    bool operator<(const Point& p) {
        return x < p.x || (x == p.x && y < p.y);
    }
};

ll cross(Point& a, Point& b, Point& c) {
    ll dx0 = b.x - a.x, dy0 = b.y - a.y;
    ll dx1 = c.x - a.x, dy1 = c.y - a.y;
    return dx0 * dy1 - dx1 * dy0;
}

vector<Point> upper_hull(vector<Point>& P) {
    // sort points lexicographically
    int n = P.size(), k = 0;
    sort(P.begin(), P.end());

    // build upper hull
    vector<Point> uh(n);
    invrep (i, n-1, 0) {
        while (k >= 2 && cross(uh[k-2], uh[k-1], P[i]) <= 0) k--;
        uh[k++] = P[i];
    }
    uh.resize(k);
    return uh;
}

vector<Point> lower_hull(vector<Point>& P) {
    // sort points lexicographically
    int n = P.size(), k = 0;
    sort(P.begin(), P.end());

    // collect lower hull
    vector<Point> lh(n);
    rep (i, 0, n-1) {
        while (k >= 2 && cross(lh[k-2], lh[k-1], P[i]) <= 0) k--;
        lh[k++] = P[i];
    }
    lh.resize(k);
    return lh;
}

vector<Point> convex_hull(vector<Point>& P) {
    int n = P.size(), k = 0;

    // set initial capacity
    vector<Point> H(2*n);

    // Sort points lexicographically
    sort(P.begin(), P.end());

    // Build lower hull
    for (int i = 0; i < n; ++i) {
        while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
        H[k++] = P[i];
    }

    // Build upper hull
    for (int i = n-2, t = k+1; i >= 0; i--) {
        while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
        H[k++] = P[i];
    }

    // remove extra space
    H.resize(k-1);
    return H;
}
```

Puntos y Líneas

```
#include <bits/stdc++.h>
#define EPS         1e-9
```

```
struct line {double a,b,c;}; // ax + by + c = 0
bool areParallel(line a, line b)
{
    return((fabs(a.a-b.a)<EPS)&&(fabs(a.b-b.b)<EPS));
}
bool areSame(line a, line b)
{
    return areParallel(a,b)&&(fabs(a.c-b.c)<EPS);
}
struct point
{
    double x,y;
    point() {x=y=0;}
    point(double _x, double _y) : x(_x), y(_y) {}
    point operator+(point a) const
    {
        a.x+=x;
        a.y+=y;
        return a;
    }
};
double dist(point a, point b)
{
    return hypot(a.x-b.x,a.y-b.y);
}
void toline(point a, point b, line &l) //dados dos puntos
{
    if(fabs(a.x-b.x)<EPS)
        {l.a = 1, l.b = 0, l.c = -a.x; return;}
    l.a = -(a.y - b.y) / (a.x - b.x);
    l.b = 1;
    l.c = -l.a * a.x - a.y;
}
void tolinegr(point a, double gr, line &l) // a linea dado el gradiente
{
    l.a = -gr;
    l.b = 1;
    l.c = a.x * gr - a.y;
}
point tovec(point a, point b)
{
    return point(b.x-a.x,b.y-a.y);
}
}
point translate (point p, point v)
{
    return point(p.x+v.x,p.y+v.y);
}
point scale(point v, double sc)
{
    return point(v.x*sc,v.y*sc);
}
point rotate(point v, double theta) //rotacion antihorario ccw
{
    theta *= acos(-1)/180.0;
    return point(v.x*cos(theta)-v.y*sin(theta),v.x*sin(theta)+ v.y*cos(theta));
}
bool areIntersect(line l1, line l2, point &p) //interseccion de lineas
{
    if(areParallel(l1,l2)) return false;
    p.x = (-l1.c*l2.b + l2.c*l1.b) / (l1.a*l2.b-l2.a*l1.b);
    if(fabs(l1.b) > EPS) p.y = -(l1.a*p.x + l1.c);
    else
        p.y = -(l2.a*p.x +l2.c);
    return true;
}
point clos(point a, line l, line &pe) //closest point in a line and perpendicular line from a
{
    if(fabs(l.a) < EPS)
    {
        pe.a = 1, pe.b = 0, pe.c = -a.x;
        return point(a.x,-l.c);
    }
    if(fabs(l.b) < EPS)
    {
        pe.a = 0, pe.b = 1, pe.c = -a.y;
        return point(-l.c,a.y);
    }
    tolinegr(a, 1/(l.a),pe);
    areIntersect(l,pe,a);
    return a;
}
```

