El Chivo Ma Sicario

ICPC 2018-2019

Contents

1 General Tricks

1.1 Easy PI

```
    const double pi = 4.0 * atan(1.0);
    const double PI = 3.1415926535897932385;
```

1.2 Check if number is Power of two

```
//check if number is power of two
bool ispot(int x) { return x && (!(x &(x-1)));}
```

1.3 Enumerate submask of a bitmask

```
for (int s=m; ; s=(s-1)&m) {
 ... you can use s ...
 if (s==0) break;
}
```

1.4 ditto pero 3^n pa DP

```
for (int m=0; m<(1<<n); ++m)
    for (int s=m; s; s=(s-1)&m)
 ... s and m ...
```

1.5 Swap two nums with xor

```
int a = 10, b = 20;
a ^= b;
b ^= a;
a ^= b;
cout << a << ' ' << b << endl; // 20 10
```

1.6 Balanced bracketsequence – Catalan

```
long long catalan[200];

long long solve(int n) {
   if (n % 2 == 1) return 0; // odd
```

```
  if (n == 0) return 1; // empty
  long long & r = catalan[n];
  if (r != -1) return r;
  r = solve(n - 2); // case (R)
  // case (R)Q
  for (int i = 2; i < n; i += 2)
    r += solve(i - 2) * solve(n - i);
  return r;
}

int main() {
  memset(catalan, -1, sizeof catalan);
  for (int i = 0; i <= 60; i += 2)
    cout << solve(i) << " ";
  return 0;
}
```

1.7 built-in funcs

1.7.1 GCD

```
__gcd(x,y);
```

1.7.2 Count number of ones in binary

```
int x = 9 // 1001, 2 set bits
int sbx = __builtin_popcount(x); returns 2
```

2 Data Structures

2.1 Minimum-set and coordinate compression

```
//to-do -> hacer que soporte updates
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;

//Cuenta cuantos numeros menores que n hay en el arreglo (query)
//Para esto ordena los elementos y borra duplicados. N de elementos
    menores a x =
// elementos diferentes a x a la izquierda.
```

```cpp
struct LowCount {
    vector<int> V;
    LowCount(const vector<int>& v) {
        V = v;
        sort(V.begin(), V.end());
        V.erase(unique(V.begin(),V.end()),V.end());
    };
    int query(int n) {
        int Q = lower_bound(V.begin(),V.end(), n) - V.begin();
        return Q;
    }
};
vector<int> V;
vector<pair<int,int> > cV;

int main() {
    LowCount v(V);
    //coordinate compression -> asigna un key x a cada numero del arreglo
    for(int i = 0; i < 1000000; i++)
        V.push_back(pow(rand(),2));

    for(int& i : V) {
        cV.push_back({v.query(i), i});
    }

    for(auto i : cV) {
        cout << i.first << ' ' << i.second << endl;
    }
    cout << "done" << endl;
    return 0;
}
```

2.2 NearestSet (Used on C - Equalize!)

```cpp
struct NearestSet {
    set<int> s;
    void insert(int pos) {
        s.insert(pos);
    }
    int nearestElement(int pos) {
        auto k = s.lower_bound(pos);
        int best = -1;
        if (k != s.end()) {
            best = *k;
        }
        if (k != s.begin()) {
            k--;
            if (best == -1 || abs(pos - *k) <= abs(pos - best)) {
```

```
                best = *k;
            }
        }
        return best;
    }
};
```

2.3 SQRT-Decomposition

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using pll = pair<ll,ll>;
using VI = vector<int>;
using VL = vector<ll>;
using VVI = vector<VI>;
const int INF = (1 << 30);
//Sqrt debaratation
struct SQRTDecomp {
    //Este code resuelve D-query de SPOJ.
    int NC,NR;
    vector<vector<ll> > SQMat;
    vector<ll> sumArr;
    vector<ll> lazy;
    SQRTDecomp(vector<ll> &V) {
        NC = 1000; //change acording to constraint
        while(V.size() % NC != 0) V.push_back((0)); //avoid division by 0
        NR = (int)V.size() / NC;
        cerr << "NC : " << NC << endl;
        cerr << "NR : " << NR << endl;

        SQMat = vector<vector<ll> > (NR, vector<ll> (NC));
        sumArr = vector<ll> (NR);
        lazy = vector<ll> (NR + 5, 0);
        //modify sqrt decomposition creation if needed
        for(int i = 0; i < (int)V.size(); i++) {
            int row = i / NC;
            int col = i % NC;
            SQMat[row][col] = V[i];
            sumArr[row] += V[i]; //modify this op for sumArr in case of
                change
        }
    }
    //applies lazy update over a whole row, modify as needed (or delete)
    void applyLazy(int pos) {
        if (lazy[pos] != 0) {
            for (int i = 0; i < NC; i++)
                SQMat[pos][i] += lazy[pos];
```

```
        lazy[pos] = 0; //represents empty update.
    }
}
//updates elements in range by a delta
void update(int pos, int delta) {
    int row = pos / NC;
    int col = pos % NC;
    SQMat[row][col] += delta;
    sumArr[row] += delta;
}
void update(int l, int r, ll delta) {
    int startRow = l/NC;
    int endRow = r/NC;
    int startCol = l % NC;
    int endCol = r % NC;
    applyLazy(startRow);
    applyLazy(endRow);
    if (startRow == endRow) {
        for(int i = startCol; i <= endCol; i++)
            SQMat[startRow][i] += delta;
        sumArr[startRow] += (delta*(r-l+1)); //sums delta times the
            range updated
        return;
    }
    for(int i = startCol; i < NC; i++) {
        SQMat[startRow][i] += delta;
        sumArr[startRow] += delta;
    }
    for(int i = startRow+1; i < endRow; i++) {
        sumArr[i] += delta*NC; //sets the total sum to delta times
            the number of columns
        lazy[i] += delta; //
    }
    for(int i = 0; i <= endCol; i++) {
        SQMat[endRow][i] += delta;
        sumArr[endRow] += delta;
    }
}
//gets range sum
ll queryRange(int l, int r) {
    if (l > r)
        return -INF; //return -INF if query is not valid!
    ll ans = 0;
    int startRow = l / NC;
    int endRow = r / NC;
    int startCol = l % NC;
    int endCol = r % NC;
    //remove if not needed
    applyLazy(startRow);
    applyLazy(endRow);
```

```
        if (startRow == endRow) {
            for (int i = startCol; i <= endCol; i++)
                ans += SQMat[startRow][i];
            return ans;
        }

        for(int i = startCol; i < NC; i++) {
            ans += SQMat[startRow][i];
        }
        for(int i = startRow+1; i < endRow; i++) {
            ans += sumArr[i];
        }
        for(int i = 0; i <= endCol; i++) {
            ans += SQMat[endRow][i];
        }

        return ans;
    }
};

int main() {
    int n;
    cin >> n;
    VL V(n, 0);
    int q;
    cin >> q;
    SQRTDecomp D(V);
    while (q--) {
        int t; //type of query
        cin >> t;
        if (!t) {
            ll i,j,k;
            cin >> i >> j >> k;
            i--,j--;
            D.update(i,j,k);
        }
        else {
            ll i, j;
            cin >> i >> j;
            i--,j--;
            cout << D.queryRange(i, j) << endl;
        }
    }
    return 0;
}
```

2.4 Segment Tree

