Team notebook

November 8, 2024

Contents

1 Geometry

1.1 ClosestPair

```
#include <bits/stdc++.h>
#include <unordered_set>
using namespace std;
struct point{
double x,y;
int id;
point(){}
point(double a,double b):x(a),y(b){}
};
double dist(const point&o,const point&p){
double a=p.x-o.x,b=p.y-o.y;
return sqrt(a*a+b*b);
}
double cp(vector<point>&p,vector<point>&x,vector<point>&y){
if(p.size()<4){
 double best = 1e100;
for (int i = 0; i < p.size(); ++i)
for (int j = i + 1; j < p.size(); ++j)
best = min(best, dist(p[i], p[j]));
return best;
}
int ls = (p.size() + 1) >> 1;
double l = (p[ls- 1].x + p[ls].x) * 0.5;
vector<point> xl(ls), xr(p.size()- ls);
unordered_set<int> left;
for (int i = 0; i < ls; ++i) {
xl[i] = x[i];
left.insert(x[i].id);
}
for (int i = ls; i < p.size(); ++i) {
xr[i- ls] = x[i];
}
vector<point> yl, yr;
vector<point> pl, pr;
yl.reserve(ls); yr.reserve(p.size()- ls);
pl.reserve(ls); pr.reserve(p.size()- ls);
for (int i = 0; i < p.size(); ++i) {
if (left.count(y[i].id))
yl.push_back(y[i]);
else
yr.push_back(y[i]);
if (left.count(p[i].id))
pl.push_back(p[i]);
else
pr.push_back(p[i]);
}
double dl = cp(pl, xl, yl);
double dr = cp(pr, xr, yr);
double d = min(dl, dr);
vector<point> yp; yp.reserve(p.size());
for (int i = 0; i < p.size(); ++i) {
if (fabs(y[i].x- l) < d)
yp.push_back(y[i]);
}
for (int i = 0; i < yp.size(); ++i) {
for (int j = i + 1; j < yp.size() && j < i + 7; ++j) {
d = min(d, dist(yp[i], yp[j]));
}
}
return d;
}
double closest_pair(vector<point> &p) {
vector<point> x(p.begin(), p.end());
sort(x.begin(), x.end(), [](const point &a, const point
     &b) {
```

```
return a.x < b.x;
});
vector<point> y(p.begin(), p.end());
sort(y.begin(), y.end(), [](const point &a, const point
     &b) {
return a.y < b.y;
});
return cp(p, x, y);
}
```

1.2 GRAHAM

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

using namespace std;
#define pb push_back
#define f first
#define s second
#define int int64_t
#define pi pair<int,int>


pi operator-(const pi &l, const pi &r) { return {l.f -
     r.f, l.s - r.s}; }
int norm(const pi &p) { return (p.f*p.f) + (p.s*p.s); } //
     x^2 + y^2
int cross(const pi &a, const pi &b) { return a.f * b.s -
     a.s * b.f; } // cross product
int cross(const pi &p, const pi &a, const pi &b) {
                 // cross product
       return cross(a - p, b - p);
}

vector<int> hullInd(const vector<pi> &v) {
   if(v.size()==0)return {};
       int ind = int(min_element(v.begin(),v.end()) -
            v.begin());
       vector<int> cand, hull{ind};
       for(int i=0;i<v.size();i++) if (v[i] != v[ind])
            cand.pb(i);

       sort(cand.begin(),cand.end(), [&](int a, int b) {
            // sort by angle, tiebreak by distance
              pi x = v[a] - v[ind], y = v[b] - v[ind];
              int t = cross(x, y);
              return t != 0 ? t > 0 : norm(x) < norm(y);
       });

       for(int c : cand) { // for every point
              while (hull.size() > 1 &&
                   cross(v[end(hull)[-2]],
                   v[hull.back()], v[c]) <= 0) {
                     hull.pop_back(); // pop until
                          counterclockwise and size > 1
              }
              hull.pb(c);
       }
       return hull;
}


signed main(){
   ios_base::sync_with_stdio(false);
   cin.tie(NULL);

   #ifndef ONLINE_JUDGE
   freopen("file.txt", "r", stdin);
   #endif

   int t; cin >> t;
   while(t--){
       vector<pi> v;
       int n; cin >> n;
       for(int i=0;i<n;i++){
           int a; int b; cin >> a >> b;
           v.pb({a,b});
       }

       vector<int> ans = hullInd(v);
       cout << ans.size() << "\n";
       for(int i : ans){
           cout << v[i].f << " " << v[i].s << "\n";
       }
   }


