Team Notebook

HUS.Penguin$_OK$

November 12, 2022

Contents

1 DataStructures

1.1 PersistentSegmentTree

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
using namespace std;

#define long long long
#define f1(i,n) for (int i=1; i<=n; i++)
#define f0(i,n) for (int i=0; i<n; i++)

#define N 100005
int m, n, a[N], l[N], Root[N], Peak=0;
int Sum[80*N], Left[80*N], Right[80*N]; // (n*4)+(n log n)

int create(int n){
    if (n==1) { ++Peak; Sum[Peak]=0; return Peak; }
    int u = ++Peak;
    Left[u]=create(n-n/2);
    Right[u]=create(n/2);
    return u;
}

struct node {
    int ll, rr, id;

    node(int L, int R, int X)
    { ll=L, rr=R, id=X; }
    node left()
    { return node(ll, (ll+rr)/2, Left[id]); }
    node right()
    { return node((ll+rr)/2+1, rr, Right[id]); }

    int update(int U, int X){
        if (ll>U || U>rr) return id;
        if (ll==rr) { Sum[++Peak]=X; return Peak; }
        int u = ++Peak;
        Left[u] = left().update(U, X);
        Right[u] = right().update(U, X);
        Sum[u]=Sum[Left[u]]+Sum[Right[u]];
        return u;
    }
    int sum_range(int L, int R){
        if (L>rr || ll>R || L>R) return 0;
        if (L<=ll && rr<=R) return Sum[id];
        int Sum1 = left().sum_range(L, R);
        int Sum2 = right().sum_range(L, R);
        return Sum1 + Sum2;
    }
};

bool as_a(int x, int y)
    { return a[x]<a[y]; }

main(){
    scanf("%d%d", &n, &m);
    f1(i,n) scanf("%d", &a[i]);
//    f1(i,n) printf("%d ", a[i]=rand()%100); printf("\n");

    f1(i,n) l[i]=i;
    sort(l+1, l+n+1, as_a);
    Root[0]=create(n);
    f1(i,n) {
        Root[i]=node(1, n, Root[i-1]).update(l[i], 1);
//        cout << endl << Peak << " " << 80*N << endl;
    }

    f1(i,m) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        int ll=1, rr=n, mm=(ll+rr)/2;
        while (ll!=rr){
            if (node(1, n, Root[mm]).sum_range(x, y)>=z) rr=
                mm; else ll=mm+1;
            mm=(ll+rr)/2;
        }
        printf("%d\n", a[l[mm]]);
    }
}
```

1.2 RMQ

```
// RMQ {{{
//
// Sparse table
// Usage:
// RMQ<int, _min> st(v);
//
// Note:
// - doesn't work for empty range
//
// Tested:
// - https://judge.yosupo.jp/problem/staticrmq

#include <vector>
using namespace std;
template <class T, T *(op)(T, T)> struct RMQ {
  RMQ() = default;
  RMQ(const vector<int> &v) : t{v}, n{(int)v.size()} {
    for (int k = 1; (1 << k) <= n; ++k) {
      t.emplace_back(n - (1 << k) + 1);
      for (int i = 0; i + (1 << k) <= n; ++i) {
        t[k][i] = op(t[k - 1][i], t[k - 1][i + (1 << (k - 1))
            ]);
      }
    }
  }
  // get range [l, r-1]
  // doesn't work for empty range
  T get(int l, int r) const {
    assert(0 <= l && l < r && r <= n);
    int k = __lg(r - l);
    return op(t[k][r - l], t[k][r - (1 << k)]);
  }

private:
  vector<vector<T>> t;
  int n;
};
template <class T> T _min(T a, T b) { return a < b ? a : b;
    }

template <class T> T _max(T a, T b) { return a > b ? a : b;
    }
```

1.3 fenwickTree(BIT)

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

const int N = 1e6 + 5;
const int LOGN = log(N) + 1;
int bit[N];
int a[N];
int n; // n is size of array.

void initialize() { // create bit in O(N)
    for(int i = 1 ; i <= n; ++i) {
        bit[i] += a[i];
        if (i + (i&-i) <= n) bit[i+(i&-i)] += bit[i];
    }
}

void update(int i, int val) {
    for(; i <= n; i += i&(-i))
        bit[i] += val;
```

```
}

int get(int i) {
    int res = 0;
    for(; i > 0; i -= i&(-i))
        res += bit[i];
    return res;
}

int get(int l, int r) {
    return get(r) - get(l-1);
}

int bit_search(int v) {
    int sum = 0;
    int pos = 0;
    for(int i = LOGN; i >= 0; --i) {
        if (pos + (1<<i) < N && sum + bit[pos + (1<<i)] < v)
             {
            pos += 1<<i;
            sum += bit[pos];
        }
    }
    return pos + 1;
    // +1 because 'pos' will have position of largest value
         less than 'v'
}

int main() {

}
```

1.4 fullHash

1.5 orderedSet

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
      tree_order_statistics_node_update>;
```

1.6 pbds-faster-map

```
// From https://codeforces.com/blog/entry/60737

// Code copied from https://codeforces.com/contest/1006/
     submission/41804666
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

unsigned hash_f(unsigned x) {
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = (x >> 16) ^ x;
    return x;
}
struct chash {
    int operator()(int x) const { return hash_f(x); }
};

gp_hash_table<int, int, chash> mp;


// alternative hash function:
// Code copied from https://ideone.com/LhpILA
const ll TIME = chrono::high_resolution_clock::now().
     time_since_epoch().count();
const ll SEED = (ll)(new ll);
const ll RANDOM = TIME ^ SEED;
const ll MOD = (int)1e9+7;
const ll MUL = (int)1e6+3;

struct chash{
    ll operator()(ll x) const { return std::hash<ll>{}((x ^
         RANDOM) % MOD * MUL); }
};
```

1.7 segmentTree-fast

```
#include<cstdio>
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;
}

int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) scanf("%d", t + n + i);
  build();
  modify(0, 1);
  printf("%d\n", query(3, 11));
  return 0;
}
```

1.8 segmentTree-merge

```
// CPP program to implement k-th order statistics
#include <bits/stdc++.h>
using namespace std;

const int MAX = 4e5 + 5; // max size of segtree = 4*N

// Constructs a segment tree and stores tree[]
void buildTree(int treeIndex, int l, int r, vector<pair<int,
      int>> &a,
               vector<int> tree[]) {

  /* l => start of range,
         r => ending of a range
         treeIndex => index in the Segment Tree/Merge
                                 Sort Tree */

  /* leaf node */
  if (l == r) {
    tree[treeIndex].push_back(a[l].second);
    return;
  }

  int mid = (l + r) / 2;

  /* building left subtree */
```

```
  buildTree(2 * treeIndex, l, mid, a, tree);

  /* building left subtree */
  buildTree(2 * treeIndex + 1, mid + 1, r, a, tree);

  /* merging left and right child in sorted order */
  merge(tree[2 * treeIndex].begin(), tree[2 * treeIndex].end
       (),
        tree[2 * treeIndex + 1].begin(), tree[2 * treeIndex +
             1].end(),
        back_inserter(tree[treeIndex]));
}

// Returns the Kth smallest number in query range
int queryRec(int segmentStart, int segmentEnd, int
     queryStart, int queryEnd,
             int treeIndex, int K, vector<int> tree[]) {
  /*
         segmentStart => start of a Segment,
         segmentEnd => ending of a Segment,
         queryStart => start of a query range,
         queryEnd => ending of a query range,
         treeIndex => index in the Segment
                                    Tree/Merge Sort Tree,
         K => kth smallest number to find */

  if (segmentStart == segmentEnd)
    return tree[treeIndex][0];

  int mid = (segmentStart + segmentEnd) / 2;

  // finds the last index in the segment
  // which is <= queryEnd
  int last_in_query_range = (upper_bound(tree[2 * treeIndex
       ].begin(),
                                    tree[2 * treeIndex].end
                                         (), queryEnd) -
                            tree[2 * treeIndex].begin());

  // finds the first index in the segment
  // which is >= queryStart
  int first_in_query_range =
      (lower_bound(tree[2 * treeIndex].begin(), tree[2 *
           treeIndex].end(),
                queryStart) -
       tree[2 * treeIndex].begin());

  int M = last_in_query_range - first_in_query_range;

  if (M >= K) {
    // Kth smallest is in left subtree,
    // so recursively call left subtree for Kth
    // smallest number
    return queryRec(segmentStart, mid, queryStart, queryEnd,
         2 * treeIndex, K,
                    tree);
  }

  else {

    // Kth smallest is in right subtree,
    // so recursively call right subtree for the
    // (K-M)th smallest number
    return queryRec(mid + 1, segmentEnd, queryStart, queryEnd
         ,
                    2 * treeIndex + 1, K - M, tree);
  }
}

// A wrapper over query()
int query(int queryStart, int queryEnd, int K, int n, vector
     <pair<int, int>> &a,
         vector<int> tree[]) {

  return queryRec(0, n - 1, queryStart - 1, queryEnd - 1, 1,
       K, tree);
}

// Driver code
int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  int n, q;
  cin >> n >> q;
  int arr[n];
  for (int i = 0; i < n; ++i)
    cin >> arr[i];
  // vector of pairs of form {element, index}
  vector<pair<int, int>> v;
  for (int i = 0; i < n; i++) {
    v.push_back(make_pair(arr[i], i));
  }