```
#include <bits/stdc++.h>
using namespace std;
using Long = long long;
//This segment tree has lazy propagation.
struct SegmentTree {
private:
    struct Node {
        Long val;
        bool isLazy;
        Long lazyVal;
        Node (Long _v) {
            val = _v;
            isLazy = false;
            lazyVal = 0;
        }
    };

    vector<Long> V;
    vector<Node> Tree;
    const int INF = (1 << 30);

    Node merge (Node p1, Node p2) {
        return p1.val + p2.val;
    }

    void pushDown(int id, int st, int en, int mid) {
        if (Tree.at(id).isLazy) {
            if (st != en) {
                Tree.at(id*2).lazyVal += Tree.at(id).lazyVal;
                Tree.at(id*2).val += (mid-st+1) * Tree.at(id).lazyVal;
                Tree.at(id*2 + 1).lazyVal += Tree.at(id).lazyVal;
                Tree.at(id*2 + 1).val += ((en)-(mid+1)+1) *
                    Tree.at(id).lazyVal;
                Tree.at(id*2).isLazy = true;
                Tree.at(id*2 + 1).isLazy = true;
            }
            Tree.at(id).lazyVal = 0;
            Tree.at(id).isLazy = false;
        }
    }

    void create(int id, int st, int en) {
        if (st == en) {
            Tree.at(id) = V.at(st);
            return;
        }
        int le = id*2;
```

```
        int ri = le + 1;
        int mid = (st + en) / 2;
        create(le, st, mid);
        create(ri, mid + 1, en);
        Tree.at(id) = merge(Tree.at(le), Tree.at(ri));
        //cout << "Created tree id " << id << " alo " << Tree[id].val <<
            endl;
    }

    Node query(int id, int st, int en, int lef, int ri) {
        if (st > ri || en < lef) {
            assert(false);
        }
        int mid = (st + en) / 2;
        if (lef <= st && en <= ri) { //totally inside range
            return Tree.at(id);
        }
        pushDown(id,st,en,mid);
        if (lef > mid) //go right, range is to the right
            return query(id*2 + 1, mid + 1, en, lef, ri);
        else if (ri <= mid) // go left
            return query(id*2, st, mid, lef, ri);

        Node lQ = query(id*2, st, mid, lef, ri);
        Node rQ = query(id*2 + 1, mid + 1, en, lef, ri);
        return merge(lQ,rQ);
    }
    //range update with delta
    void update(int id, int st, int en, int lIdx, int rIdx, Long delta) {
        int mid = (st+en) / 2;
        if (st > en) assert(false);
        if (st > rIdx || en < lIdx) return;
        if (lIdx <= st && en <= rIdx) {
            Tree.at(id).val += (en - st + 1) * delta;
            Tree.at(id).lazyVal += delta;
            Tree.at(id).isLazy = true;
            return;
        }
        pushDown(id,st,en,mid);
        update(id*2 + 1, mid + 1, en, lIdx, rIdx, delta);
        update(id*2, st, mid, lIdx, rIdx, delta);
        Tree.at(id) = merge(Tree.at(2*id), Tree.at(2*id + 1));
    }
public:
    SegmentTree(vector<Long> v) {
        V = v;
        Tree = vector<Node>(4*(int)V.size(), 0);
        create(1, 0, V.size()-1);
    }
```

```
    Long query(int lef, int ri) {
        return query(1, 0, V.size()-1, lef, ri).val;
    }
    void update(int lIdx, int rIdx, Long delta) {
        update(1,0,V.size()-1, lIdx, rIdx, delta);
    }
};

int main() {
    int tc;
    scanf("%d", &tc);
    while(tc--) {
        int n, q;
        scanf("%d %d", &n, &q);
        vector<Long> V(n + 10);
        SegmentTree bazooka(V);
        while(q--) {
            int ty;
            scanf("%d", &ty);
            if (!ty) {
                int a,b;
                Long v;
                scanf("%d %d %lld", &a, &b, &v);
                a--,b--;
                bazooka.update(a,b,v);
            }
            else {
                int a,b;
                scanf("%d %d", &a, &b);
                a--,b--;
                printf("%lld\n",bazooka.query(a,b));
            }
        }
    }
    return 0;
}
```

2.5 Fengüis tree

```
#include <iostream>
#include <vector>
#include <map>
#include <set>
using namespace std;

struct FenwickTree {
    //literalmente solo e eta shit
    vector<int> tri;
```

```
    FenwickTree(int N) : tri(N+10, 0) {}
    void add(int x, int d) {
        for (int i = x + 1; i < tri.size(); i += i&(-i)) {
            tri[i] += d;
        }
    }

    int query(int x) {
        int ans = 0;
        for (int i = x + 1; i > 0; i -= i&(-i)) {
            ans += tri[i];
        }
        return ans;
    }

    void pr() {
        for(int i = 0; i < (int)tri.size(); i++)
            cout << i+1 << ' ';
        cout << endl;
        for(int i = 0; i < (int)tri.size(); i++)
            cout << "query for i " << i << " : " << query(i) << ' ';
        cout << endl;
    }
};

int main() {
    FenwickTree FT(16);
    FT.add(6,8);
    FT.add(3,2);
    FT.pr();
    cout << FT.query(6) - FT.query(2) << endl;
    FT.add(8,10);
    FT.add(14,1);
    FT.pr();
    cout << FT.query(13) << endl;
    cout << FT.query(14) << endl;
    cout << FT.query(8) << endl;
    return 0;
}
```

2.6 Disjoint Set

2.6.1 Version Toribio

```
#include <vector>
#include <iostream>
using namespace std;
struct DisjointSet
```

```
{
   vector<int> P; // if < 0 then negative size, else parentId
   DisjointSet(int N) : P(N, -1) {}
   int find(int x) {
      return P[x] < 0 ? x : (P[x] = find(P[x]));
   }
   bool join(int x,int y) {
      if((x = find(x)) == (y = find(y))) return false;
      if(P[y] < P[x]) swap(x,y);
      P[x] += P[y];
      P[y] = x;
      return true;
   }
};
```

2.6.2 Version Joa (Usada para Kruskal)