}
```

1.3 HullDiameter

```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
#define f first
#define s second
#define int int64_t
#define pi pair<int,int>


pi operator-(const pi &l, const pi &r) { return {l.f -
     r.f, l.s - r.s}; }
int norm(const pi &p) { return (p.f*p.f) + (p.s*p.s); } //
     x^2 + y^2
int cross(const pi &a, const pi &b) { return a.f * b.s -
     a.s * b.f; } // cross product
int cross(const pi &p, const pi &a, const pi &b) {
                 // cross product
       return cross(a - p, b - p);
}
double dist(pi &a, pi &b){
   pi d = a-b; return sqrt(norm(d));
}
// db diameter2(vP P) {
//     P = hull(P);
//     int n = sz(P), ind = 1; T ans = 0;
//     if (n > 1) FOR(i,n) for (int j = (i+1)%n;;ind =
     (ind+1)%n) {
//             ckmax(ans, abs2(P[i]-P[ind]));
//             if (cross(P[j]-P[i],P[(ind+1)%n]-P[ind]) <=
     0) break;
//     }
//     return ans;
// }
// unchecked
double diameter2(vector<pi> v) {
       int n = v.size(), ind = 1; double ans = 0;
       if (n > 1) for(int i=0;i<n;i++) for (int j =
            (i+1)%n;;ind = (ind+1)%n) {
       if(dist(v[i],v[ind]) > ans){
           ans = dist(v[i],v[ind]);
       }
              if (cross(v[j]-v[i],v[(ind+1)%n]-v[ind]) <=
                   0) break;
       }
       return ans;
}
signed main(){
   ios_base::sync_with_stdio(false);
   cin.tie(NULL);

   #ifndef ONLINE_JUDGE
   freopen("file.txt", "r", stdin);
   #endif


}
```

1.4 LineContainer

```
#include <bits/stdc++.h>
using namespace std;
#define int int64_t
struct Line {
       mutable int k, m, p;
       bool operator<(const Line& o) const { return k <
            o.k; }
       bool operator<(int x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
       // (for doubles, use inf = 1/.0, div(a,b) = a/b)
       static const int inf = LLONG_MAX;
       int div(int a, int b) { // floored division
              return a / b - ((a ^ b) < 0 && a % b); }
       bool isect(iterator x, iterator y) {
              if (y == end()) return x->p = inf, 0;
              if (x->k == y->k) x->p = x->m > y->m ? inf :
                   -inf;
              else x->p = div(y->m - x->m, x->k - y->k);
              return x->p >= y->p;
       }
       void add(int k, int m) {
              auto z = insert({k, m, 0}), y = z++, x = y;
              while (isect(y, z)) z = erase(z);
```

```
            if (x != begin() && isect(--x, y)) isect(x,
                y = erase(y));
            while ((y = x) != begin() && (--x)->p >=
                y->p)
                  isect(x, erase(y));
      }
      int query(int x) {
            assert(!empty());
            auto l = *lower_bound(x);
            return l.k * x + l.m;
      }
};
```

1.5 MONOTONEHULL

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

using namespace std;
#define pb push_back
#define f first
#define s second
#define int int64_t
#define pi pair<int,int>


pi operator-(const pi &l, const pi &r) { return {l.f -
    r.f, l.s - r.s}; }
int norm(const pi &p) { return (p.f*p.f) + (p.s*p.s); } //
    x^2 + y^2

int cross(const pi &a, const pi &b) { return a.f * b.s -
    a.s * b.f; } // cross product
int cross(const pi &p, const pi &a, const pi &b) {
                 // cross product
      return cross(a - p, b - p);
}

vector<pi> hull;
vector<pi> points;
void monotone_chain() {
      // sort with respect to the x and y coordinates
      sort(points.begin(), points.end());
      // distinct the points
      points.erase(unique(points.begin(), points.end()),
          points.end());
      int n = points.size();

      // 1 or 2 points are always in the convex hull
      if (n < 3) {
            hull = points;
            return;
      }

      // lower hull
      for (int i = 0; i < n; i++) {
            // if with the new point points[i], a right
                turn will be formed,
            // then we remove the last point in the hull
                and test further
            while (hull.size() > 1 &&
                  cross(hull[hull.size() - 2],
                      hull.back(), points[i]) <= 0)

                  hull.pop_back();
            // otherwise, add the point to the hull
            hull.push_back(points[i]);
      }

      // upper hull, following the same logic as the
          lower hull
      auto lower_hull_length = hull.size();
      for (int i = n - 2; i >= 0; i--) {
            // we can only remove a point if there are
                still points left in the
            // upper hull
            while (hull.size() > lower_hull_length &&
                  cross(hull[hull.size() - 2],
                      hull.back(), points[i]) <= 0)
                  hull.pop_back();
            hull.push_back(points[i]);
      }
      // delete point[0] that has been added twice
      hull.pop_back();
}


signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    // #ifndef ONLINE_JUDGE
    // freopen("file.txt", "r", stdin);
    // #endif

    int t; cin >> t;
    while(t--){
        int n; cin >> n;
        points.clear();
        hull.clear();
        for(int i=0;i<n;i++){
            int a; int b; cin >> a >> b;
            points.pb({a,b});
        }

        monotone_chain();
        cout << hull.size() << "\n";
        for(pi i : hull){
            cout << i.f << " " << i.s << "\n";
        }
        //cout << "\n---------\n";
    }


}
```