  // sort the vector
  sort(v.begin(), v.end());

  // Construct segment tree in tree[]
  vector<int> tree[MAX];
  buildTree(1, 0, n - 1, v, tree);

  // Answer queries
  // kSmallestIndex hold the index of the kth smallest
       number
  for (int i = 0; i < q; ++i) {
    int l, r;
    cin >> l >> r;
    int mid = (l + r) / 2 - l + 1;
    int median = query(l, r, mid, n, v, tree);
    cout << arr[median] << endl;
  }

  return 0;
}
```

1.9 segmentTree2D

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
using namespace std;

int Max[4096][4096];

struct dir {
    int ll, rr, id;
    dir (int L, int R, int X)
        { ll=L, rr=R, id=X; }
    dir left() const
        { return dir(ll, (ll+rr)/2, id*2); }
    dir right() const
        { return dir((ll+rr)/2+1, rr, id*2+1); }
    inline bool irrelevant(int L, int R) const
        { return ll>R || L>rr || L>R; }
};

void maximize(int &a, int b)
    { a=max(a, b); }

void maximize(const dir &dx, const dir &dy, int x, int y,
     int k, bool only_y) {
    if (dx.irrelevant(x, x) || dy.irrelevant(y, y)) return;
    maximize(Max[dx.id][dy.id], k);
    if (!only_y && dx.ll != dx.rr) {
        maximize(dx.left(), dy, x, y, k, false);
        maximize(dx.right(), dy, x, y, k, false);
    }
    if (dy.ll != dy.rr) {
```

```
        maximize(dx, dy.left(), x, y, k, true);
        maximize(dx, dy.right(), x, y, k, true);
    }
}

int max_range(const dir &dx, const dir &dy, int lx, int rx,
     int ly, int ry) {
    if (dx.irrelevant(lx, rx) || dy.irrelevant(ly, ry))
         return 0;
    if (lx<=dx.ll && dx.rr<=rx) {
        if (ly<=dy.ll && dy.rr<=ry) return Max[dx.id][dy.id];
        int Max1 = max_range(dx, dy.left(), lx, rx, ly, ry);
        int Max2 = max_range(dx, dy.right(), lx, rx, ly, ry);
        return max(Max1, Max2);
    } else {
        int Max1 = max_range(dx.left(), dy, lx, rx, ly, ry);
        int Max2 = max_range(dx.right(), dy, lx, rx, ly, ry);
        return max(Max1, Max2);
    }
}

const int M=100005, N=1003;
int m, k, x[M], y[M], z[M];

main() {
    scanf("%d%d", &m, &k);
    for (int i=1; i<=m; i++)
    scanf("%d%d%d", &x[i], &y[i], &z[i]);

    dir dx(0, N+N, 1), dy(0, N+N, 1);
    for (int i=m; i>=1; i--) {
        #define actual(x, y, k) x+y-k, x+y+k, x-y-k+N, x-y+k+
             N
        int F = max_range(dx, dy, actual(x[i], y[i], k)) + z[
             i];
        maximize(dx, dy, x[i]+y[i], x[i]-y[i]+N, F, false);
    }
    cout << max_range(dx, dy, actual(0, 0, k)) << endl;
}
```

1.10 treap

```
#include <bits/stdc++.h>
#define elif else if
#define NIL &leaf
using namespace std;
const int INF=1<<30;
struct node {
    node *p, *l,*r;
    int key,pr;
    };
node *root,leaf;
node* newnode(node* parent,int key) {
    node *x=new node;
    x->p=parent;
    x->l=x->r=NIL;
    x->pr=rand();
    x->key=key;
    if(parent!=NIL) {
        if(parent->key>key) parent->l=x;
        else parent->r=x;
        }
    else root=x;
    return x;
    }
void init() {
    leaf.l=leaf.r=leaf.p=NIL;
    leaf.key=-INF;
    leaf.pr=-1;
    root=NIL;
    }
void link(node* x,node* y) {
    if(y==root) root=x;
    elif(y==y->p->l) y->p->l=x;
    else y->p->r=x;
    x->p=y->p;
    y->p=x;
    }
void uptree(node* x) {
    node *parent=x->p;
    link(x,parent);
    if(x==parent->l) {
        parent->l=x->r;
        if(parent->l!=NIL) parent->l->p=parent;
        x->r=parent;
        }
    else {
        parent->r=x->l;
        if(parent->r!=NIL) parent->r->p=parent;
        x->l=parent;
        }
    }
void insert(int key) {
    node* x=root,*parent=NIL;
    while(x!=NIL) {
        parent=x;
        if(key==x->key) return;
        if(key<x->key) x=x->l;
        else x=x->r;
        }
    x=newnode(parent,key);

    while(x!=root&&x->pr>x->p->pr) {
        uptree(x);
        }
    }
node* find(int key) {
    node* x=root;
    while(x!=NIL) {
        if(key==x->key) return x;
        if(key<x->key) x=x->l;
        else x=x->r;
        }
    return NIL;
    }
void delall(node* x) {
    if(x==root) root=NIL;
    elif(x==x->p->l) x->p->l=NIL;
    else x->p->r=NIL;
    }
bool del(int key) {
    node* x=find(key);
    if(x==NIL) return false;
    while(x->l!=NIL&&x->r!=NIL) {
        if(x->l->pr>x->r->pr) uptree(x->l);
        else uptree(x->r);
        }
    if(x->l!=NIL) link(x->l,x);
    elif(x->r!=NIL) link(x->r,x);
    else delall(x);
    free(x);
    return true;
    }
node* Min() {
    node* x=root;
    while(x->l!=NIL) {
        x=x->l;
        }
    return x;
    }
node* Max() {
    node* x=root;
    while(x->r!=NIL) {
        x=x->r;
        }
    return x;
    }
node* succ(int key) {
    node* ans=NIL;
```

```
    node* x=root;
    while(x!=NIL) {
        if(key<x->key) {
            ans=x;
            x=x->l;
            }
        else x=x->r;
        }
    return ans;
    }
node* pred(int key) {
    node* ans=NIL;
    node* x=root;
    while(x!=NIL) {
        if(key>x->key) {
            ans=x;
            x=x->r;
            }
        else x=x->l;
        }
    return ans;
    }
char sss[30];
void dfs(node *x) {
    if(x==NIL) return ;
    printf("%d->>>",x->key);
    dfs(x->l);
    printf("||||| %d->>>",x->key);
    dfs(x->r);
    }
main() {
    //freopen("out","w",stdout);
    ios_base::sync_with_stdio(false);
    srand(time(NULL));
    init();
    int n;
    int x;
    while(cin>>n&&n) {
      // dfs(root);
       // puts("");
        if(n==1) {
            cin>>x;
            insert(x);
            }
        elif(n==2) {
            cin>>x;
            del(x);
            }
        elif(n==3) {
            if(root==NIL) {
                puts("empty");
                continue;
                }
            printf("%d\n",Min()->key);
            }
        elif(n==4) {

            if(root==NIL) {
                puts("empty");
                continue;
                }
            printf("%d\n",Max()->key);
            }
        elif(n==5) {
            cin>>x;

            if(root==NIL) {
                puts("empty");
                continue;
                }
            node* f=succ(x);
            if(f!=NIL) printf("%d\n",f->key);
            else puts("no");
            }
        elif(n==6) {
            cin>>x;

            if(root==NIL) {
                puts("empty");
                continue;
                }
            node* f=find(x);
            if(f==NIL) f=succ(x);
            if(f!=NIL) printf("%d\n",f->key);
            else puts("no");
            }
        elif(n==7) {
            cin>>x;

            if(root==NIL) {
                puts("empty");
                continue;
                }
            node* f=pred(x);
            if(f!=NIL) printf("%d\n",f->key);
            else puts("no");

            }
        elif(n==8) {
            cin>>x;

            if(root==NIL) {
                puts("empty");
                continue;
                }
            node* f=find(x);
            if(f==NIL) f=pred(x);
            if(f!=NIL) printf("%d\n",f->key);
            else puts("no");
            }
        }

    }

// https://sites.google.com/site/kc97ble/container/treap-cpp
```