```
point reflexion(point p, point a, point b) // del punto p a linea ab
{
    line l,li;
    toline(a,b,li);
    point p1 = clos(p,li,l);
    p1 = p1+(tovec(p,p1));
    return p1;
}
double norm_sq(point a)
{
    return a.x * a.x + a.y * a.y;
}
double dot(point a, point b)
{
    return a.x*b.x + b.y*a.y;
}
double angle(point a, point b, point c) //b el del medio
{
    a = tovec(b,a), b = tovec(b,c);
    double res = dot(a,b);
    res = acos(res / (sqrt(norm_sq(a))*sqrt(norm_sq(b))));
    res*= 180.0/acos(-1);
    return res;
}
double cross(point a, point b) //prosucto cruz
{
    return a.x * b.y - a.y * b.x;
}
bool left(point a, point b, point c) //ccw
{
    c = tovec(b,c);
    a = tovec(b,a);
    return (cross(c,a)>0.0);
}
bool coolinear(point a, point b, point c)
{

    c = tovec(b,c);
    a = tovec(b,a);
    return (fabs(cross(c,a))<EPS);
}
```

```
double distToLine(point p, point a, point b, point &c) //con producto punto halla el punto
{
    //c = a + u*ab
    point ab = tovec(a,b), ap = tovec(a,p);
    double u = dot(ab,ap) / norm_sq(ab);
    c = translate(a, scale(ab,u));
    return dist(p,c);
}
double distToline1(point p, point a, point b) //con producto cruz solo distancia
{
    point ap = tovec(a,p), ab = tovec(a,b);
    return fabs(cross(ab,ap)/hypot(ab.x,ab.y));
}
```

Poligonos

```
struct point{
    double x,y;
    point(){x=y=0;}
    point(double X, double Y): x(X), y(Y) {}
    point operator+(point a) const
    {
        a.x+=x;
        a.y+=y;
        return a;
    }
    bool operator<(point a) const
    {
        return (a.x == x? a.y < y : a.x < x);
    }
};
double dist(point a, point b)
{
    return hypot(a.x-b.x, a.y-b.y);
}
point tovec(point a, point b)
{
    return point(b.x-a.x,b.y-a.y);
}
double norm(point a)
{
    return hypot(a.x,a.y);
}
```

```
double dot(point a, point b)
{
    return a.x*b.x + a.y*b.y;
}
double cross(point a, point b)
{
    return a.x*b.y - a.y*b.x;
}
bool ccw(point a, point b, point c)
{
    return cross(tovec(a,b),tovec(a,c)) >= 0; //depende si se acepta colinear o no
}
double an(point a, point b, point c)
{
    a = tovec(b,a), b = tovec(b,c);
    return acos(dot(a,b)/(norm(a)*norm(b)));
}
double perimeter(const vector<point> &p)
{
    double result = 0.0;
    for(int i = 0; i<p.size()-1; i++)
    {
        result += dist(p[i],p[i+1]);
    }
    return result;
}
double area(const vector<point> &p)
{
    double result = 0.0;
    for(int i=0;i<p.size()-1;i++)
    {
        result += p[i].x*p[i+1].y - p[i].y*p[i+1].x;;
    }
    return fabs(result)/2.0;
}
point lineIntersectSeg(point p, point q, point A, point B)
{
    double a = B.y - A.y;
    double b = A.x - B.x;
    double c = B.x * A.y - A.x * B.y;
    double u = fabs(a * p.x + b * p.y + c);
    double v = fabs(a * q.x + b * q.y + c);
    return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) / (u+v));
}
vector<point> cutPolygon(point a, point b, const vector<point> &Q)
{
    vector<point> P;
    for (int i = 0; i < (int)Q.size(); i++) {
        double left1 = cross(tovec(a, b), tovec(a, Q[i])), left2 = 0;
        if (i != (int)Q.size()-1) left2 = cross(tovec(a, b), tovec(a, Q[i+1]));
        if (left1 > -EPS) P.push_back(Q[i]); // Q[i] is on the left of ab ; left1 < EPS para la derecha
        if (left1 * left2 < -EPS) // edge (Q[i], Q[i+1]) crosses line ab
            P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
    }
    if (!P.empty() && (P.back().x != P.front().x || P.back().y != P.front().y))
        P.push_back(P.front()); // make P's first point = P's last point
    return P;
}
bool isConvex(const vector<point> &p)
{
    int sz = p.size();
    if(sz<=3) return false;
    bool left = ccw(p[0],p[1],p[2]);
    cout<<left<<endl;
    for(int i = 1; i < sz - 1; i++)
    {
        cout<<i<<' '<<ccw(p[i],p[i+1],p[((i+2)==sz)? 1:i+2])<<endl;
        if(ccw(p[i],p[i+1],p[((i+2)==sz)? 1:i+2])!=left)
            return false;
    }
    return true;
}
bool isIn(const vector<point> &p, point a)
{
```