```
#include <joa/wave.h>

using namespace std;

class DisjointSet {
   int N;
   int ncomp;
   vector<int> par;
   vector<int> rank;

public:
   DisjointSet(size_t _N) : N(_N), ncomp(_N), par(_N, -1), rank(_N, 0) {}
   void reset() {
      par.assign(N, -1);
      rank.assign(N, 0);
      ncomp = N;
   }
   int size() const {
      return ncomp;
   }
   int find_rep(int u) {
   // path compression
      return par[u] < 0 ? u : par[u] = find_rep(par[u]);
      /*
      vector<int> s;
      while (par[u] >= 0) {
         s.push_back(u);
         u = par[u];
      }
      for (int i = 0; i < (int) s.size(); ++i)
         par[s[i]] = u;
```

```
      return u;
      */
   }
   bool union_rep(int u, int v) {
      int u_root = find_rep(u);
      int v_root = find_rep(v);
      if (u_root == v_root)
         return false;
      if (rank[u_root] > rank[v_root])
         par[v_root] = u_root;
      else {
         par[u_root] = v_root;
         if (rank[u_root] == rank[v_root])
            rank[v_root] = rank[u_root] + 1;
      }
      --ncomp;
      return true;
   }
};

int main() {
   DisjointSet dset(1000);
// join elements
   dset.union_rep(3, 10);
   dset.union_rep(500, 87);
   dset.union_rep(77, 760);
   dset.union_rep(500, 10);
   printf("Representative of %d is %d\n", 3, dset.find_rep(5));
   printf("Representative of %d is %d\n", 77, dset.find_rep(77));
   return 0;
}
```

2.7 Fraction Representation

```
using ll = long long;
#define ftype ll //replaces int or long long with whatever's needed
struct Fraction {
    ftype num;
    ftype dem;
    Fraction(ftype _num, ftype _dem) {
        num = _num;
        dem = _dem;
    }
    bool operator<(Fraction f)const {
        return f.dem * num < f.num * dem;
    }
    Fraction operator+(Fraction f)const {
        return Fraction(num * f.dem + f.num * dem, f.dem * dem);
```

```
    }
    Fraction operator-(Fraction f)const {
        return Fraction(num * f.dem - f.num * dem, f.dem * dem);
    }
    Fraction operator*(Fraction f)const {
        return Fraction(num * f.num, dem * f.dem);
    }
    Fraction operator/(Fraction f)const {
        return Fraction(num, dem) * Fraction(f.dem, f.num);
    }
};
```

2.8 Matrix

```
#include <bits/stdc++.h>
#define N 100
using namespace std;
using ll = int;
const int MOD = 1e9+7;
struct Matrix {
   vector<vector<ll>> M;
   Matrix() {
      M = vector<vector<ll>> (N, vector<ll>(N));
   }

   Matrix operator*(Matrix &b) const {
      Matrix C = Matrix();
      for(int i = 0; i < N; ++i)
         for(int j = 0; j < N; ++j)
            for(int k = 0; k < N; ++k)
               C.M[i][j] = (C.M[i][j] + 1LL * M[i][k] * b.M[k][j]) % MOD;
      return C;
   }

   Matrix operator+(const Matrix &b) const {
      Matrix C = Matrix();
      for(int i = 0; i < N; ++i)
         for(int j = 0; j < N; ++j)
            C.M[i][j] = M[i][j] + b.M[i][j];
      return C;
   }

   Matrix unit() {
      Matrix C;
      for(int i = 0; i < N; i++)
         C.M[i][i] = 1;
      return C;
   }
```

```
};

Matrix modPow(Matrix A, int n) {
   if (n == 0)
      return A.unit();
   Matrix h = modPow(A, n / 2);
   Matrix o = h*h;
   if (n % 2)
      o = o*A;
   return o;
}

int main() {
   Matrix A, B;
   for(int i = 0; i < 3; i++)
      for(int j = 0; j < 3; j++)
         A.M[i][j] = 3;
   for(int i = 0; i < 3; i++)
      for(int j = 0; j < 3; j++)
         B.M[i][j] = 3;
   Matrix C = A + B;
   modPow(C, 2);
   for(int i = 0; i < 3; i++) {
      for(int j = 0; j < 3; j++)
         cout << C.M[i][j] << ' ';
      cout << endl;
   }
   Matrix neo //lmao
   return 0;
}
```

3 Algorithms

3.1 Minimum Spanning Tree

```
#include <cstdio>

#include <vector>
#include <algorithm>

using namespace std;

class DisjointSet {
   int N;
   int ncomp;
   vector<int> par;
   vector<int> rank;
```

```
public:
   DisjointSet(size_t _N) : N(_N), ncomp(_N), par(_N, -1), rank(_N, 0) {}
   void reset() {
      par.assign(N, -1);
      rank.assign(N, 0);
      ncomp = N;
   }
   int size() const {
      return ncomp;
   }
   int find_rep(int u) {
   // path compression
      return par[u] < 0 ? u : par[u] = find_rep(par[u]);
      /*
      vector<int> s;
      while (parent[u] >= 0) {
         s.push_back(u);
         u = parent[u];
      }
      for (int i = 0; i < (int) s.size(); ++i)
         parent[s[i]] = u;
      return u;
      */
   }
   bool union_rep(int u, int v) {
      int u_root = find_rep(u);
      int v_root = find_rep(v);
      if (u_root == v_root)
         return false;
      if (rank[u_root] > rank[v_root])
         par[v_root] = u_root;
      else {
         par[u_root] = v_root;
         if (rank[u_root] == rank[v_root])
            rank[v_root] = rank[u_root] + 1;
      }
      --ncomp;
      return true;
   }
};

struct Edge {
   int u, v;
   int cost;
   Edge(int _u, int _v, int _cost) : u(_u), v(_v), cost(_cost) {}
};

class CostCmp {
public:
```

```
   bool operator()(const Edge& e1, const Edge& e2) {
      if (e1.cost != e2.cost) return e1.cost < e2.cost;
      if (e1.u != e2.u) return e1.u < e2.u;
      return e1.v < e2.v;
   }
};

// vector<bool> in_mst;
long long kruskal(int N, vector<Edge> Edges) {
// in_mst.assign( edges.size(), false );
   sort(Edges.begin(), Edges.end(), CostCmp());
   DisjointSet dset(N);
   long long cost = 0;
   for (int j = 0; j < int(Edges.size()) && int(dset.size()) > 1; ++j) {
      if (dset.union_rep(Edges[j].u, Edges[j].v)) {
         cost += Edges[j].cost;
      // in_mst[ edges[j].id ] = true;
      }
   }
   return cost;
}