2 Geometry

2.1 smallestEnclosingClosure

```
// Smallest enclosing circle:
// Given N points. Find the smallest circle enclosing these
    points.
// Amortized complexity: O(N)
//
// Tested:
// - https://www.spoj.com/problems/ALIENS/
// - https://www.spoj.com/problems/QCJ4/
// - https://www.acmicpc.net/problem/2626
// - https://oj.vnoi.info/problem/icpc22_mt_l

struct SmallestEnclosingCircle {
    Circle getCircle(vector<Point> points) {
        assert(!points.empty());

        random_shuffle(points.begin(), points.end());
        Circle c(points[0], 0);
        int n = points.size();

        for (int i = 1; i < n; i++)
            if ((points[i] - c).len() > c.r + EPS)
            {
                c = Circle(points[i], 0);
                for (int j = 0; j < i; j++)
                    if ((points[j] - c).len() > c.r + EPS)
                    {
```

```
                    c = Circle((points[i] + points[j]) / 2,
                          (points[i] - points[j]).len() /
                         2);
                    for (int k = 0; k < j; k++)
                        if ((points[k] - c).len() > c.r +
                             EPS)
                            c = getCircumcircle(points[i],
                                 points[j], points[k]);
                }
            }

        return c;
    }

    // NOTE: This code work only when a, b, c are not
         collinear and no 2 points are same --> DO NOT
    // copy and use in other cases.
    Circle getCircumcircle(Point a, Point b, Point c) {
        assert(a != b && b != c && a != c);
        assert(ccw(a, b, c));

        double d = 2.0 * (a.x * (b.y - c.y) + b.x * (c.y - a.
             y) + c.x * (a.y - b.y));
        assert(fabs(d) > EPS);
        double x = (a.norm() * (b.y - c.y) + b.norm() * (c.y
             - a.y) + c.norm() * (a.y - b.y)) / d;
        double y = (a.norm() * (c.x - b.x) + b.norm() * (a.x
             - c.x) + c.norm() * (b.x - a.x)) / d;
        Point p(x, y);
        return Circle(p, (p - a).len());
    }
};
```

3 Graph

3.1 BellmanFord

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

#define ar array
#define vt vector
#define all(v) (v).begin(), (v).end()
#define pb push_back
#define ll long long
#define ld long double
#define ii pair<int, int>
#define iii pair<int, ii>
#define fi first
#define se second
#define FORIT(i, s) for (auto it=(s.begin()); it!=(s.end());
     ++it)
#define F_OR(i, a, b, s) for (int i=(a); (s)>0? i<(b) : i>(b
    ); i+=(s))
#define F_OR1(n) F_OR(i, 0, n, 1)
#define F_OR2(i, e) F_OR(i, 0, e, 1)
#define F_OR3(i, b, e) F_OR(i, b, e, 1)
#define F_OR4(i, b, e, s) F_OR(i, b, e, s)
#define GET5(a, b, c, d, e, ...) e
#define F_ORC(...) GET5(__VA_ARGS__, F_OR4, F_OR3, F_OR2,
    F_OR1)
#define FOR(...) F_ORC(__VA_ARGS__)(__VA_ARGS__)
#define EACH(x, a) for(auto& x: a)

const int d4x[] = {-1, 0, 1, 0},
        d4y[] = {0, -1, 0, 1},
        d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
        d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1},
        N = 2e5+1;
const ll oo = LLONG_MAX;
int n, // number of vertices
    m, // number of edges
    s, // start vertex
    e; // end vertex
vt<vt<ii>> G; // adjancency list of edge, G is directed
    weighted graph
ll d[N]; // distance from s_v to e_v

void bellmanFord(vt<vt<ii>> G){
    fill_n(d, sizeof(d)/sizeof(d[0]), oo);
    d[s]=0;
    FOR(u, 1, n+1){
        EACH(e, G[u]){
            int v(e.se), uv(e.fi);
            d[v]=min(d[v], d[u]+uv);
        }
    }
    FOR(u, 1, n+1){
        EACH(e, G[u]){
            int v(e.se), uv(e.fi);
            if (d[v]>d[u]+uv) d[v]=-oo; // v is in a negative
                 cycle
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);

    cin >> n >> m;
    G = vt<vt<ii>>(n+1);
    FOR(m){
        int u, v, w;
        cin >> u >> v >> w;
        G[u].pb(ii(w, v));
    }
    cin >> s >> e;
    bellmanFord(G);
    cout << d[e];
}
```

3.2 Dijkstra

```
const int oo = 1e9;
vvii G;
vi d;

void dijkstra(){
    int n = G.size();
    priority_queue<ii, vii, greater<ii>> pq;
    pb.push(ii(0, 1));

    while(pq.size()){
        int u = pq.top().se,
            du = pq.top().fi;
        pq.pop();
        if (du!=d[u]) continue;

        FOR(i, G[u].size()){
            int v = G[u][i].se,
                uv = G[u][i].fi;
            if (d[v] > du+uv) d[v] = du+uv, pq.push({d[v], v
                 });
        }
    }
}

int main(){
```

```
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);

    int n, m;
    G = vvii(n+1);
    d = vi(n+1, oo);
    while(m--){
        int u, v, w;
        cin >> u >> v >> w;
        G[u].pb({w, v});
        G[v].pb({w, u});
    }
    dijkstra();
}
```

3.3 FloydWarshall

```
const int d4x[] = {-1, 0, 1, 0},
          d4y[] = {0, -1, 0, 1},
          d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
          d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1},
          N = 2e3+1,
          oo = 1e9;
int n, // number of vertices
    m, // number of edges
    s, // start vertex
    e; // end vertex
ll d[N][N]; // distance from x to y

void floydWarshall(){
    FOR(i, 1, n+1){
        FOR(j, 1, n+1){
            FOR(k, 1, n+1){
                d[i][j] = min(d[i][j], d[i][k]+d[k][j]);
            }
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);

    cin >> n >> m;
    FOR(i, 1, n+1){
        FOR(j, 1, n+1) d[i][j] = oo;
    }
    FOR(i, 1, n+1) d[i][i]=0;
    FOR(m){
        int u, v, w;
        cin >> u >> v >> w;
        d[u][v] = w;
    }
    cin >> s >> e;
    floydWarshall();
    cout << d[s][e];
}
```

3.4 MinMaxPathDAG

```
const ll oo = LLONG_MAX;
int n, m, cnt;
vt<vt<ii>> G; // G is a DAG
vt<bool> vs;
vt<int> topo;
vt<ll> d;

void dfs(int u){
    vs[u]=true;
    // cout << u << '\n';
    EACH(e, G[u]){
        int w(e.fi), v(e.se);
        if (!vs[v]){
            dfs(v);
        }
    }
    topo[--cnt]=u;
}

void topoSort(){
    topo = vt<int>(n, 0);
    cnt = n;
    vs = vt<bool>(n+1, false);
    FOR(i, 1, n+1){
        if (!vs[i]) dfs(i);
    }
}

ll shortestPathDAG(vt<vt<ii>> G){
    vt<ll> d(n+1, oo);
    d[1] = 0LL;
    EACH(u, topo){
        EACH(e, G[u]){
            int w(e.fi), v(e.se);
            d[v] = min(d[v], d[u]+w);
        }
    }
    return d[n];
}

ll longestPathDAG(vt<vt<ii>> G){
    vt<vt<ii>> G_ = G;
    FOR(i, 1, n+1){
        EACH(e, G_[i])
            e.fi*=-1;
    }
    return -1*shortestPathDAG(G_);
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);

    cin >> n >> m;
    G = vt<vt<ii>>(n+1);
    vs = vt<bool>(n+1, false);
    FOR(m){
        int u, v, w;
        cin >> u >> v >> w;
        G[u].pb(ii(w, v));
    }
    topoSort();
    cout << shortestPathDAG(G) << " " << longestPathDAG(G);
}
```

3.5 Tarjan

```
#include <stdio.h>

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

```cpp
const int N = 100005;
const int oo = 0x3c3c3c3c;

int n, m, Num[N], Low[N], cnt = 0;
vector<int> a[N];
stack<int> st;
int Count = 0;

void visit(int u) {
    Low[u] = Num[u] = ++cnt;
    st.push(u);

    for (int v : a[u])
        if (Num[v])
            Low[u] = min(Low[u], Num[v]);
        else {
            visit(v);
            Low[u] = min(Low[u], Low[v]);
        }

    if (Num[u] == Low[u]) { // found one
        Count++;
        int v;
        do {
            v = st.top();
            st.pop();
            Num[v] = Low[v] = oo; // remove v from graph
        } while (v != u);
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        a[x].push_back(y);
    }

    for (int i = 1; i <= n; i++)
        if (!Num[i]) visit(i);

    cout << Count << endl;
}
```

3.6 TopologicalSort

```cpp
const int d4x[] = {-1, 0, 1, 0},
          d4y[] = {0, -1, 0, 1},
          d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
          d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1};
int n, m, cnt;
vt<int> res;
vt<vt<int>> G;
vt<bool> vs;

void dfs(int u){
    EACH(v, G[u]){
        if (!vs[v]){
            vs[v] = true;
            dfs(v);
        }
    }
    res[--cnt]=u;
}

void topoSort(){
    res = vt<int>(n);
    cnt=n;
    vs = vt<bool>(n+1, false);
    FOR(i, 1, n+1){
        if (!vs[i]){
            vs[i]=true;
            dfs(i);
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    freopen("test.inp", "r", stdin);
    freopen("test.out", "w", stdout);

    cin >> n >> m;
    G = vt<vt<int>>(n+1);
    FOR(m){
        int u, v;
        cin >> u >> v;
        G[u].pb(v);
    }
    topoSort();
    EACH(u, res) cout << u << '\n';

}
```