1.6 PointToStandardForm

```
#include <bits/stdc++.h>
using namespace std;
#define f first
#define s second
#define pi pair<int,int>
//tested very little
//returns Ax + By + C, where A is positive and gcd(A,B) = 1
pair<pi, int> get_line(pi a, pi b) {
      pi z = {b.f - a.f, b.s - a.s};
      swap(z.f, z.s);

      z.f *= -1;
      int g = __gcd(z.f, z.s);
      z.f /= g;
      z.s /= g;

      z = max(z, {-z.f, -z.s});
      return {z, z.f * a.f + z.s * a.s};
}
```

1.7 SegmentIntersection

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

using namespace std;
#define pb push_back
#define f first
#define s second
#define int int64_t
#define pi pair<int,int>


pi operator-(const pi &l, const pi &r) { return {l.f -
    r.f, l.s - r.s}; }
int norm(const pi &p) { return (p.f*p.f) + (p.s*p.s); } //
    x^2 + y^2
int cross(const pi &a, const pi &b) { return a.f * b.s -
    a.s * b.f; } // cross product
int cross(const pi &p, const pi &a, const pi &b) { //
    cross product
      return cross(a - p, b - p);
}

int sn(int x){
   if (x == 0) return 0;
   return x/abs(x);
}
bool rect_int(pi p1, pi p2, pi p3, pi p4) {
      int x1, x2, x3, x4, y1, y2, y3, y4;
      x1 = min(p1.f, p2.f), x2 = max(p1.f, p2.f);
      y1 = min(p1.s, p2.s), y2 = max(p1.s, p2.s);
      x3 = min(p3.f, p4.f), x4 = max(p3.f, p4.f);
      y3 = min(p3.s, p4.s), y4 = max(p3.s, p4.s);
      return !(x2 < x3 || x4 < x1 || y2 < y3 || y4 < y1);
}

bool segmentIntersect(pi p1, pi p2, pi p3, pi p4){
```

```
    return (rect_int(p1,p2,p3,p4) && sn(cross(p1,p2,p4)) *
         sn(cross(p1,p2,p3)) <= 0 && sn(cross(p3,p4,p1)) *
         sn(cross(p3,p4,p2)) <= 0);
}

signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif
    int t; cin >> t;
    while(t--){
        int x1; int y1; int x2; int y2; int x3; int y3; int
             x4; int y4;
        cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4;
        if(segmentIntersect({x1,y1},{x2,y2},{x3,y3},{x4,y4})){
            cout << "YES";
        } else {
            cout << "NO";
        }
        cout <<"\n";

    }


}
```

1.8 Shoelace

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

using namespace std;
#define pb push_back
#define f first
#define s second
#define int int64_t
#define pi pair<int,int>

//area is signed
int twiceArea(vector<pi>pts){
    int ans = 0;
    for(int i=0;i<pts.size()-1;i++){
        ans += pts[i].f * pts[i+1].s - pts[i].s *
             pts[i+1].f;
    }
    ans += pts.back().f *pts[0].s - pts.back().s * pts[0].f;
    return ans;
}
signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif
    int n; cin >> n;
    vector<pi> pts;
    for(int i=0;i<n;i++){
        int a; int b; cin >> a >> b; pts.pb({a,b});
    }
    cout << abs(twiceArea(pts)); // maybe use labs here
}
```