/*
7 11
0 1 7
0 3 5
1 2 8
1 3 9
1 4 7
2 4 5
3 4 15
3 5 6
4 5 8
4 6 9
5 6 11
*/

int main(int argc, char* argv[]) {
   int N, M;
   vector<Edge> edges;
   scanf("%d %d", &N, &M);
   for (int j = 0; j < M; ++j) {
      int u, v, cost;
      scanf("%d %d %d", &u, &v, &cost);
      edges.push_back(Edge(u, v, cost));
   }

   long long res = kruskal();
   printf("%lld\n", res);
```

```
   return 0;
}
```

3.2 BFS

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <cassert>
#include <queue>
#define INF 1e9+7
using namespace std;
vector<int> Nodes[1000];
vector<int> d(1000, INF);

void bfs(int u) {
    queue<int> q;
    q.push(u);

    while(!q.empty()) {
        int v = q.front();
        q.pop();
        cerr << "for layer " << v << endl;
        for(int i = 0; i < (int)Nodes[v].size(); i++) {
            if (d[Nodes[v][i]] == INF) {
                d[Nodes[v][i]] = d[v] + 1;
                q.push(Nodes[v][i]);
                cerr << "vis : " << Nodes[v][i] << " with new dist " <<
                    d[Nodes[v][i]] <<endl;
            }
        }

    }

}

int main() {

   Nodes[0].push_back(1);
   Nodes[0].push_back(2);
   Nodes[0].push_back(3);
   Nodes[1].push_back(3);
   Nodes[1].push_back(4);
   Nodes[4].push_back(5);
   Nodes[3].push_back(1);
   Nodes[6].push_back(7);
   Nodes[7].push_back(8);
```

```
    Nodes[7].push_back(9);
    Nodes[9].push_back(7);

    int n, m, v;
    for(int i = 0; i < 10; i++) {
        cout << "Adj list for " << i << endl;
        for(auto k : Nodes[i]) {
            cout << k << ' ';
        }
        cout << endl;
    }


    int ncomp = 0;

    for(int i = 0; i < 10; i++) {
        if(d[i] == INF) {
            ncomp++;
            cerr << "CALL BFS" << endl << endl;
            d[i] = 0;
            bfs(i);
        }
    }
    for(int i = 0; i < 10; i++) {
        cout << "Dist Node " << i << " : " << d[i] << endl;
    }
    cout << ncomp << endl;
}
```

3.3 Ditra

```
#include <iostream>
#include <string>
#include <queue>
#include <algorithm>
using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

#define MAXN 1004

int N;
VVI adj;
VVI adjC;


struct State {
```

```
   int id;
   int cost;
   State( int _id, int _cost ) {
      id  = _id;
      cost = _cost;
   }

   State() {}

   bool operator< ( const State& s ) const {
      if (cost != s.cost)
         return cost > s.cost;


      return id > s.id;
   }
};

const int INF = 1000000000;
int D[MAXN];
State P[MAXN];

int ditra( int src, int dst ) {

   for (int u = 0; u < N; ++u)
      D[u] = INF;

   D[src] = 0;
   P[src].id = -1;

   priority_queue<State> pq;
   pq.push( State(src, 0) );

   while ( !pq.empty() ) {
      State cur = pq.top();
      pq.pop();

      if (cur.id == dst) {
         vector<State> V;
         for (State x = cur; x.id != -1; x = P[x.id])
            V.push_back(x);

         reverse(V.begin(), V.end());

         for (int j = 0; j < V.size(); ++j) {
            if (j > 0) cout << " -> ";
            cerr << V[j].id;
         }
         cerr << endl;
```

```
            return D[cur.id];
        }

        if ( cur.cost > D[cur.id] )
            continue;

        for (int j = 0; j < adj[cur.id].size(); ++j) {
            State nxt( adj[cur.id][j], cur.cost );
            nxt.cost += adjC[cur.id][j];

            if (D[nxt.id] > nxt.cost) {
                D[nxt.id] = nxt.cost;
                P[nxt.id] = cur;
                pq.push(nxt);
            }
        }
    }
    return INF;  // o devuelves -1
}

int main() {
     N = 11;
    adj = VVI(N);
    adjC = VVI(N);

    adj[0].push_back(1); adjC[0].push_back(4);

    adj[1].push_back(0); adjC[1].push_back(4);
    adj[1].push_back(5); adjC[1].push_back(3);

    adj[2].push_back(5); adjC[2].push_back(3);
    adj[2].push_back(7); adjC[2].push_back(9);

    adj[3].push_back(4); adjC[3].push_back(9);
    adj[3].push_back(9); adjC[3].push_back(2);

    adj[4].push_back(3); adjC[4].push_back(9);
    adj[4].push_back(9); adjC[4].push_back(5);

    adj[5].push_back(1); adjC[5].push_back(3);
    adj[5].push_back(2); adjC[5].push_back(3);
    adj[5].push_back(7); adjC[5].push_back(7);

    adj[6].push_back(7); adjC[6].push_back(4);

    adj[7].push_back(2); adjC[7].push_back(9);
    adj[7].push_back(5); adjC[7].push_back(7);
    adj[7].push_back(6); adjC[7].push_back(4);
    adj[7].push_back(10); adjC[7].push_back(12);
```

```
    adj[9].push_back(4); adjC[9].push_back(5);
    adj[9].push_back(3); adjC[9].push_back(2);

    adj[10].push_back(7); adjC[10].push_back(12);

    int dist = ditra(0, 10);
    cout << "Distancia = " << dist << endl;

    dist = ditra(1, 9);
    cout << "Distancia = " << dist << endl;

    return 0;

}
```

3.4 MergeSort

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;
int inv = 0;

vector<int> merge(vector<int>& A, vector<int>& B) {
    vector<int> res;
    int pA = 0, pB = 0;

    while(pA < A.size() || pB < B.size()) {
        if (pA < A.size() && pB < B.size()) {
            if (A[pA] <= B[pB]) {
                res.push_back(A[pA]);
                pA++;
            }
            else {
                res.push_back(B[pB]);
                pB++;
                inv+=A.size()-pA;
            }
        } else if (pA == A.size()) {
            while(pB < B.size()) {
                res.push_back(B[pB]);
                pB++;
            }
        } else {
            while(pA < A.size()) {
                res.push_back(A[pA]);
                pA++;
```

```
            }
        }
    }
    return res;
}

vector<int> mergeSort(vector<int> A) {
   if ((int)A.size() == 1)
      return A;

   vector<int> vA, vB;

   for(int i = 0; i < (int)A.size()/2; i++)
        vA.push_back(A[i]);

   for(int i = (int)A.size()/2; i < A.size(); i++)
      vB.push_back(A[i]);

   vA = mergeSort(vA);
   vB = mergeSort(vB);
   vector<int> res = merge(vA,vB);
   return res;
}