3.7 dfsTree

3.7.1 Biconnected-component

```cpp
// Input graph: vector< vector<int> > a, int n
// Note: 0-indexed
// Usage: BiconnectedComponent bc; (bc.components is the
    list of components)
//
// This is biconnected components by edges (1 vertex can
    belong to
// multiple components). For vertices biconnected component,
     remove
// bridges and find components
int n;
vector<vector<int>> g;
struct BiconnectedComponent {
    vector<int> low, num, s;
    vector< vector<int> > components;
    int counter;

    BiconnectedComponent() : low(n, -1), num(n, -1), counter
        (0) {
        for (int i = 0; i < n; i++)
            if (num[i] < 0)
                dfs(i, 1);
    }

    void dfs(int x, int isRoot) {
        low[x] = num[x] = ++counter;
        if (g[x].empty()) {
            components.push_back(vector<int>(1, x));
            return;
        }
        s.push_back(x);

        for (int i = 0; i < (int) g[x].size(); i++) {
            int y = g[x][i];
            if (num[y] > -1) low[x] = min(low[x], num[y]);
            else {
                dfs(y, 0);
                low[x] = min(low[x], low[y]);

                if (isRoot || low[y] >= num[x]) {
                    components.push_back(vector<int>(1, x));
                    while (1) {
                        int u = s.back();
                        s.pop_back();
                        components.back().push_back(u);
                        if (u == y) break;
```

```
                }
            }
        }
    }
};
```

3.7.2 BridgeArticulation

```
const int d4x[] = {-1, 0, 1, 0},
          d4y[] = {0, -1, 0, 1},
          d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
          d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1},
          N = 2e5+1;
int n, m, cnt, low[N], num[N], numChild[N];
bool isArt[N];
vt<vt<int>> G;
vt<ii> bridges;

void dfs(int u, int p){
    low[u] = num[u] = ++cnt;
    EACH(v, G[u]){
        if (!num[v]){
            ++numChild[u];
            dfs(v, u);
            low[u] = min(low[u], low[v]);
        } else {
            if (v != p){
                low[u] = min(low[u], num[v]);
            }
        }
        if (low[u]==num[u]){
            if (numChild[u]>1) isArt[u]=true;
        }
        if (num[u]<low[v]){
            bridges.pb({u, v});
            isArt[u]=true;
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);
    cin >> n >> m;
    G = vt<vt<int>>(n+1);
    FOR(m){
        int u, v;
        cin >> u >> v;
        G[u].pb(v);
        // G[v].pb(u); // if undirected
    }
    FOR(i, 1, n+1) if (!num[i]) dfs(i, -1);
    FOR(i, 1, n+1) cout << i << " " << num[i] << " " << low[i
        ] << '\n';
    EACH(b, bridges) cout << b.fi << " " << b.se << '\n';
    FOR(i,1 , n+1) if (isArt[i]) cout << i << '\n';
}
```

3.7.3 StronglyConnected

```
// Index from 0
// Usage:
// DirectedDfs tree;
// Now you can use tree.scc
//
// Note: reverse(tree.scc) is topo sorted
//
// Tested:
// - (requires scc to be topo sorted) https://judge.yosupo.
    jp/problem/scc
// - https://cses.fi/problemset/task/1686/
struct DirectedDfs {
    vector<vector<int>> g;
    int n;
    vector<int> num, low, current, S;
    int counter;
    vector<int> comp_ids;
    vector< vector<int> > scc;

    DirectedDfs(const vector<vector<int>>& _g) : g(_g), n(g.
        size()),
          num(n, -1), low(n, 0), current(n, 0), counter(0),
                comp_ids(n, -1) {
        for (int i = 0; i < n; i++) {
            if (num[i] == -1) dfs(i);
        }
    }

    void dfs(int u) {
        low[u] = num[u] = counter++;
        S.push_back(u);
        current[u] = 1;
        for (auto v : g[u]) {
            if (num[v] == -1) dfs(v);
            if (current[v]) low[u] = min(low[u], low[v]);
        }
        if (low[u] == num[u]) {
            scc.push_back(vector<int>());
            while (1) {
                int v = S.back(); S.pop_back(); current[v] =
                    0;
                scc.back().push_back(v);
                comp_ids[v] = ((int) scc.size()) - 1;
                if (u == v) break;
            }
        }
    }

    // build DAG of strongly connected components
    // Returns: adjacency list of DAG
    std::vector<std::vector<int>> build_scc_dag() {
        std::vector<std::vector<int>> dag(scc.size());
        for (int u = 0; u < n; u++) {
            int x = comp_ids[u];
            for (int v : g[u]) {
                int y = comp_ids[v];
                if (x != y) {
                    dag[x].push_back(y);
                }
            }
        }
        return dag;
    }
};
```

3.8 heavylight-adamat

```
// HeavyLight {{{
// Index from 0
// Best used with SegTree.h
//
// Usage:
// HLD hld(g, root);
// // build segment tree. Note that we must use hld.order[i]
// vector<T> nodes;
// for (int i = 0; i < n; i++)
//   nodes.push_back(initial_value[hld.order[i]])
// SegTree<S, op, e> st(nodes);
//
// // Update path
```

```
// hld.apply_path(from, to, is_edge_or_vertex, [&] (int l,
    int r) {
//   st.apply(l, r+1, F);
// });
//
// // Query path
// hld.prod_path_commutative<S, op, e> (from, to,
    is_edge_or_vertex, [&] (int l, int r) {
//   return st.prod(l, r+1);
// });
//
// Tested:
// - (vertex, path) https://judge.yosupo.jp/problem/
    vertex_add_path_sum
// - (vertex, path, non-commutative) https://judge.yosupo.jp
    /problem/vertex_set_path_composite
// - (vertex, subtree) https://judge.yosupo.jp/problem/
    vertex_add_subtree_sum
// - (vertex, path, non-commutative, 1-index) https://oj.
    vnoi.info/problem/icpc21_mt_l
// - (vertex, path) https://oj.vnoi.info/problem/qtree3
//
// - (edge, path) https://oj.vnoi.info/problem/qtreex
// - (edge, path) https://oj.vnoi.info/problem/lubenica
// - (edge, path) https://oj.vnoi.info/problem/pwalk
// - (edge, path, lazy) https://oj.vnoi.info/problem/kbuild
// - (edge, path, lazy) https://oj.vnoi.info/problem/
    onbridge
//
// - (lca) https://oj.vnoi.info/problem/fselect
// - (kth_parent) https://cses.fi/problemset/task/1687
#include<bits/stdc++.h>
using namespace std;
struct HLD {
    HLD(const vector<vector<int>>& _g, int root)
            : n(_g.size()), g(_g),
            parent(n), depth(n), sz(n),
            dfs_number(0), nxt(n), in(n), out(n), order(n)
    {
        assert(0 <= root && root < n);

        // init parent, depth, sz
        // also move most heavy child of u to g[u][0]
        depth[root] = 0;
        dfs_sz(root, -1);

        // init nxt, in, out
        nxt[root] = root;
        dfs_hld(root);
    }

    int lca(int u, int v) const {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        while (true) {
            if (in[u] > in[v]) swap(u, v); // in[u] <= in[v]
            if (nxt[u] == nxt[v]) return u;
            v = parent[nxt[v]];
        }
    }

    // return k-th parent
    // if no such parent -> return -1
    int kth_parent(int u, int k) const {
        assert(0 <= u && u < n);
        if (depth[u] < k) return -1;

        while (true) {
            int v = nxt[u];
            if (in[u] - k >= in[v]) return order[in[u] - k];
            k -= in[u] - in[v] + 1;
            u = parent[v];
        }
    }

    // return k-th vertex on path from u -> v (0 <= k)
    // if k > distance -> return -1
    int kth_vertex_on_path(int u, int v, int k) const {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);

        int l = lca(u, v);
        int ul = depth[u] - depth[l];
        if (k <= ul) return kth_parent(u, k);
        k -= ul;
        int vl = depth[v] - depth[l];
        if (k <= vl) return kth_parent(v, vl - k);
        return -1;
    }

    int dist(int u, int v) const {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        int l = lca(u, v);
        return depth[u] + depth[v] - 2*depth[l];
    }

    // apply f on vertices on path [u, v]
    // edge = true -> apply on edge
    //
    // f(l, r) should update segment tree [l, r] INCLUSIVE
    void apply_path(int u, int v, bool edge, const function<
        void(int, int)> &f) {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        if (u == v && edge) return;

        while (true) {
            if (in[u] > in[v]) swap(u, v); // in[u] <= in[v]
            if (nxt[u] == nxt[v]) break;
            f(in[nxt[v]], in[v]);
            v = parent[nxt[v]];
        }
        if (u == v && edge) return;
        f(in[u] + edge, in[v]);
    }

    // get prod of path u -> v
    // edge = true -> get on edges
    //
    // f(l, r) should query segment tree [l, r] INCLUSIVE
    // f must be commutative. For non-commutative, use
        getSegments below
    template<class S, S (*op) (S, S), S (*e)()>
    S prod_path_commutative(
            int u, int v, bool edge,
            const function<S(int, int)>& f) const {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        if (u == v && edge) {
            return e();
        }
        S su = e(), sv = e();
        while (true) {
            if (in[u] > in[v]) { swap(u, v); swap(su, sv); }
            if (nxt[u] == nxt[v]) break;
            sv = op(sv, f(in[nxt[v]], in[v]));
            v = parent[nxt[v]];
        }
        if (u == v && edge) {
            return op(su, sv);
        } else {
            return op(su, op(sv, f(in[u] + edge, in[v])));
        }
    }

    // f(l, r) modify seg_tree [l, r] INCLUSIVE
    void apply_subtree(int u, bool edge, const function<void(
        int, int)>& f) {
        assert(0 <= u && u < n);
```

```
        f(in[u] + edge, out[u] - 1);
    }

    // f(l, r) queries seg_tree [l, r] INCLUSIVE
    template<class S>
    S prod_subtree_commutative(int u, bool edge, const
         function<S(S, S)>& f) {
        assert(0 <= u && u < n);
        return f(in[u] + edge, out[u] - 1);
    }

    // Useful when functions are non-commutative
    // Return all segments on path from u -> v
    // For this problem, the order (u -> v is different from
         v -> u)
    vector< pair<int,int> > getSegments(int u, int v) const {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        vector< pair<int,int> > upFromU, upFromV;

        int fu = nxt[u], fv = nxt[v];
        while (fu != fv) { // u and v are on different chains
            if (depth[fu] >= depth[fv]) { // move u up
                upFromU.push_back({u, fu});
                u = parent[fu];
                fu = nxt[u];
            } else { // move v up
                upFromV.push_back({fv, v});
                v = parent[fv];
                fv = nxt[v];
            }
        }
        upFromU.push_back({u, v});
        reverse(upFromV.begin(), upFromV.end());
        upFromU.insert(upFromU.end(), upFromV.begin(),
             upFromV.end());
        return upFromU;
    }

    // return true if u is ancestor
    bool isAncestor(int u, int v) {
        return in[u] <= in[v] && out[v] <= out[u];
    }

// private:
    int n;
    vector<vector<int>> g;
    vector<int> parent; // par[u] = parent of u. par[root] =
         -1
    vector<int> depth;  // depth[u] = distance from root -> u
    vector<int> sz;     // sz[u] = size of subtree rooted at
         u
    int dfs_number;
    vector<int> nxt;    // nxt[u] = vertex on heavy path of u
         , nearest to root
    vector<int> in, out; // subtree(u) is in range [in[u],
         out[u]-1]
    vector<int> order;  // euler tour

    void dfs_sz(int u, int fu) {
        parent[u] = fu;
        sz[u] = 1;
        // remove parent from adjacency list
        auto it = std::find(g[u].begin(), g[u].end(), fu);
        if (it != g[u].end()) g[u].erase(it);

        for (int& v : g[u]) {
            depth[v] = depth[u] + 1;
            dfs_sz(v, u);

            sz[u] += sz[v];
            if (sz[v] > sz[g[u][0]]) swap(v, g[u][0]);
        }
    }

    void dfs_hld(int u) {
        order[dfs_number] = u;
        in[u] = dfs_number++;

        for (int v : g[u]) {
            nxt[v] = (v == g[u][0] ? nxt[u] : v);
            dfs_hld(v);
        }
        out[u] = dfs_number;
    }
};
// }}}
int main() {

}
```