1.9 Two points + radius

```
#include <bits/stdc++.h>
#include <unordered_set>
using namespace std;
struct point{
 double x,y;
 int id;
 point(){}
 point(double a,double b):x(a),y(b){}
 double dot(point p){
    return x*p.x + y*p.y;
 }
 point operator+(const point &p){
    return point(x + p.x,y+p.y);
 }
 point operator-(const point &p){
    return point(x - p.x,y-p.y);
 }
 point operator*(const int p){
    return point(x*p,y*p);
 }
};
vector<point> find_center(point a,point b,long double r){
    point d=(a-b)*0.5;
    if(d.dot(d)>r*r){
    return vector<point>();
    }
    point e=b+d;
    long double fac=sqrt(r*r-d.dot(d));
    vector<point>ans;
    point x=point(-d.y,d.x);
    long double l=sqrt(x.dot(x));
    x=x*(fac/l);
    ans.push_back(e+x);
    x= point(d.y,-d.x);
    x=x*(fac/l);
    ans.push_back(e+x);
    return ans;
}
```

2 Graphs

2.1 Bridges

```
#include<bits/stdc++.h>
using namespace std;
struct Graph {
vector<vector<Edge>> g;
vector<int> vi, low, d, pi, is_b;
int bridges_computed;
int ticks, edges;
Graph(int n, int m) {
g.assign(n, vector<Edge>());
is_b.assign(m, 0);
vi.resize(n);
low.resize(n);
d.resize(n);
pi.resize(n);
edges = 0;
bridges_computed = 0;
}
void AddEdge(int u, int v) {
g[u].push_back(Edge(v, edges));
g[v].push_back(Edge(u, edges));
edges++;
}
void Dfs(int u) {
vi[u] = true;
d[u] = low[u] = ticks++;
for (int i = 0; i < (int)g[u].size(); ++i) {
int v = g[u][i].to;
if (v == pi[u]) continue;
if (!vi[v]) {
pi[v] = u;
Dfs(v);
if (d[u] < low[v]) is_b[g[u][i].id] = true;
low[u] = min(low[u], low[v]);
} else {
low[u] = min(low[u], d[v]);
}
}
}
// Multiple edges from a to b are not allowed.
// (they could be detected as a bridge).
// If you need to handle this, just count
// how many edges there are from a to b.
void CompBridges() {
fill(pi.begin(), pi.end(),-1);
fill(vi.begin(), vi.end(), 0);
fill(low.begin(), low.end(), 0);
fill(d.begin(), d.end(), 0);
ticks = 0;
for (int i = 0; i < (int)g.size(); ++i)
if (!vi[i]) Dfs(i);
bridges_computed = true;
}
map<int, vector<Edge>> BridgesTree() {
if (!bridges_computed) CompBridges();
int n = g.size();
Dsu dsu(g.size());
for (int i = 0; i < n; i++)
for (auto e : g[i])
if (!is_b[e.id]) dsu.Join(i, e.to);
map<int, vector<Edge>> tree;
for (int i = 0; i < n; i++)
for (auto e : g[i])
if (is_b[e.id])
tree[dsu.Find(i)].emplace_back(dsu.Find(e.to), e.id);
return tree;
```

```
}
};
```

2.2 DINIC

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


typedef long long ll;

using namespace std;
//using namespace __gnu_pbds;
#define ordered_set tree<int, null_type,less<int>,
     rb_tree_tag,tree_order_statistics_node_update>
#define pb push_back
#define f first
//#define s second
//#define int ll
#define pi pair<int,int>
#define pf pair<float,float>