int main() {
  inv = 0;
   vector<int> A = {1,3,2,4,5};
   vector<int> B = mergeSort(A);
   for(int& i : B)
      cout << i << ' ';
   cout << "\n inversions: " << inv << endl;
   return 0;
}
```

3.5 Flood-fill

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int N = 10000, M = 1000;
int components = 0;

struct Node {
   char content;
   bool visited;
};
```

```
void floodFill(int r, int c, vector< vector <Node> > & Land) {
   if (r < 0 || r >= N || c < 0 || c >= M) //if out of range
      return;

   if (Land[r][c].visited)
      return;

   //Not visited, keep on
   Land[r][c].visited = true;

   if (Land[r][c].content != '@') //@ -> TARGET CELL
      return;

   //call flood fill recursively in all cardinal directions
   floodFill(r-1, c, Land);
   floodFill(r, c-1, Land);
   floodFill(r+1, c, Land);
   floodFill(r, c+1, Land);
//    floodFill(r+1, c+1, Land);
//    floodFill(r-1, c-1, Land);
//    floodFill(r+1, c-1, Land);
//    floodFill(r-1, c+1, Land);
}

//Traverses the entire graph looking for connected components
void floodAll(vector< vector<Node> >& Land, int N, int M) {
   for(int i = 0; i < N; i++) {
      for(int j = 0; j < M; j++) {
         if (!Land[i][j].visited && Land[i][j].content != '*') {
            floodFill(i,j, Land);
            components++;
         }
      }
   }
}

int main() {
   while(true) {

      cin >> N >> M;
      if (N == 0 || M == 0)
         break;

      vector< vector <Node> > Land(N, vector<Node> (M));

      for(int i = 0; i < N; i++) {
         for (int j = 0; j < M; j++) {
            cin >> Land[i][j].content;
            Land[i][j].visited = false;
```

```
        }
      }

      floodAll(Land, N, M);

      cout << components << endl;

   }
   return 0;
}
```

3.6 Event Processing

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <map>

using namespace std;

struct Event {
    int x;
    char type;
};

int N;
int L[100004], R[100004];

void solve() {
    for(int i = 0; i < N; i++)
        cin >> L[i] >> R[i];

    vector<Event> events;

    for(int i = 0; i < N; i++) {
        events.push_back({L[i], 'E'});
        events.push_back({R[i], 'X'});
    }

    sort(events.begin(), events.end(),
         [&] (Event a, Event b) -> bool {
             if (a.x != b.x) return a.x < b.x;
             return a.type < a.type;
         });
    int cnt = 0;
    int ans = 0;
    for(Event e : events) {
        if (e.type == 'E') {
```

```
            ++cnt;
            ans = max(cnt,ans);
        }
        else
            --cnt;
    }

    cout << ans << endl;

}

int main() {

    vector<int> points = {3,2,10,4};
    vector<int> cp = points;
    sort(cp.begin(), cp.end());
    map<int, int> ccomp;
    for(int i = 0; i < (int)cp.size(); i++) {
        ccomp[i] = cp[i];
    }

    for(auto x : ccomp) {
        cout << x.first << " -> " << x.second << endl;
    }

    return 0;
}
```

3.7 BFS on Grid

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

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;


vector<string> grid = {
   "..........",
   "..$.....*..",
   ".*****..*..",
   ".....*..*..",
   "...#.*..*..",
   ".........."
};

struct Pos {
```

```
   int row, col;
};

int dr[] = {-1, 0, 0, 1};
int dc[] = {0, -1, 1, 0};

int bfs( Pos src, Pos dst ) {
   const int INF = 1000000000;
   int R = grid.size();
   int C = grid[0].size();

   queue<Pos> q;
   q.push(src);
   VVI D( R, VI( C, INF ) );
   D[ src.row ][ src.col ] = 0;

   while (!q.empty()) {
      Pos cur = q.front();
      q.pop();

      int dist = D[cur.row][cur.col];

      if (cur.row == dst.row && cur.col == dst.col) // llegue al destino?
         return dist;
      for (int k = 0; k < 4; ++k) {
         Pos nxt = { cur.row + dr[k], cur.col + dc[k] };
         if ( nxt.row < 0 || nxt.row >= R || nxt.col < 0 || nxt.col >= C
             )
            continue;
         if ( grid[ nxt.row ][ nxt.col ] == '*' )
            continue;
         if ( D[ nxt.row ][ nxt.col ] == INF ) {
            D[ nxt.row ][ nxt.col ] = dist + 1;
            q.push(nxt);
         }

      }

   }

   return -1;
}



int main(int argc, char* argv[]) {

   int res = bfs( {1, 2}, {4, 3} ); // bfs desde la posicion (1, 2) a
       (4, 3)
   cout << "Cantidad de pasos: " << res << endl;
```

```
   return 0;
}
```

4 Math

4.1 Prime factorization

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <cmath>
using namespace std;

//get prime factors on vector
vector<int> primeDecomp(int n) {
   vector<int> pf;
   int pfidx = 0, PF = primeStuff[pfidx]; //sieve with primes
   while(PF*PF <= n) {
      while(n % PF == 0) {
         n /= PF;
         pf.push_back(PF);
      }
      PF = primeStuff[++pfidx];
   }
   if (n != 1)
      pf.push_back(PF);
   return pf;
}
```

4.2 LCM

```
int LCM(int a, int b) {
   return b*a/GCD(a,b);
}
```

4.3 Sieve

```
typedef long long ll;

bool isPrime[2000002];
set<ll> tPrimes;
```

```
//klk joa
void Sieve() {
   for(int i = 2; i <= 2000000; i++) {
      if(!isPrime[i]) {
         for(int j = i + i; j <= 2000000; j += i) {
            isPrime[j] = true;
         }
      }
   }
}
```

4.4 Pascal Triangle

```
const int maxn = ...;
int C[maxn+1][maxn+1];
for (int n=0; n<=maxn; ++n) {
    C[n][0] = C[n][n] = 1;
    for (int k=1; k<n; ++k)
        C[n][k] = C[n-1][k-1] + C[n-1][k];
}
```

4.5 ModPow

```
unsigned int powmod(unsigned long long x, unsigned long long n, unsigned
    int m)
{
   unsigned long long res = 1 % m;
   x %= m;
   for (; n > 0; n >>= 1) {
      if (n & 1) {
         res = (res * x) % m;
      // n--;
      }
      x = (x * x) % m;
   }
   return (unsigned int) res;
}


vector<int> primes;
void sieve() {
   // use sieve to fill in primes vector
   // ...
}
```

5 Geometry

5.1 Point 2d

```
struct point2d {
    ftype x, y;
    point2d() {}
    point2d(ftype x, ftype y): x(x), y(y) {}
    point2d& operator+=(const point2d &t) {
        x += t.x;
        y += t.y;
        return *this;
    }
    point2d& operator-=(const point2d &t) {
        x -= t.x;
        y -= t.y;
        return *this;
    }
    point2d& operator*=(ftype t) {
        x *= t;
        y *= t;
        return *this;
    }
    point2d& operator/=(ftype t) {
        x /= t;
        y /= t;
        return *this;
    }
    point2d operator+(const point2d &t) const {
        return point2d(*this) += t;
    }
    point2d operator-(const point2d &t) const {
        return point2d(*this) -= t;
    }
    point2d operator*(ftype t) const {
        return point2d(*this) *= t;
    }
    point2d operator/(ftype t) const {
        return point2d(*this) /= t;
    }
};
point2d operator*(ftype a, point2d b) {
    return b * a;
}
```

5.2 Operaciones con 2-D geometry