```
    double ang = 0;
    int sz = p.size();
    if(sz == 0) return false;
    for(int i = 0; i<sz-1;i++)
    {
        if(ccw(a,p[i],p[i+1]))
            ang += an(p[i],a,p[i+1]);
        else
            ang -= an(p[i],a,p[i+1]);
    }
    cout<<ang<<endl;
    return fabs(ang - 2.0*PI) < EPS;
}
```

Triangulos y Circulos

```
struct point
{
    double x,y;
    point() {x=0.0; y = 0.0;}
    point(int _x, int _y) : x(_x), y(_y) {}
    point operator+(point b) const
    {
        b.x += x;
        b.y+=y;
        return b;
    }
};
struct line
{
    double a,b,c;
};
double dist(point a, point b)
{
    return hypot(fabs(a.x-b.x),fabs(a.y-b.y));
}
point tovec(point a, point b)
{
    return point(b.x-a.x,b.y-a.y);
}
point translate(point a, point b)
{
    a= a+b;
    return a;
}
point scale(point a, double s)
{
    a.x*=s;
    a.y*= s;
    return a;
}
void pointsToLine(point a, point b, line &l) //linea da
dos 2 puntos
{
    if(fabs(a.x-b.x)<EPS)
    {
        l.a = 1, l.b = 0, l.c = -a.x;
    }
    else
    {
        l.a = -(a.y-
b.y) / (a.x - b.y), l.b = 1, l.c = -l.a * a.x - a.y;
    }
}
double rInCircle(double ab, double bc, double ca)
{
    double s = (ab+bc+ca)/2;
    return sqrt(s*(s-ab)*(s-bc)*(s-ca));
}
double rIncircle(point a, point b, point c)
{
    return rInCircle(dist(a,b),dist(b,c),dist(c,a));
}
bool areParallel(line a, line b)
{
    return (fabs(a.a-b.a)<EPS)&&(a.b == b.b);
}
bool areIntersect(line a, line b, point &c)
{
    if(areParallel(a,b)) return false;
    c.x = (b.c*a.b-a.c*b.b) / (a.a*b.b-b.a*a.b);
    if(a.b == 0.0) c.y = -(b.b*c.x + b.c);
    else           c.y = -(a.b*c.x + a.c);
    return true;
}
double areaTri1(double a, double b, double c) //heron
{
    double s = (a+b+c)/2;
    return sqrt(s*(s-a)*(s-b)*(s-c));
```

```
}
double areaTri(point a, point b, point c)
{
    return areaTri1(dist(a,b),dist(b,c),dist(a,c));
}
line perp(line a, point p) //perpendicular
{
    line res;
    if(a.b=0)
    {
        res.a = 0, res.b = 1, res.c = -p.y;
    }
    else
        if(fabs(a.a)<EPS)
        {
            res.a = 1, res.b = 0, res.c = -p.y;
        }
        else
        {
            res.a = -1.0/a.a, res.b = 1, res.c = -res.a*p.x-p.y;
        }