4 Math

4.1 BigNum

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

using namespace std;

#define vt vector
#define pb push_back
#define ll long long
#define vi vt<int>
#define FORIT(i, s) for (auto it=(s.begin()); it!=(s.end());
     ++it)
#define F_OR(i, a, b, s) for (int i=(a); (s)>0? i<(b) : i>(b
    ); i+=(s))
#define F_OR1(n) F_OR(i, 0, n, 1)
#define F_OR2(i, e) F_OR(i, 0, e, 1)
#define F_OR3(i, b, e) F_OR(i, b, e, 1)
#define F_OR4(i, b, e, s) F_OR(i, b, e, s)
#define GET5(a, b, c, d, e, ...) e
#define F_ORC(...) GET5(__VA_ARGS__, F_OR4, F_OR3, F_OR2,
    F_OR1)
#define FOR(...) F_ORC(__VA_ARGS__)(__VA_ARGS__)
#define EACH(x, a) for(auto& x: a)

/*
   Treat BigNum as a vector<int>, char from _size-1 -> 0.
   BASE: each character of BigNum is a integer in range [0,
        BASE).
*/
const int BASE = 1e5;

void fix(vi &x){
    /*
       fix to the right form after operator
    */
    x.pb(0);
    FOR(x.size()-1){
        x[i+1] += x[i]/BASE;
        x[i] %= BASE;
        if (x[i]<0) x[i]+=BASE, --x[i+1];
    }
    while(x.size()>1 && !x.back()) x.pop_back();
}

vi operator + (vi x, const vi &y){
    x.resize(max(x.size(), y.size()));
    FOR(y.size()) x[i] += y[i];
    return fix(x), x;
}

vi operator - (vi x, const vi &y){
    x.resize(max(x.size(), y.size()));
    FOR(y.size()) x[i] -= y[i];
    return fix(x), x;
}
```

```
vi operator * (vi x, int k){
    assert(k<BASE);
    EACH(xi, x) xi *= k;
    return fix(x), x;
}

vi operator * (const vi &x, const vi &y){
    vi z(x.size()+y.size()+1);
    FOR(x.size()) FOR(j, y.size()){
        z[i+j] += x[i]*y[j];
        z[i+j+1] += z[i+j]/BASE;
        z[i+j] %= BASE;
    }
    return fix(z), z;
}

vi operator / (vi x, int k){
    assert(k<BASE);
    for(int i=x.size()-1, r=0; i>=0; --i) r=r*BASE+x[i], x[i
        ]=r/k;
    return fix(x), x;
}

bool operator < (const vi &x, const vi &y){
    if (x.size()!=y.size()) return x.size()<y.size();
    FOR(i, x.size()-1, -1, -1) if (x[i]!=y[i]) return x[i]<y[
        i];
    return false;
}

istream &operator>>(istream &cin, vi &a) {
    string s;
    cin >> s;
    a.clear();
    a.resize(s.size()/4+1);
    FOR(s.size()){
        int x = (s.size()-1-i)/4; // <- log10(BASE)=4
        a[x] = a[x]*10+(s[i]-'0');
    }
    return fix(a), cin;
}

ostream &operator<<(ostream &cout, const vi &a) {
    printf("%d", a.back());
    FOR(i, a.size()-2, -1, -1) printf("%04d", a[i]);
    return cout;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);
}
```

4.2 euler-totient

```
void phi_1_to_n(int n) {
    vector<int> phi(n+1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; ++i) {
        phi[i] = i-1;
    }
    for(int i = 2; i <= n; ++i) {
        for(int j = 2*i; j <= n; j+=i) {
            phi[j] -= phi[i];
        }
    }
}

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

4.3 matrix

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

using namespace std;

const int mod = 111539786;

using type = int;

struct Matrix {
    vector <vector <type> > data;

    int row() const { return data.size(); }

    int col() const { return data[0].size(); }

    auto & operator [] (int i) { return data[i]; }

    const auto & operator[] (int i) const { return data[i]; }

    Matrix() = default;

    Matrix(int r, int c): data(r, vector <type> (c)) { }

    Matrix(const vector <vector <type> > &d): data(d) { }

    friend ostream & operator << (ostream &out, const Matrix
        &d) {
        for (auto x : d.data) {
            for (auto y : x) out << y << ' ';
            out << '\n';
        }
        return out;
    }

    static Matrix identity(long long n) {
        Matrix a = Matrix(n, n);
        while (n--) a[n][n] = 1;
        return a;
    }

    Matrix operator * (const Matrix &b) {
        Matrix a = *this;
        assert(a.col() == b.row());
        Matrix c(a.row(), b.col());
        for (int i = 0; i < a.row(); ++i)
            for (int j = 0; j < b.col(); ++j)
                for (int k = 0; k < a.col(); ++k){
                    c[i][j] += 1ll * a[i][k] % mod * (b[k][j]
                        % mod) % mod;
                    c[i][j] %= mod;
                }
        return c;
    }

    Matrix pow(long long exp) {
```

```
        assert(row() == col());
        Matrix base = *this, ans = identity(row());
        for (; exp > 0; exp >>= 1, base = base * base)
            if (exp & 1) ans = ans * base;
        return ans;
    }
};

int main(){
    Matrix a({
        {1, 1},
        {1, 0}
    });

    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        Matrix tmp = a.pow(n - 1);
        cout << (tmp[0][0] + tmp[0][1]) % mod << '\n';
    }
}
```

4.4 miller

```
pair<ll, ll> factor(ll n)
{
 ll s = 0;
 while ((n & 1) == 0)
 {
  s++;
  n >>= 1;
 }
 return {s, n};
}
ll pow(ll a, ll d, ll n)
{
 ll r = 1;
 a = a % n;
 while (d > 0)
 {
  if (d & 1)
   r = (r * a) % n;
  d >>= 1;
  a = (a * a) % n;
 }
 return r;
}
bool test_a(ll s, ll d, ll n, ll a)
{
 if (n == a)
  return true;
 ll p = pow(a, d, n);
 if (p == 1)
  return true;
 for (; s > 0; s--)
 {
  if (p == n - 1)
   return true;
  p = p * p % n;
 }
 return false;
}
bool miller(ll n)
{
 if (n < 2)
  return false;
 if ((n & 1) == 0)
  return n == 2;
 ll s, d;
 tie(s, d) = factor(n - 1);

 if (n < 1373653)
 {
  return test_a(s, d, n, 2) && test_a(s, d, n, 3);
 }
 else if (n < 4759123141)
 {
  return test_a(s, d, n, 2) && test_a(s, d, n, 7) && test_a(
      s, d, n, 61);
 }
 else
 {
  return test_a(s, d, n, 2) && test_a(s, d, n, 3) && test_a(
      s, d, n, 5) && test_a(s, d, n, 7) && test_a(s, d, n,
      11) && test_a(s, d, n, 13) && test_a(s, d, n, 17) &&
      test_a(s, d, n, 19) && test_a(s, d, n, 23);
 }
}
```

4.5 primeFactor