```
ftype dot(point2d a, point2d b) {
    return a.x * b.x + a.y * b.y;
}
ftype norm(point2d a) {
    return dot(a, a);
}
double abs(point2d a) {
    return sqrt(norm(a));
}
double proj(point2d a, point2d b) {
    return dot(a, b) / abs(b);
}
double angle(point2d a, point2d b) {
    return acos(dot(a, b) / abs(a) / abs(b));
}
point3d cross(point3d a, point3d b) {
    return point3d(a.y * b.z - a.z * b.y,
                   a.z * b.x - a.x * b.z,
                   a.x * b.y - a.y * b.x);
}
ftype triple(point3d a, point3d b, point3d c) {
    return dot(a, cross(b, c));
}
ftype cross(point2d a, point2d b) {
    return a.x * b.y - a.y * b.x;
}

point2d intersect(point2d a1, point2d d1, point2d a2, point2d d2) {
    return a1 + cross(a2 - a1, d2) / cross(d1, d2) * d1;
}
```

6 Soluciones sicarias

6.1 LIS con Fenwick

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

void bf(vector<int>& a) {
    vector<int> dp(a.size(),1);
    for(int i = 0; i < (int)a.size(); i++)
        for(int j = 0; j < i; j++)
            if (a[j] < a[i])
                dp[i] = max(dp[i], dp[j]+1);
    int ans = *max_element(dp.begin(), dp.end());
    cout << ans << " bf "<< endl;
}
```

```
struct FenwickTree {

    vector<int> tri;
    FenwickTree(int N) : tri(N, 0) {}
    void add(int x, int d) {
        for (int i = x + 1; i < tri.size(); i += i&(-i)) {
            tri[i] = max(tri[i],d);
            //cout << i << " :add : " << tri[i] << endl;
        }
    }

    int query(int x) {
        int ans = 0;
        for (int i = x + 1; i > 0; i -= i&(-i)) {
            ans = max(ans,tri[i]);
            //cout << i << " get: " << tri[i] << endl;
        }
        return ans;
    }

    void pr() {
        for(int i = 0; i < (int)tri.size(); i++)
            cout << i+1 << ' ';
        cout << endl;
        for(int i = 0; i < (int)tri.size(); i++)
            cout << tri[i] << ' ';
        cout << endl;
    }
};

/*
 7
 1 3 5 4 3 7 3
 */
int main() {
    int n;
    cin >> n;
    vector<int> s(n);
    for(int &t : s) t = rand() % 7;
    for(int &t : s)
        cout << t << ' ';
    cout << endl;
    FenwickTree FT(22);
    bf(s);
    int ans = 0;
    for(int i = 0; i < (int)s.size(); i++) {
        int q = FT.query(s[i]);
        ans = max(ans, q+1);
        FT.add(s[i],q+1);
     // FT.pr();
```

```
    }
    cout << ans << endl;
}
```

6.2 Bracket matching segment tree

```
#include <bits/stdc++.h>
using namespace std;
using Long = int;
//This segment tree has lazy propagation.
struct SegmentTree {
private:
    struct Node {
        Long match;
        Long op;
        Long cl;
        bool isLazy;
        Long lazyVal;
        Node (Long _m, Long _o, Long _c) {
            match = _m;
            op = _o;
            cl = _c;
        }
    };

    vector<char> V;
    vector<Node> Tree;
    const int INF = (1 << 30);

    Node merge (Node p1, Node p2) {
        Node N(0,0,0);
        N.match = p1.match + p2.match + min(p1.op, p2.cl);
        N.op = p1.op + p2.op - min(p1.op, p2.cl);
        N.cl = p1.cl + p2.cl - min(p1.op, p2.cl);
        return N;
    }

    void create(int id, int st, int en) {
        if (st == en) {
            if (V[st] == '('
            Tree[id] = Node(0,1,0);
         else if (V[st] == ')')
            Tree[id] = Node(0,0,1);
            return;
        }
        int le = id*2;
        int ri = le + 1;
        int mid = (st + en) / 2;
```

```
        create(le, st, mid);
        create(ri, mid + 1, en);
        Tree.at(id) = merge(Tree.at(le), Tree.at(ri));
        //cout << "Created tree id " << id << " st en " << st << ' ' <<
            en << " alo " << Tree[id].match << endl;
    }

    Node query(int id, int st, int en, int lef, int ri) {
        if (st > ri || en < lef) {
            assert(false);
        }
        int mid = (st + en) / 2;
        if (lef <= st && en <= ri) { //totally inside range
            return Tree[id];
        }
        if (lef > mid) //go right, range is to the right
            return query(id*2 + 1, mid + 1, en, lef, ri);
        else if (ri <= mid) // go left
            return query(id*2, st, mid, lef, ri);

        Node lQ = query(id*2, st, mid, lef, ri);
        Node rQ = query(id*2 + 1, mid + 1, en, lef, ri);
        return merge(lQ,rQ);
    }
    //range update with delta
    void update(int id, int st, int en, int lIdx, int rIdx, Long delta) {
        int mid = (st+en) / 2;
        if (st > en) assert(false);
        if (st > rIdx || en < lIdx) return;
        if (lIdx <= st && en <= rIdx) {
            if (Tree[id].op)
            Tree[id] = Node(0,0,1);
         else
            Tree[id] = Node(0,1,0);
            return;
        }
        update(id*2 + 1, mid + 1, en, lIdx, rIdx, delta);
        update(id*2, st, mid, lIdx, rIdx, delta);
        Tree.at(id) = merge(Tree.at(2*id), Tree.at(2*id + 1));
    }
public:
    SegmentTree(vector<char> v) {
        V = v;
        Tree = vector<Node>(4*(int)V.size(), Node(0,0,0));
        create(1, 0, V.size()-1);
    }

    bool query(int lef, int ri) {
        return query(1, 0, V.size()-1, lef, ri).op || query(1, 0,
            V.size()-1, lef, ri).cl ? false : true;
```

```
    }
    void update(int lIdx, int rIdx, Long delta) {
        update(1,0,V.size()-1, lIdx, rIdx, delta);
    }
};

int main() {
   ios_base::sync_with_stdio(false);cin.tie(NULL);
   int tc = 10;
   int cnt = 0;
   while(tc-->0) {
      cnt++;
      int n;
      cin >> n;
      vector<char> V(n);
      for(int i = 0; i < n; i++) cin >> V[i];
      cout << "Test " << cnt << ":" << endl;
      int m;
      cin >> m;
      SegmentTree ST(V);
      while(m--) {
         int x;
         cin >> x;
         if (!x) ST.query(0,n-1) ? cout << "YES" << endl : cout << "NO"
             << endl;
         else {
            ST.update(x-1,x-1,0);
         }
      }
   }
   return 0;
}
```