}
bool circumCircle(point a, point b, point c, point &ctr, double &r) //circuncentro completo
{
    double area = areaTri(a,b,c);
    if(fabs(area)<EPS) return 0;
    line l1, l2;
    pointsToLine(a,b,l2);
    pointsToLine(a,c,l2);
    point p1 = point((a.x+b.x)/2.0,(a.y+b.y)/2.0), p2 = point((a.x+c.x)/2.0,(a.y+c.y)/2.0);
    l1 = perp(l1,p1), l2 = perp(l2,p2);
    areIntersect(l1,l2,ctr);
    r = dist(a,b)*dist(b,c)*dist(a,c)/(4.0*areaTri(a,b,c));
    return true;
}
bool isInCircum(point a, point b, point c, point p) //si esta dentro del circulo circunscrito
{
    double r;
    point ctr;
    if(!circumCircle(a,b,c,ctr,r)) return false;
    return dist(ctr,p) <= r ;
}
bool inCircle(point a, point b, point c, point &ctr) //incentro
{
    double r = rIncircle(a,b,c);
    if(r< EPS) return false;
    line l1,l2;
    point p1;
    double ratio = dist(a,b) / dist(a,c);
    p1 = translate(b, scale(tovec(b,c),ratio/(1+ratio)));
    pointsToLine(a,p1,l1);
    ratio = dist(b,a) / dist(b,c);
    p1 = translate(a, scale(tovec(a,c),ratio/(1+ratio)));
    pointsToLine(b,p1,l2);
    areIntersect(l1,l2,ctr);
    return true;
}
line toLinep(point a, point b, point c) //para mediatriz
{
    line l;
    if(b.x == c.x)
    {
        l.a = 0, l.b = 1 , l.c = -a.y;
    }
    else
        if(b.y == c.y)
        {
            l.a = 1, l.b = 0, l.c = -a.x;
        }
        else
        {
            l.a = 1/((b.y-a.y)/(b.x-a.x)), l.b = 1, l.c = -l.a*a.x-a.y;
        }
        return l;
}
point circun(point a, point b, point c) //circuncentro
```

```
{
    line l1, l2;
    l1 = toLinep(point((a.x+b.x)/2,(a.y+b.y)/2),a,b);

    l2 = toLinep(point((a.x+c.x)/2,(a.y+c.y)/2),a,c);

    areIntersect(l1,l2,a);
    return a;
}
bool circle2PtsRad(point a, point b, double r, point &
c) //dados 2 puntos y un radio
{
    double det = (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-
b.y);
    det = r * r / det - 0.25;
    if(det < 0.0) return false;
    det = sqrt(det);
    c.x = (a.x + b.x) * 0.5 + (b.y-a.y) * det;
    c.y = (a.y + b.y) * 0.5 + (a.x-
b.x) * det;
    return true;
}
```

Matematicas

Catalan

```
ll cata(int n)
{
   ll bino[n+1];
   memset(bino,0ll,sizeof(bino));
   bino[0]=1;
   for(int i=1;i<=2*n;i++)
      for(int j=min(i,n);j>0;j--)
         bino[j]+=bino[j-1];
   return bino[n]/(n+1);
}

int main()
{
   ll scat[26];//super catalan
   scat[0]=scat[1]=1;
   for(int i=2;i<26;i++)
      scat[i]=(3*(2*i-1)*scat[i-1]-(i-2)*scat[i-2])/(i+1);
   ll cat[26];
   for(int i=0;i<26;i++)
      cat[i]=cata(i);
   return 0;
}
```

Euclides Extendido

```
ll gcd(ll a, ll b){return b==0? a:gcd(b,a%b);}

int x, y, d;
void extendedEuclid(int a, int b)//ecuacion diofantica ax + by = 1
{
   if(b==0) {x=1; y=0; d=a; return;}
   extendedEuclid(b,a%b);
   int x1=y;
   y = x-(a/b)*y;
   x=x1;
}
```

Phi de Euler, Criba

```
bitset<100000> bi;
vi primos;
vector<ll> pric;
void criba()
{
   bi.set();
   for(int i=2;i<100000;i++)
      if(bi[i])
      {
         for(int j=i+i;j<100000;j+=i)
            bi[j]=0;
         primos.push_back(i);
         pric.push_back((ll)i*(ll)i);
      }
}
int euler(int n)
{
   int res=n;
   for(int i=0;pric[i]<=n;i++)
   {
      if(n%primos[i]==0)
      {
         res-= res/primos[i];
```