```
vector<long long> trial_division3(long long n) {
    vector<long long> factorization;
    for (int d : {2, 3, 5}) {
        while (n % d == 0) {
            factorization.push_back(d);
            n /= d;
        }
    }
    static array<int, 8> increments = {4, 2, 4, 2, 4, 6, 2,
         6};
    int i = 0;
    for (long long d = 7; d * d <= n; d += increments[i++]) {
        while (n % d == 0) {
            factorization.push_back(d);
            n /= d;
        }
        if (i == 8)
            i = 0;
    }
    if (n > 1)
        factorization.push_back(n);
    return factorization;
}
```

4.6 rabin

```
template <typename _Tp>
int rabin(_Tp n)
{
 if (n == 2)
  return 1;
 if (n < 2 || !(n & 1))
  return 0;
 const _Tp p[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
 _Tp a, d = n - 1, mx = 4;
 int i, r, s = 0;
 while (!(d & 1))
 {
  ++s;
  d >>= 1;
 }
 for (i = 0; i < mx; i++)
 {
  if (n == p[i])
   return 1;
  if (!(n % p[i]))
   return 0;
  a = powmod(p[i], d, n);
  if (a != 1)
  {
   for (r = 0; r < s && a != n - 1; r++)
    a = mulmod(a, a, n);
   if (r == s)
    return 0;
```

```
 }
 }
 return 1;
}
```

5 String

5.1 Aho-Corasick

```
// Tested:
// - https://open.kattis.com/problems/stringmultimatching
// - https://icpc.kattis.com/problems/firstofhername
// - https://oj.vnoi.info/problem/binpal
//
// Notes:
// - Node IDs from 0 to aho.sz.
// - Characters should be normalized to [0, MC-1].
// - For each node of AhoCorasick, we store a linked list
     containing all queries "associated" with this node.
//   The reason is that, when we reach a node in AhoCorasick,
      it's possible to match several queries at once.
//   (this happens when queries are suffix of others, e.g. C,
      BC, ABC).
//   This also means 1 node maps to several queries, and 1
     query maps to several nodes.
//   However I believe that the sum of length of all linked
     list is O(N) -- TODO: Source / proof required.
#include<bits/stdc++.h>
#include<string.h>
#include<assert.h>

const int MN = 1000111; // MN > total length of all patterns
const int MC = 26; // Alphabet size.

// Start of Linked list
struct Node {
    int x; Node *next;
} *nil;
struct List {
    Node *first, *last;
    List() { first = last = nil; }
    void add(int x) {
        Node *p = new Node;
        p->x = x; p->next = nil;
        if (first == nil) last = first = p;
        else last->next = p, last = p;
    }
};
// End of linked list
//
struct Aho {
    int qu[MN], suffixLink[MN];
    List leaf[MN];
    int link[MN][MC];
    int sz;
    bool calledBuildLink;

    void init() {
        calledBuildLink = false;
        sz = 0;
        memset(suffixLink, 0, sizeof suffixLink);
        leaf[0] = List();
        memset(link[0], -1, sizeof link[0]);
    }
    int getChild(int type, int v, int c) {
        if (type == 2) assert(calledBuildLink);

        if (link[v][c] >= 0) return link[v][c];
        if (type == 1) return 0;
        if (!v) return link[v][c] = 0;
        return link[v][c] = getChild(type, suffixLink[v], c);
    }
    void buildLink() {
        calledBuildLink = true;
        int first, last;
        qu[first = last = 1] = 0;
        while (first <= last) {
            int u = qu[first++];
            for(int c = 0; c < MC; ++c) {
                int v = link[u][c]; if (v < 0) continue;
                qu[++last] = v;
                if (u == 0) suffixLink[v] = 0;
                else suffixLink[v] = getChild(2, suffixLink[u
                    ], c);

                if (leaf[suffixLink[v]].first != nil) {
                    if (leaf[v].first == nil) {
                        leaf[v].first = leaf[suffixLink[v]].
                            first;
                        leaf[v].last = leaf[suffixLink[v]].last
                            ;
                    }
                    else {
                        leaf[v].last->next = leaf[suffixLink[v
                            ]].first;
                        leaf[v].last = leaf[suffixLink[v]].last
                            ;
                    }
                }
            }
        }
    }
} aho;
// Usage:
int main() {
    aho.init(); // Initialize
    // Foreach query, insert one character at a time:
                int p = 0;
                while (k--) {
                    int x; scanf("%d", &x);
                    int t = aho.getChild(1, p, x);
                    if (t > 0) p = t;
                    else {
                        ++aho.sz;
                        aho.leaf[aho.sz] = List();
                        memset(aho.link[aho.sz], -1, sizeof aho
                            .link[aho.sz]);

                        aho.link[p][x] = aho.sz;
                        p = aho.sz;
                    }
                }
                aho.leaf[p].add(i);
    // Init back link
            aho.buildLink();
    // After this stage, we should use aho.getChild(2, node,
        c) to jump
}
```

5.2 KMP-online

```
// C++ program to implement a
// real time optimized KMP
// algorithm for pattern searching

#include <iostream>
#include <set>
#include <string>
#include <unordered_map>

using std::string;
using std::unordered_map;
using std::set;
using std::cout;

// Function to print
```

```
// an array of length len
void printArr(int* F, int len,
   char name)
{
 cout << '(' << name << ')'
  << "contain: [";

 // Loop to iterate through
 // and print the array
 for (int i = 0; i < len; i++) {
  cout << F[i] << " ";
 }
 cout << "]\n";
}

// Function to print a table.
// len is the length of each array
// in the map.
void printTable(
 unordered_map<char, int*>& FT,
 int len)
{
 cout << "Failure Table: {\n";

 // Iterating through the table
 // and printing it
 for (auto& pair : FT) {

  printArr(pair.second,
    len, pair.first);
 }
 cout << "}\n";
}

// Function to construct
// the failure function
// corresponding to the pattern
void constructFailureFunction(
 string& P, int* F)
{

 // P is the pattern,
 // F is the FailureFunction
 // assume F has length m,
 // where m is the size of P

 int len = P.size();

 // F[0] must have the value 0
 F[0] = 0;

 // The index, we are parsing P[1..j]
 int j = 1;
 int l = 0;

 // Loop to iterate through the
 // pattern
 while (j < len) {

  // Computing the failure function or
  // lps[] similar to KMP Algorithm
  if (P[j] == P[l]) {
   l++;
   F[j] = l;
   j++;
  }
  else if (l > 0) {
   l = F[l - 1];
  }
  else {
   F[j] = 0;
   j++;
  }
 }
}

// Function to construct the failure table.
// P is the pattern, F is the original
// failure function. The table is stored in
// FT[][]
void constructFailureTable(
 string& P,
 set<char>& pattern_alphabet,
 int* F,
 unordered_map<char, int*>& FT)
{
 int len = P.size();

 // T is the char where we mismatched
 for (char t : pattern_alphabet) {

  // Allocate an array
  FT[t] = new int[len];
  int l = 0;
  while (l < len) {
   if (P[F[l]] == t)

    // Old failure function gives
    // a good shifting
    FT[t][l] = F[l] + 1;

   else {

    // Move to the next char if
    // the entry in the failure
    // function is 0
    if (F[l] == 0)
     FT[t][l] = 0;

    // Fill the table if F[l] > 0
    else
     FT[t][l] = FT[t][F[l] - 1];
   }
   l++;
  }
 }
}

// Function to implement the realtime
// optimized KMP algorithm for
// pattern searching. T is the text
// we are searching on and
// P is the pattern we are searching for
void KMP(string& T, string& P,
  set<char>& pattern_alphabet)
{

 // Size of the pattern
 int m = P.size();

 // Size of the text
 int n = T.size();

 // Initialize the Failure Function
 int F[m];

 // Constructing the failure function
 // using KMP algorithm
 constructFailureFunction(P, F);
 printArr(F, m, 'F');

 unordered_map<char, int*> FT;

 // Construct the failure table and
 // store it in FT[][]
 constructFailureTable(
  P,
  pattern_alphabet,
  F, FT);
 printTable(FT, m);
```

```
// The starting index will be when
// the first match occurs
int found_index = -1;

// Variable to iterate over the
// indices in Text T
int i = 0;

// Variable to iterate over the
// indices in Pattern P
int j = 0;

// Loop to iterate over the text
while (i < n) {
 if (P[j] == T[i]) {

  // Matched the last character in P
  if (j == m - 1) {
   found_index = i - m + 1;
   break;
  }
  else {
   i++;
   j++;
  }
 }
 else {
  if (j > 0) {

   // T[i] is not in P's alphabet
   if (FT.find(T[i]) == FT.end())

    // Begin a new
    // matching process
    j = 0;

   else
    j = FT[T[i]][j - 1];