6.3 Rotation segment tree

```
#include <bits/stdc++.h>
using namespace std;
using Long = long long;
//This segment tree has lazy propagation.
struct SegmentTree {
private:
    struct Node {
        int a, b, c;
        bool isLazy;
        Long lazyVal;
        Node() {
            a = 1, b = 0, c = 0;
            isLazy = false;
```

```
        lazyVal = 0;
    }
    Node(int x, int y, int z) {
        a = x;
        b = y;
        c = z;
        lazyVal = 0;
        isLazy = false;
    }
};

vector<Long> V;
vector<Node> Tree;
const int INF = (1 << 30);

void rotate(int id) {
    int temp = Tree.at(id).c;
    Tree.at(id).c = Tree.at(id).b;
    Tree.at(id).b = Tree.at(id).a;
    Tree.at(id).a = temp;
}

Node merge (Node p1, Node p2) {
    return Node(p1.a + p2.a, p1.b + p2.b, p1.c + p2.c);
}

void pushDown(int id, int st, int en, int mid) {
    if (Tree.at(id).isLazy) {
        if (st != en) {
            Tree.at(id*2).lazyVal += Tree.at(id).lazyVal % 3;
            for(int i = 0; i < Tree.at(id).lazyVal; i++) {
                rotate(id * 2);
            }
            Tree.at(id*2 + 1).lazyVal += Tree.at(id).lazyVal % 3;
            for(int i = 0; i < Tree.at(id).lazyVal; i++) {
                rotate(id * 2 + 1);
            }
            Tree.at(id*2).isLazy = true;
            Tree.at(id*2 + 1).isLazy = true;
        }
        Tree.at(id).lazyVal = 0;
        Tree.at(id).isLazy = false;
    }
}

void create(int id, int st, int en) {
    if (st == en) {
        Tree.at(id) = Node(1,0,0);
        return;
    }
```

```
        int le = id*2;
        int ri = le + 1;
        int mid = (st + en) / 2;
        create(le, st, mid);
        create(ri, mid + 1, en);
        Tree.at(id) = merge(Tree.at(le), Tree.at(ri));
        //cout << "Created tree id " << id << " alo " << Tree[id].val <<
            endl;
    }

    Node query(int id, int st, int en, int lef, int ri) {
        if (st > ri || en < lef) {
            assert(false);
        }
        int mid = (st + en) / 2;
        if (lef <= st && en <= ri) { //totally inside range
            return Tree.at(id);
        }
        pushDown(id,st,en,mid);
        if (lef > mid) //go right, range is to the right
            return query(id*2 + 1, mid + 1, en, lef, ri);
        else if (ri <= mid) // go left
            return query(id*2, st, mid, lef, ri);

        Node lQ = query(id*2, st, mid, lef, ri);
        Node rQ = query(id*2 + 1, mid + 1, en, lef, ri);
        return merge(lQ,rQ);
    }
    //range update with delta
    void update(int id, int st, int en, int lIdx, int rIdx, Long delta) {
        int mid = (st+en) / 2;
        if (st > en) assert(false);
        if (st > rIdx || en < lIdx) return;
        if (lIdx <= st && en <= rIdx) {
            rotate(id);
            Tree.at(id).lazyVal += delta;
            Tree.at(id).lazyVal = 3;
            Tree.at(id).isLazy = true;
            return;
        }
        pushDown(id,st,en,mid);
        update(id*2 + 1, mid + 1, en, lIdx, rIdx, delta);
        update(id*2, st, mid, lIdx, rIdx, delta);
        Tree.at(id) = merge(Tree.at(2*id), Tree.at(2*id + 1));
    }
public:
    SegmentTree(vector<Long> v) {
        V = v;
        Tree = vector<Node>(4*(int)V.size());
        create(1, 0, V.size()-1);
```

```
    }

    Long query(int lef, int ri) {
        return query(1, 0, V.size()-1, lef, ri).a;
    }
    void update(int lIdx, int rIdx, Long delta) {
        update(1,0,V.size()-1, lIdx, rIdx, delta);
    }
};

int main() {
    int n, q;
    scanf("%d%d", &n, &q);
    vector<Long> s(n);
    SegmentTree ST(s);
    for(int i = 0; i < q; i++) {
        int k,l,r;
        scanf("%d%d%d", &k,&l,&r);
        if (!k) {
            ST.update(l,r,1);
        }
        else {
            printf("%d\n", ST.query(l,r));
        }
    }
    return 0;
}
```

6.4 Digit DP chuipi – Numeros entre A y B con no mas de 3 digitos no cero

```
#include <bits/stdc++.h>
using namespace std;
using Long = long long;
Long dp[50][20][5];
vector<Long> sz;
Long go(int pos, int cnt, bool f) {
    if (pos == sz.size()) {
        if (cnt <= 3) return 1LL;
        return 0;
    }
    if (dp[pos][cnt][f] != -1) return dp[pos][cnt][f];
    Long ans = 0, lm;
    if (!f) lm = sz[pos];
    else lm = 9;
    for(int dgt = 0; dgt <= lm; dgt++) {
        bool nf = f;
```

```
        if (!f && dgt < lm) nf = true;
        if (!dgt)
            ans += go(pos + 1, cnt, nf);
        else
            ans += go(pos + 1, cnt + 1, nf);
    }
    return dp[pos][cnt][f] = ans;
}
Long solve(Long k) {
    vector<Long> s;
    while(k > 0) {
        s.push_back(k % 10LL);
        k /= 10LL;
    }
    sz.clear();
    sz = s;
    reverse(sz.begin(), sz.end());
    memset(dp, -1LL, sizeof(dp));
    Long ans = go(0,0,0);
    return ans;
}
int main() {
    int n;
    scanf("%d", &n);
    while(n--) {
        Long l, r;
        cin >> l >> r;
        Long range = r-l+1LL;
        Long AR = solve(r);
        Long AL = solve(l-1LL);
        cout << AR-AL << endl;
    }
    return 0;
}
```

6.5 Double Bounded Digit DP

```
#include <bits/stdc++.h>
using namespace std;
using Long = long long;
string LO, HI;
int k, p;
int dp[2010][2010][2][2];
const Long MOD = (int)(1e9) + 7;

int go(int pos, int m, bool l = 1, bool h = 1) {
    Long ans = 0;
    if (pos >= LO.size()) return !m;
```

```
    if (dp[pos][m][l][h] != -1) return dp[pos][m][l][h];
    int lo = 0, hi = 9;
    if (l) lo = LO[pos]-'0';
    if (h) hi = HI[pos]-'0';
    for(int dgt = lo; dgt <= hi; dgt++) {
        if (pos % 2 == 1 && dgt != p) continue;
        if (pos % 2 == 0 && dgt == p) continue;
        bool nl = l && lo == dgt;
        bool nh = h && hi == dgt;
        ans += go(pos + 1, ((10*m) + dgt) % k, nl, nh);
        ans %= MOD;
    }
    ans %= MOD;
    return dp[pos][m][l][h] = ans;
}

int main() {
    cin >> k >> p;
    cin >> LO >> HI;
    memset(dp, -1, sizeof(dp));
    cout << go(0, 0) << endl;
    return 0;
}
```