```
            while(n%primos[i]==0) n/=primos[i];
        }
    }
    if(n!=1) res-=res/n;
    return res;
}
```

Criba modificada

```
int phi[2000001]; //eulerphi
void criba()
{
    for(int i=0;i<2000001;i++) phi[i]=i;
    for(int i=2;i<2000001;i++)
        {
            if(phi[i]==i)
                for(int j=i;j<2000001;j+=i)
                    phi[j]-=phi[j]/i;
        }
}
int crib[1000001]; //number of prime factors
void criba()
{
    memset(crib,0,sizeof(crib));
    for(int i=2;i<1000001;i++)
        if(crib[i]==0)
            for(int j=i;j<1000001;j+=i)
                crib[j]++;
}
```

Pollard Rho y Miller-Rabin

```
ll mulmod(ll a, ll b, ll c) //multiplicacion modular binaria
{
    ll x =0, y = a % c;
    while(b > 0)
    {
        if(b % 2 == 1) x = (x + y) % c;
        y = (y * 2) % c;
        b /= 2;
    }
    return x % c;
}
ll gcd(ll a, ll b) {return (b==0? a : gcd(b,a%b));}
ll poll(ll n) //numoro grande p = a*b devuelve a  (a y b primos)
{
    srand (time(NULL));
    ll x = rand()%(n-2) + 2, y = x, c = rand()%(n-1) + 1, d = 1;
    while(d==1)
    {
        x = (mulmod(x, x, n) + c) % n;
        y = (mulmod(y, y, n) + c) % n;
        y = (mulmod(y, y, n) + c) % n;
        d = gcd((x>y?x-y:y-x),n);
        if(d == n) return poll(n);
    }
    return d;
}
ll modpow(ll b, ll e, ll m)
{
    if(e == 0) return 1;
    //return (modpow((b*b) % m, e/2, m) * (e & 1? b : 1)) % m;
    return mulmod(modpow(mulmod(b,b,m), e/2, m), (e & 1? b : 1), m);

}
bool mill(ll n, int k) //k = 10 es seguro
{
    ll p = n - 1;
    int r = 0;
    bool pp;
    while(p % 2 == 0)
    {
        p /= 2;
        r++;
    }
    for(int i =0; i < k; i++)
    {
        ll a = rand()%(n-4) + 2;
        ll exp = modpow(a,p,n);
        if(exp == 1 || exp == n-1) return true;
        pp = false;
        for(int j = 0; j < r; j++)
        {
            exp = mulmod(exp, exp, n);
            if(exp == 1) return false;
            if(exp == n-1) {pp = true; break;}
        }
        if(!pp) return false;
    }
```

```
57.
58.     return true;
59. }
```

Strings

Algoritmo Z

```
1.  typedef vector<ll>     vll;
2.  vector<int> alz(string a)
3.  {
4.     int n = a.size();
5.     vi res(n);
6.     int left=0,right=0;
7.     for(int k=1;k<n;k++)
8.     {
9.        if(right<k)
10.       {
11.          right=left=k;
12.          while(right<n&&a[right]==a[right-left])
13.             right++;
14.          res[k] = right - left;
15.          right--;
16.       }
17.       else
18.       {
19.          int k1 = k - left;
20.          if(res[k1]< right - k + 1)
21.             res[k] = res[k1];
22.          else
23.          {
24.             left=k;
25.             while(right<n&&a[right] == a[right-left])
26.                right++;
27.             res[k] = right - left;
28.             right--;
29.          }
30.       }
31.    }
32.    return res;
33. }
34. int main()
35. {
36.    string a,b;
37.    cin>>a>>b;
38.    vector<int> v = alz(b+'$'+a);
39.    int ta = a.size();
40.    int tb = b.size();
41.    ta = ta + tb + 1;
42.    for(int i=tb+1;i<ta;i++)
43.    {
44.       if(v[i]==tb)
45.          cout<<i-tb-1<<endl;
46.    }
47.    return 0;
48. }
```

Suffix Array, LCP