   // Update 'j' to be the length of
   // the longest suffix of P[1..j]
   // which is also a prefix of P

   i++;
  }
  else
   i++;
 }
}
```

```
 // Printing the index at which
 // the pattern is found
 if (found_index != -1)
  cout << "Found at index "
   << found_index << '\n';
 else
  cout << "Not Found \n";

 for (char t : pattern_alphabet)

  // Deallocate the arrays in FT
  delete[] FT[t];

 return;
}

// Driver code
int main()
{
 string T = "cababababcababaca";
 string P = "abababaca";
 set<char> pattern_alphabet
  = { 'a', 'b', 'c' };
 KMP(T, P, pattern_alphabet);
}
/*
The new preprocessing step has a running time complexity of
     O(|\Sigma_P| \cdot M), where \Sigma_P is the alphabet
     set of pattern P, M is the size of P.
The whole modified KMP algorithm has a running time
     complexity of O(|\Sigma_P| \cdot M + N). The auxiliary
     space usage of O(|\Sigma_P| \cdot M).
The running time and space usage look like worse than the
     original KMP algorithm. However, if we are searching
     for the same pattern in multiple texts or the alphabet
     set of the pattern is small, as the preprocessing step
     only needs to be done once and each character in the
     text will be compared at most once (real-time). So, it
     is more efficient than the original KMP algorithm and
     good in practice.
*/
```

5.3 KMP

```
// C++ program for implementation of KMP pattern searching
// algorithm
#include <bits/stdc++.h>

void computeLPSArray(char* pat, int M, int* lps);

// Prints occurrences of txt[] in pat[]
void KMPSearch(char* pat, char* txt)
{
 int M = strlen(pat);
 int N = strlen(txt);

 // create lps[] that will hold the longest prefix suffix
 // values for pattern
 int lps[M];

 // Preprocess the pattern (calculate lps[] array)
 computeLPSArray(pat, M, lps);

 int i = 0; // index for txt[]
 int j = 0; // index for pat[]
 while ((N - i) >= (M - j)) {
  if (pat[j] == txt[i]) {
   j++;
   i++;
  }

  if (j == M) {
   printf("Found pattern at index %d ", i - j);
   j = lps[j - 1];
  }

  // mismatch after j matches
  else if (i < N && pat[j] != txt[i]) {
   // Do not match lps[0..lps[j-1]] characters,
   // they will match anyway
   if (j != 0)
    j = lps[j - 1];
   else
    i = i + 1;
  }
 }
}

// Fills lps[] for given pattern pat[0..M-1]
void computeLPSArray(char* pat, int M, int* lps)
{
 // length of the previous longest prefix suffix
 int len = 0;

 lps[0] = 0; // lps[0] is always 0

 // the loop calculates lps[i] for i = 1 to M-1
 int i = 1;
```

```
 while (i < M) {
  if (pat[i] == pat[len]) {
   len++;
   lps[i] = len;
   i++;
  }
  else // (pat[i] != pat[len])
  {
   // This is tricky. Consider the example.
   // AAACAAAA and i = 7. The idea is similar
   // to search step.
   if (len != 0) {
    len = lps[len - 1];

    // Also, note that we do not increment
    // i here
   }
   else // if (len == 0)
   {
    lps[i] = 0;
    i++;
   }
  }
 }
}

// Driver program to test above function
int main()
{
 char txt[] = "ABABDABACDABABCABAB";
 char pat[] = "ABABCABAB";
 KMPSearch(pat, txt);
 return 0;
}
```

6 Test

6.1 binaryTrie

```
// #include <cassert>
#include <bits/stdc++.h>
// #include <array>
// #include <iostream>
// #include <vector>
// #include <pair>
using namespace std;

// Binary Trie
// Based on https://judge.yosupo.jp/submission/72657
// Supports:
// - get min / max / kth element
// - given K, find x: x^K is min / max / kth
//
// Notes:
// - high mem usage. If just need kth_element
//   -> use OrderedSet.h if MAX_VALUE is ~10^6
//   -> use STL/order_statistic.cpp if MAX_VALUE is big /
     custom type
//
// Tested:
// - (insert, remove, min xor) https://judge.yosupo.jp/
     problem/set_xor_min
// - (insert, max xor) https://cses.fi/problemset/task/1655/
template<
    class Val = long long, // values stored in Trie
    class Count = long long, // frequency of values
    int B = (sizeof(Val) * 8 - 2) // max number of bit
> struct BinaryTrie {
    struct Node {
        std::array<int, 2> child;
        Count count;
        Node() : child{-1, -1}, count(0) {}
    };

    BinaryTrie() : nodes{Node()} {} // create root node

    // Number of elements in the trie
    Count size() {
        return nodes[0].count;
    }

    void insert(Val x, Count cnt = 1) {
        update(x, cnt);
    }
    void remove(Val x, Count cnt = 1) {
        update(x, -cnt);
    }

    // return X: X ^ xor_val is minimum
    pair<Val, Node> min_element(Val xor_val = 0) {
        //assert(0 < size());
        return kth_element(0, xor_val);
    }

    // return X: X ^ xor_val is maximum
    pair<Val, Node> max_element(Val xor_val = 0) {
        //assert(0 < size());
        return kth_element(size() - 1, xor_val);
    }

    // return X: X ^ xor_val is K-th (0 <= K < size())
    pair<Val, Node> kth_element(Count k, Val xor_val = 0) {
        //assert(0 <= k && k < size());
        int u = 0;
        Val x = 0;
        for (int i = B - 1; i >= 0; i--) {
            int b = get_bit(xor_val, i);
            int v0 = get_child(u, b);
            if (nodes[v0].count <= k) {
                k -= nodes[v0].count;
                u = get_child(u, 1-b);
                x |= 1LL << i;
            } else {
                u = v0;
            }
        }
        return {x, nodes[u]};
    }

    // return frequency of x
    Count count(Val x) {
        int u = 0;
        for (int i = B - 1; i >= 0; i--) {
            int b = get_bit(x, i);
            if (nodes[u].child[b] == -1) {
                return 0;
            }
            u = get_child(u, b);
        }
        return nodes[u].count;
    }

// private:
    vector<Node> nodes;

    int get_child(int p, int b) {
        //assert(0 <= p && p < (int) nodes.size());
        //assert(0 <= b && b < 2);
        if (nodes[p].child[b] == -1) {
            nodes[p].child[b] = nodes.size();
            nodes.push_back(Node{});
        }
        return nodes[p].child[b];
    }

    void update(Val x, Count cnt) {
        int u = 0;
        for (int i = B - 1; i >= 0; i--) {
```

```
            nodes[u].count += cnt;
            //assert(nodes[u].count >= 0); // prevent over
                 delete
            int b = get_bit(x, i);
            u = get_child(u, b);
        }
        nodes[u].count += cnt;
        //assert(nodes[u].count >= 0); // prevent over delete
    }

    inline int get_bit(Val v, int bit) {
        return (v >> bit) & 1;
    }
};
int main() {
    int n; cin >> n;
    vector<long long> a(n);
    BinaryTrie<long long, long long> bt;
    bt.insert(0);
    long long preXor = 0LL;
    long long res = LLONG_MIN;
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
        preXor ^= a[i];
        bt.insert(preXor);
        res = max(res, bt.max_element(preXor).first);
    }
    cout << res;
    // long long xor_sum = 0LL;
    // for(int i = 0 ; i < n; ++i) {
    //     xor_sum ^= a[i];
    //     auto tmp = bt.max_element(xor_sum);
    //     mx.push_back(tmp.first);
    //
    // }
    // mx.push_back(bt.max_element().first);
    // auto ans = bt.max_element();
    // cout << ans.first;
    // for(auto &it: mx) cout << it << " ";
    // cout << *max_element(begin(mx), end(mx));
}
```

7 Tree

7.1 lowestCommonAncestor

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

const int N = 5e5+5;
vector<int> g[N];
int n, q;
int h[N], up[N][20];

void dfs(int u) {
    for (auto v: g[u]) {
        if (v == up[u][0]) continue; // v = ancestor of u
        h[v] = h[u] + 1;
        up[v][0] = u;
        for(int j = 1; j < 20; ++j) {
            up[v][j] = up[up[v][j-1]][j-1];
        }
        dfs(v);
    }
}

int lca(int u, int v) {
    if (h[u] != h[v]) {
        if (h[u] < h[v]) swap(u, v); // Without lost of
             generality

        // find ancestor u' of u so h[u'] = h[v]
        int k = h[u] - h[v];
        for(int j = 0; (1<<j) <= k; ++j) {
            if (k >> j & 1) u = up[u][j];
        }
    }
    if (u == v) return u;
    int k = __lg(h[u]);
    for(int j = k; j >= 0; --j) {
        if (up[u][j] != up[v][j]) { // if ancestor 2^j th of
             u and v is different
            u = up[u][j], v = up[v][j];
        }
    }
    return up[u][0];
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> n >> q;
    for(int i = 1; i < n; ++i) {
        int x; cin >> x;
        // g[i].push_back(x);
        // g[x].push_back(i);
        g[i].emplace_back(x);
        g[x].emplace_back(i);
    }
    // for(int i = 0; i < n; ++i) {
    //     cout << "Number " << i << ":\n ";
    //     for(auto &it: g[i]) cout << it << " ";
    //     cout << "\n";
    // }
    dfs(0);
    for(int i = 0; i < q; ++i) {
        int u, v; cin >> u >> v;
        cout << lca(u, v) << "\n";
    }
    return 0;
}
```