6.6 Offline processing Kquery (numeros mayores que tal)

```
#include <bits/stdc++.h>
using namespace std;
#define MOD 1000000007;
using ll = long long;
using pll = pair<ll,ll>;
using VI = vector<int>;
using VL = vector<ll>;
using VVI = vector<VI>;

int N;
struct Query {
    int id, x;
    pair<int,int> r;
    Query (int _id, int _x, pair<int,int> _r) {
        id = _id;
        x = _x;
        r = _r;
    }
    bool operator<(Query o) const {
        return x < o.x;
    }
};
```

```
struct FenwickTree {

    vector<int> tri;
    FenwickTree(int N) : tri(N+10, 0) {}
    void add(int x, int d) {
        for (int i = x + 1; i < tri.size(); i += i&(-i)) {
            tri[i] += d;
            cout << i << " :add : " << tri[i] << endl;
        }
    }

    int sum(int x) {
        int ans = 0;
        for (int i = x + 1; i > 0; i -= i&(-i)) {
            ans += tri[i];
            //cout << i << " get: " << tri[i] << endl;
        }
        return ans;
    }

    void pr() {
        for(int i = 0; i < (int)tri.size(); i++)
            cout << tri[i] << ' ';
        cout << endl;
    }
};


int main() {
    //ios_base::sync_with_stdio(false);
    //cin.tie(NULL);
    scanf("%d", &N);
    vector<pair<int,int> > V(N);
    for(int i = 0; i < N; ++i) {
        scanf("%d", &V[i].first);
        V[i].second = i + 1;
    }
    sort(V.rbegin(), V.rend());
    cout << endl << endl;
    for(int i = 0; i < N; i++) {
        cout << V[i].first << " -> " << V[i].second << endl;
    }
    queue<pair<int,int> > Q;
    for(auto &p : V) Q.push(p);
    int qx;
    scanf("%d", &qx);
    FenwickTree FT(N);
    vector<Query> St;
    ll ANS[200010];
```

```
    for(int i = 0; i < qx; i++) {
        pair<int,int> k;
        int z;
        scanf("%d %d %d", &k.first, &k.second, &z);
        Query q = {i,z,k};
        St.push_back(q);
    }
    sort(St.rbegin(), St.rend());
    for(auto t : St) cout << t.x << " id -> " << t.id << endl;
    for (Query q : St) {
        while(Q.size() && Q.front().first > q.x) {
            cout << "processing query " << q.id << endl;
            FT.add(Q.front().second, 1);
            FT.pr();
            Q.pop();
        }
        //cerr << "ans for range " << q.r.first << ' ' << q.r.second <<
             " " << " : " << FT.sum(q.r.second)- FT.sum(q.r.first) <<
             endl;
        ANS[q.id] = FT.sum(q.r.second) - FT.sum(q.r.first-1);

    }
    for(int i = 0; i < qx; i++)
        printf("%ld\n", ANS[i]);

    return 0;
}
```

6.7 Binary Search anidado – Problema local A triangulo equilatero dado dos distancias

```
#include <bits/stdc++.h>
using namespace std;
double d1,d2;

int main() {
    int t;
    scanf("%d", &t);
    while(t--) {
        scanf("%lf %lf", &d1, &d2);
        double lo = 0, hi = 1e9;
        double r1;
        for(int i = 0; i < 100; i++) {
            double C = (lo+hi) / 2.0; //fix C
            double l = 0, r = 1e9;
            double fixp;
```

```
            for(int j = 0; j < 100; j++) {
                double A = (l+r) / 2.0;
                double AToC = sqrt((d2+d1)*(d2+d1) + (C-A)*(C-A));
                double AtoB = sqrt((-d1)*(-d1) + (A*A));
                //cerr << "chk2 l" << l << " h " << r << " A " << A << "
                    AtoC " << AToC << " AtoB" << AtoB << endl;
                if (AtoB > AToC) {
                    r = A;
                    fixp = AToC;
                }
                else {
                    l = A;
                }
            }
            double CtoB = sqrt(d2*d2 + C*C);
            //cerr << "chck1 " << lo << " hi " << hi << " C " << C <<
                endl;
            if (CtoB > fixp) {
                r1 = fixp;
                hi = C;
            }
            else {
                lo = C;
            }
        }
        double ans = (sqrt(3.00)/4.00) * (r1*r1);
        printf("%.12lf\n", ans);
    }

}
```

6.8 Version Joa 1 BS – BS a la longitud del lado en si

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

int main() {
    int t;
    scanf("%d", &t);
    while(t--) {
        double d1,d2;
        scanf("%lf %lf", &d1, &d2);
        double lo = 0.0, hi = 1e9 + 10.0;
        double len;
        for(int i = 0; i < 100; i++) {
            double clen = (lo+hi) / 2.0;
            double CY = sqrt((clen*clen) - (d2*d2));
            double AY = sqrt((clen*clen) - (d1*d1));
```

```
            double AToC = sqrt((d2-(-d1))*(d2-(-d1)) + (CY-AY)*(CY-AY));
            if (AToC < clen) {
                hi = clen;
                len = clen;
            }
            else {
                lo = clen;
            }
        }
        double ans = (sqrt(3.0)/4.0) * (len*len);
        printf("%.10lf\n", ans);
    }
    return 0;
}
```

6.9 Problema force representacion de numeros de la forma a^b, $a, b > 1$

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
bool isSqrt(ll n) {
    ll sq = sqrt(n) + 1e-9;
    return sq * sq == n;
}
bool isCubic(ll n) {
    ll cub = cbrt(n) + 1e-9;
    return cub * cub * cub == n;
}
bool isFifth(ll n) {
    ll cub = pow(n, 1.0/5) + 1e-9;
    for (int i = max(0LL, cub-2); i <= cub + 2; i++) {
        if (i*i*i*i*i == n) return true;
    }
    return false;
}
bool check(ll n) {
    if (n == 1) return false;
    if (isSqrt(n)) return true;
    if (isCubic(n)) return true;
    if (isFifth(n)) return true;
    for (int i = 2; i*i*i*i*i*i*i <= n; ++i) {
        ll cn = n;
        while (cn % i == 0) cn /= i;
        if (cn == 1) return true;
    }
    return false;
```

```
}

int main() {
    int t;
    cin >> t;
    while(t--) {
        ll x;
        cin >> x;
        int ans = 0;
        for(int i = 1; i*i <= x; i++) {
            if (x % i == 0 && check(i)) ans++;
            if (i*i != x && x % i == 0 && check(x/i)) ans++;
        }
        cout << ans << endl;
    }
    return 0;
}
```

6.10 Ternary Search

```
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);    //evaluates the function at m1
        double f2 = f(m2);    //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x) in [l, r]
}
```