```
1.  #include <bits/stdc++.h>
2.
3.  using namespace std;
4.
5.  typedef vector<int>     vi;
6.
7.  struct suffix
8.  {
9.      int index;
10.     int rank[2];
11. };
12.
13. int cmp(struct suffix a, struct suffix b)
14. {
15.     return (a.rank[0] == b.rank[0])? (a.rank[1] < b.ran
    k[1] ?1: 0):
16.         (a.rank[0] < b.rank[0] ?1: 0);
17. }
18.
19. vector<int> buildSuffixArray(vi txt, int n)
20. {
21.     struct suffix suffixes[n];
22.
23.     for (int i = 0; i < n; i++)
24.     {
25.         suffixes[i].index = i;
26.         suffixes[i].rank[0] = txt[i];
27.         suffixes[i].rank[1] = ((i+1) < n)? (txt[i + 1])
    : -1;
28.     }
```

```
29.
30.     sort(suffixes, suffixes+n, cmp);
31.
32.     int ind[n];
33.     for (int k = 4; k < 2*n; k = k*2)
34.     {
35.         int rank = 0;
36.         int prev_rank = suffixes[0].rank[0];
37.         suffixes[0].rank[0] = rank;
38.         ind[suffixes[0].index] = 0;
39.
40.         for (int i = 1; i < n; i++)
41.         {
42.             if (suffixes[i].rank[0] == prev_rank &&
43.                     suffixes[i].rank[1] == suffixes[i-
    1].rank[1])
44.             {
45.                 prev_rank = suffixes[i].rank[0];
46.                 suffixes[i].rank[0] = rank;
47.             }
48.             else
49.             {
50.                 prev_rank = suffixes[i].rank[0];
51.                 suffixes[i].rank[0] = ++rank;
52.             }
53.             ind[suffixes[i].index] = i;
54.         }
55.
56.         for (int i = 0; i < n; i++)
57.         {
58.             int nextindex = suffixes[i].index + k/2;
59.             suffixes[i].rank[1] = (nextindex < n)?
60.                                   suffixes[ind[nextindex]
    ].rank[0]: -1;
61.         }
62.
63.         sort(suffixes, suffixes+n, cmp);
64.     }
65.
66.     vector<int>suffixArr;
67.     for (int i = 0; i < n; i++)
68.         suffixArr.push_back(suffixes[i].index);
69.
70.     return suffixArr;
```

```
71. }
72.
73. vector<int> kasai(vi txt, vector<int> suffixArr)
74. {
75.     int n = suffixArr.size();
76.
77.     vector<int> lcp(n, 0);
78.
79.     vector<int> invSuff(n, 0);
80.
81.     for (int i=0; i < n; i++)
82.         invSuff[suffixArr[i]] = i;
83.
84.     int k = 0;
85.
86.     for (int i=0; i<n; i++)
87.     {
88.         if (invSuff[i] == n-1)
89.         {
90.             k = 0;
91.             continue;
92.         }
93.
94.         int j = suffixArr[invSuff[i]+1];
95.
96.         while (i+k<n && j+k<n && txt[i+k]==txt[j+k])
97.             k++;
98.
99.         lcp[invSuff[i]] = k;
100.
101.         if (k>0)
102.             k--;
103.     }
104.
105.     return lcp;
106. }
107.
108. int main()
109. {
110.   vi txt; // vector para el que sacamos Suffix Array
111.   vi auffixArr = buildSuffixArray(txt, txt.size());
112.   //vi lcp = kasai(txt, suffixArr);
113.   // Para sacar LCP debemos haber separado cada palabr
    a con un separador
```

```
  // menor que cualquier elemento del alfabeto
  return 0;
}
```

Tecnicas

Algoritmo de Mo

```
#include <bits/stdc++.h>
#define mp                make_pair
#define F first
#define S second

using namespace std;

const int INF = int(1e9 + 7);
typedef long long       ll;
typedef pair<ll, ll>    ii;

pair<ii, int> q[200010];
ll v[200010], anss[200010];
int block = 450;

bool cmp(pair<ii, int> x, pair<ii, int> y)
{
    ii a = x.F, b = y.F;
    int A = a.F / block, B = b.F / block;
    if ( A == B )
    {
        if (A & 1) return a.S < b.S;
        else return a.S > b.S;
    }
    return A < B;
}

ll cont[1000010];
ll ans;

void add(int x)
{
    ll val = cont[x]*cont[x]*x;
    ans -= val;
    cont[x]++;
    val = cont[x]*cont[x]*x;
    ans += val;
}