7.2 suffixArray

```
#include <stdio.h>
#include <string.h>

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

const int N = 200005;
int n, sa[N], ra[N], rb[N], G;
char a[N];

bool cmp(int x, int y) {
    if (ra[x] != ra[y]) return ra[x] < ra[y];
    return ra[x + G] < ra[y + G];
}

int main() {
    scanf("%s", a + 1);
    n = strlen(a + 1);

    for (int i = 1; i <= n; i++) {
        sa[i] = i;
        ra[i] = a[i];
    }

    for (G = 1; G <= n; G *= 2) {
        sort(sa + 1, sa + n + 1, cmp);
        for (int i = 1; i <= n; i++)
            rb[sa[i]] = rb[sa[i - 1]] + cmp(sa[i - 1], sa[i])
                ;
        for (int i = 1; i <= n; i++)
            ra[i] = rb[i];
        if (ra[sa[n]] == n) break;
```

```
    }

    for (int i = 1; i <= n; i++)
        printf("%d\n", sa[i] - 1);
}

// https://sites.google.com/site/kc97ble/1-3-day-so-va-xau/
     suffix-array
```

7.3 trie

```
#include <stdio.h>
#include <vector>
using namespace std;

// trie

class trie {
  public :
    struct node {
        int a[64];
        int value;
        int& operator[] (int i){ return a[i%64]; }
        node() { for (int i=0; i<64; i++) a[i]=0; value=0; }
    };

    vector <node> a;

    int& operator[] (char *s){
        int pos=0, i, c;

        for (i=0; c=s[i]; i++)
        {
            if (a[pos][c]==0) {
                a.push_back(node());
                a[pos][c] = a.size()-1;
            }
            pos=a[pos][c];
        }
        return a[pos].value;
    }

    void clear(){ a.clear(); a.push_back(node()); }
    trie(){ clear(); }
};

trie tr;

// main
int main(){
    int cnt=0;
    char s[2309];
    for(;;){
        gets(s);
        if (tr[s]==0) tr[s]=++cnt;
        printf("%s = %d\n", s, tr[s]);
    }
    return 0;
}

// https://sites.google.com/site/kc97ble/container/trie-cpp
```

8 buffer-reader

```
// Buffered reader {{{
//
#include<iostream>
namespace IO {
    const int BUFSIZE = 1<<14;
    char buf[BUFSIZE + 1], *inp = buf;

    bool reacheof;
    char get_char() {
        if (!*inp && !reacheof) {
            memset(buf, 0, sizeof buf);
            int tmp = fread(buf, 1, BUFSIZE, stdin);
            if (tmp != BUFSIZE) reacheof = true;
            inp = buf;
        }
        return *inp++;
    }
    template<typename T>
    T get() {
        int neg = 0;
        T res = 0;
        char c = get_char();
        while (!std::isdigit(c) && c != '-' && c != '+') c =
             get_char();
        if (c == '+') { neg = 0; }
        else if (c == '-') { neg = 1; }
        else res = c - '0';

        c = get_char();
        while (std::isdigit(c)) {
            res = res * 10 + (c - '0');
            c = get_char();
        }
        return neg ? -res : res;
    }
};
// }}}
```

9 hash

```
#define long long long
const int N = 1000006, BASE = 1000000007;
int m, n;
char a[N], b[N];
long A[N], B[N], M[N];

void hash_build(char a[], int n, long H[]) {
    for (int i = 1; i <= n; i++)
        H[i] = (H[i - 1] * M[1] + a[i]) % BASE;
}

long hash_range(long H[], int L, int R) {
    return (H[R] - H[L - 1] * M[R - L + 1] + 1LL * BASE *
         BASE) % BASE;
}

// https://sites.google.com/site/kc97ble/1-3-day-so-va-xau/
     hash

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::
             steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

10 template-bak

```
/**
```

```
  *   author: delus
**/

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


// Disable this pragma by default because of debugging
// 2 pragma lines give compiler information to use SIMD
    instruction for optimize code.
// #pragma GCC target("avx2")
// #pragma GCC optimize("O3")


#define vi vector<int>
#define vl vector<long long>
#define vb vector<bool>
#define ll long long
#define ii pair<int, int>
#define vii vector<ii>
#define all(x) x.begin(), x.end()
#define FORIT(i, s) for (auto it=(s.begin()); it!=(s.end());
     ++it)
#define F_OR(i, a, b, s) for (int i=(a); (s)>0? i<(int) (b)
    : i > (int) (b); i+=(s))
#define F_OR1(n) F_OR(i, 0, n, 1)
#define F_OR2(i, e) F_OR(i, 0, e, 1)
#define F_OR3(i, b, e) F_OR(i, b, e, 1)
#define F_OR4(i, b, e, s) F_OR(i, b, e, s)
#define GET5(a, b, c, d, e, ...) e
#define F_ORC(...) GET5(__VA_ARGS__, F_OR4, F_OR3, F_OR2,
    F_OR1)
#define FOR(...) F_ORC(__VA_ARGS__)(__VA_ARGS__)
#define FOR1(n) F_OR(i, 1, n+1, 1)
#define EACH(x, a) for(auto& x: a)
#define BUG(x)                  \
   {                            \
      cout << #x << " = " << x; \
   }
#define IO                              \
   {                                    \
      freopen("input.txt", "r", stdin); \
      freopen("output.txt", "w", stdout); \
   }
#define IOS ios::sync_with_stdio(0); cin.tie(0); cout.tie(0)
    ;
template <class T>
void print(T &x)
{
   for (auto &it : x)
   {
      cout << it << " ";
   }
   cout << "\n";
};
template <class T>
void printPair(T &x)
{
   for (auto &it : x)
   {
      cout << "(" << it.first << ", " << it.second <<") ";
   }
   cout << "\n";
};

int dx[] = {1,1,0,-1,-1,-1, 0, 1};
int dy[] = {0,1,1, 1, 0,-1,-1,-1}; // S,SE,E,NE,N,NW,W,SW
    neighbors

int solve() {

   return 0;
}

int main()
{
   IOS;
   solve();
}
```

11 template

```
#include <bits/stdc++.h>
// #include<ext/pb_ds/assoc_container.hpp>
// #include<ext/pb_ds/tree_policy.hpp>
// #include<bits/extc++.h>

using namespace std;
// using namespace __gnu_pbds;
// typedef tree<int, null_type, less<int>, rb_tree_tag,
// tree_order_statistics_node_update> ordered_tree; typedef
    tree<int, null_type,
// less_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update>
// multi_ordered_tree; tree.find_by_order(x); tree.
    order_of_key(x); remove
// element in multi_ordered_tree: tree.erase(--tree.
    lower_bound(x));
#define ar array
#define vt vector
#define all(v) begin(v), end(v)
#define pb push_back
#define ll long long
#define ld long double
#define ii pair<int, int>
#define iii pair<int, ii>
#define vb vt<bool>
#define vc vt<char>
#define vi vt<int>
#define vl vt<ll>
#define vvb vt<vb>
#define vvc vt<vc>
#define vvi vt<vi>
#define vvl vt<vl>
#define vii vt<ii>
#define fi first
#define se second
#define FORIT(i, s) for (auto it = (s.begin()); it != (s.end
    ()); ++it)
#define F_OR(i, a, b, s)                                    \
  for (int i = (a); (s) > 0 ? i < (int)(b) : i > (int)(b); i
      += (s))
#define F_OR1(n) F_OR(i, 0, n, 1)
#define F_OR2(i, e) F_OR(i, 0, e, 1)
#define F_OR3(i, b, e) F_OR(i, b, e, 1)
#define F_OR4(i, b, e, s) F_OR(i, b, e, s)
#define GET5(a, b, c, d, e, ...) e
#define F_ORC(...) GET5(__VA_ARGS__, F_OR4, F_OR3, F_OR2,
    F_OR1)
#define FOR(...) F_ORC(__VA_ARGS__)(__VA_ARGS__)
#define FOR1(n) F_OR(i, 1, n + 1, 1)
#define EACH(x, a) for (auto &x : a)

#define IOS

    \
  ios_base::sync_with_stdio(0);                             \
  cin.tie(0);

      \
  cout.tie(0);

const int d4x[] = {-1, 0, 1, 0}, d4y[] = {0, -1, 0, 1},
        d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
        d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1};
template <class T> void print(T &x) {
```

```
  for (auto &it : x) {
    cerr << it << " ";
  }
  cerr << "\n";
};
template <class T> void printPair(T &x) {
  for (auto &it : x) {
    cerr << "(" << it.first << ", " << it.second << ") ";
  }
  cerr << "\n";
};
int solve() {
    return 0;
}
int main() {
  IOS;
#ifndef ONLINE_JUDGE
  freopen("in", "r", stdin);
  freopen("out", "w", stdout);
#else
  // online submission
#endif

  solve();
  return 0;
}
```

12 template1

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

int solve() {

    return 0;
}

int main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    solve();
}
```