void del(int x)
{
    ll val = cont[x]*cont[x]*x;
    ans -= val;
    cont[x]--;
    val = cont[x]*cont[x]*x;
    ans += val;
}

int main()
{
    std::ios::sync_with_stdio(false); cin.tie(0);
    //freopen("","r",stdin);
    //freopen("","w",stdout);
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < n; i++)
    {
        cin >> v[i];
    }
    for (int i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;
        a--; b--;
        q[i] = mp( mp(a, b), i );
    }
    sort(q, q + m, cmp);
    int L = 0, R = -1;
    ans = 0;
    for (int i = 0; i < m; i++)
    {
        int l = q[i].F.F, r = q[i].F.S;
        while (R < r)
        {
            R++;
            add(v[R]);
        }
        while (l < L)
        {
            L--;
```

```
            add(v[L]);
        }
        while (R > r)
        {
            del(v[R]);
            R--;
        }
        while (l > L)
        {
            del(v[L]);
            L++;
        }
        anss[ q[i].S ] = ans;
    }
    for (int i = 0; i < m; i++)
    {
        cout << anss[i] << '\n';
    }
    return 0;
}
// PLUS ULTRA!
```

Temas Raros

FFT

```
typedef complex<double> cd;
double mul;
vector<cd> fft(vector<cd> a)
{
  int n = a.size();
  if(n==1) return a;
  double theta = 2.0*acos(-1)/(double)n;
  cd w = 1,
  wn = cd(cos(theta), mul*sin(theta));
  vector<cd> y(n), aEven(n/2), aOdd(n/2);
  for(int i=0;i<n/2;i++)
  {
    aEven[i] = a[2*i];
    aOdd[i] = a[2*i+1];
  }
  aEven = fft(aEven);
  aOdd = fft(aOdd);
  for(int i=0;i<n/2;i++)
  {
    a[i] = aEven[i] + w*aOdd[i];
    a[n/2+i] = aEven[i] - w*aOdd[i];
    w*=wn;
  }
  return a;
}
vector<cd> ffmul(vector<cd> a, vector<cd> b) //tamaño de a debe ser >= al tamaño de b
{
  int n = 2*a.size();
  while(__builtin_popcount(n)>1) n+= (n&-n);
  while(a.size()<n) a.push_back(0);
  while(b.size()<n) b.push_back(0);
  mul = 1;
  a = fft(a);
  b = fft(b);
  for(int i=0;i<n;i++)
    a[i] = a[i]*b[i];
  mul = -1;
  a = fft(a);
  for(int i=0;i<n;i++)
    a[i]/=n;
  return a;
}
```

Shortcuts

Comparador – Java

```
import java.util.Comparator;
import java.util.Arrays;

public class Employee implements Comparable<Employee> {

    private int id;
    private String name;
    private int age;
    private long salary;

    public int getId() {
```

```
        return id;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public long getSalary() {
        return salary;
    }

    public Employee(int id, String name, int age, int s
alary) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    @Override
    public int compareTo(Employee emp) {
        //let's sort the employee based on id in ascend
ing order
        //returns a negative integer, zero, or a positi
ve integer as this employee id
        //is less than, equal to, or greater than the s
pecified object.
        return (this.id - emp.id);
    }

    @Override
    //this is required to print the user friendly infor
mation about the Employee
    public String toString() {
        return "[id=" + this.id + ", name=" + this.name
 + ", age=" + this.age + ", salary=" +
                this.salary + "]";
    }

    /**
     * Comparator to sort employees list or array in or
der of Salary
     */
    public static Comparator<Employee> SalaryComparator
 = new Comparator<Employee>() {

        @Override
        public int compare(Employee e1, Employee e2) {

            return (int) (e1.getSalary() - e2.getSalary
());
        }
    };

    /**
     * Comparator to sort employees list or array in or
der of Age
     */
    public static Comparator<Employee> AgeComparator =
new Comparator<Employee>() {

        @Override
        public int compare(Employee e1, Employee e2) {

            return e1.getAge() - e2.getAge();
        }
    };

    /**
     * Comparator to sort employees list or array in or
der of Name
     */
    public static Comparator<Employee> NameComparator =
 new Comparator<Employee>() {

        @Override
        public int compare(Employee e1, Employee e2) {

            return e1.getName().compareTo(e2.getName())
;
        }
    };
}

```

```
83. public class EmployeeComparatorByIdAndName implements C
    omparator<Employee> {
84. 
85.     @Override
86.     public int compare(Employee o1, Employee o2) {
87.         int flag = o1.getId() - o2.getId();
88.         if(flag==0) flag = o1.getName().compareTo(o2.ge
    tName());
89.         return flag;
90.     }
91. 
92. }
93. 
94. public class JavaObjectSorting {
95. 
96.     /**
97.      * This class shows how to sort custom objects arra
    y/list
98.      * implementing Comparable and Comparator interface
    s
99.      * @param args
100.     */
101.    public static void main(String[] args) {
102.        Map<Integer, String> datos = new HashMap<Integ
    er, String>();
103.        datos.put(1, "uno");
104.        datos.put(2, "dos");
105.        datos.put(3, "tres");
106. 
107.        for (Map.Entry<Integer, String> entry : datos.
    entrySet()) {
108.            System.out.println("clave=" + entry.getKey
    () + ", valor=" + entry.getValue());
109.        }
110. 
111.        Set<String> set = new HashSet<String>();
112. 
113.        //populate set
114. 
115.        for (String s : set) {
116.            System.out.println(s);
117.        }
118. 
119.        //sorting custom object array
```

```
120.        Employee[] empArr = new Employee[4];
121.        empArr[0] = new Employee(10, "Mikey", 25, 1000
    0);
122.        empArr[1] = new Employee(20, "Arun", 29, 20000
    );
123.        empArr[2] = new Employee(5, "Lisa", 35, 5000);
124.        empArr[3] = new Employee(1, "Pankaj", 32, 5000
    0);
125. 
126.        //sorting employees array using Comparable int
    erface implementation
127.        Arrays.sort(empArr);
128.        System.out.println("Default Sorting of Employe
    es list:\n"+Arrays.toString(empArr));
129. 
130.        //sort employees array using Comparator by Sal
    ary
131.        Arrays.sort(empArr, Employee.SalaryComparator)
    ;
132.        System.out.println("Employees list sorted by S
    alary:\n"+Arrays.toString(empArr));
133. 
134.        //sort employees array using Comparator by Age
135.        Arrays.sort(empArr, Employee.AgeComparator);
136.        System.out.println("Employees list sorted by A
    ge:\n"+Arrays.toString(empArr));
137. 
138.        //sort employees array using Comparator by Nam
    e
139.        Arrays.sort(empArr, Employee.NameComparator);
140.        System.out.println("Employees list sorted by N
    ame:\n"+Arrays.toString(empArr));
141. 
142.        //Employees list sorted by ID and then name us
    ing Comparator class
143.        empArr[0] = new Employee(1, "Mikey", 25, 10000
    );
144.        Arrays.sort(empArr, new EmployeeComparatorById
    AndName());
145.        System.out.println("Employees list sorted by I
    D and Name:\n"+Arrays.toString(empArr));
```

```
    }

}
```

Fast Input – Java

```
// Working program with FastReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
import java.util.StringTokenizer;

public class Main
{
    static class FastReader
    {
        BufferedReader br;
        StringTokenizer st;

        public FastReader()
        {
            br = new BufferedReader(new
                InputStreamReader(System.in));
        }

        String next()
        {
            while (st == null || !st.hasMoreElements())
            {
                try
                {
                    st = new StringTokenizer(br.readLine());
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
            }
            return st.nextToken();
        }

        int nextInt()
        {
            return Integer.parseInt(next());
        }

        long nextLong()
        {
            return Long.parseLong(next());
        }

        double nextDouble()
        {
            return Double.parseDouble(next());
        }

        String nextLine()
        {
            String str = "";
            try
            {
                str = br.readLine();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
            return str;
        }
    }

    public static void main(String[] args)
    {
        FastReader s=new FastReader();
        int n = s.nextInt();
        int k = s.nextInt();
        int count = 0;
        while (n-- > 0)
        {
            int x = s.nextInt();
            if (x%k == 0)
            count++;
        }
        System.out.println(count);
    }
}
```