struct Dinic {   // flow template
       using F = ll; // flow type
       struct Edge {
              int to;
              F flo, cap;
       };
       int N;
       vector<Edge> eds;
       vector<vector<int>> adj;
       void init(int _N) {
              N = _N;
              adj.resize(N), cur.resize(N);
       }
       /// void reset() { trav(e,eds) e.flo = 0; }
       void ae(int u, int v, F cap, F rcap = 0) {
              assert(min(cap, rcap) >= 0);
              adj[u].pb((eds).size());
              eds.pb({v, 0, cap});
              adj[v].pb(eds.size());
              eds.pb({u, 0, rcap});
       }
       vector<int> lev;
       vector<vector<int>::iterator> cur;
       bool bfs(int s, int t) { // level = shortest
            distance from source
              lev = vector<int>(N, -1);
              for(int i=0;i<N;i++) cur[i] = begin(adj[i]);
              queue<int> q({s});
              lev[s] = 0;
              while (q.size()) {
                     int u = q.front();
                     q.pop();
                     for (auto e : adj[u]) {
                            const Edge &E = eds[e];
                            int v = E.to;
                            if (lev[v] < 0 && E.flo <
                                 E.cap) q.push(v), lev[v]
                                 = lev[u] + 1;
                     }
              }
              return lev[t] >= 0;
       }
       F dfs(int v, int t, F flo) {
              if (v == t) return flo;
              for (; cur[v] != end(adj[v]); cur[v]++) {
                     Edge &E = eds[*cur[v]];
                     if (lev[E.to] != lev[v] + 1 || E.flo
                          == E.cap) continue;
                     F df = dfs(E.to, t, min(flo, E.cap -
                          E.flo));
                     if (df) {
                            E.flo += df;
                            eds[*cur[v] ^ 1].flo -= df;
                            return df;
                     } // saturated >=1 one edge
              }
              return 0;
       }
       F maxFlow(int s, int t) {
              F tot = 0;
              while (bfs(s, t))
                     while (F df = dfs(s, t,
                          numeric_limits<F>::max())) tot
                          += df;
              return tot;
       }
};

signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    // #ifndef ONLINE_JUDGE
    // freopen("file.txt", "r", stdin);
    // #endif
    int l; int r; int n; cin>>l>>r>>n;
    Dinic d;
    d.init(l+r+2);

    for(int i=0;i<n;i++){
        int a; int b; cin>>a>>b;
        d.ae(a+1,l+b+1,1);
    }
    for(int i=0;i<l;i++){
        d.ae(0,i+1, 1);
    }
    for(int i=0;i<r;i++){
        d.ae(i+1+l,l+r+1, 1);
    }

    cout<< d.maxFlow(0, l+r+1)<< "\n";
    d.bfs(0,l+r+1);

    for(int i=1;i<=l;i++){
        for(int v:d.adj[i]){
            if(d.eds[v].cap==0) continue;
            if(d.eds[v].cap==d.eds[v].flo) cout<< i-1 <<"
                 "<< d.eds[v].to-l-1<<"\n";
        }
    }
    return 0;

}
```

2.3 DSU by size

```
#include<bits/stdc++.h>
using namespace std;
int parent[1];//fill
int sz[1]; //fill
void make_set(int v) {
    parent[v] = v;
    sz[v] = 1;
}
int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}
void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (sz[a] < sz[b])
            swap(a, b);
        parent[b] = a;
        sz[a] += sz[b];
    }
}
```

2.4 EulerianPath

```
#include<bits/stdc++.h>
using namespace std;
// Taken from
 https://github.com/lbv/pc-code/blob/master/code/graph.cpp
 // Eulerian Trail
 struct Euler {
 ELV adj; IV t;
 Euler(ELV Adj) : adj(Adj) {}
 void build(int u) {
 while(! adj[u].empty()) {
 int v = adj[u].front().v;
 adj[u].erase(adj[u].begin());
 build(v);
 }
 t.push_back(u);
 }
 };
 bool eulerian_trail(IV &trail) {
 Euler e(adj);
 int odd = 0, s = 0;
 /*
 for (int v = 0; v < n; v++) {
UTP
```

```
30
int diff = abs(in[v]- out[v]);
if (diff > 1) return false;
if (diff == 1) {
if (++odd > 2) return false;
if (out[v] > in[v]) start = v;
}
}
*/
e.build(s);
reverse(e.t.begin(), e.t.end());
trail = e.t;
return true;
}
```

2.5 HUNGRY

```
/*
ID: sahajrastogi
LANG: C++11
*/

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

typedef long long ll;

using namespace std;
//using namespace __gnu_pbds;
#define ordered_set tree<int, null_type,less<int>,
     rb_tree_tag,tree_order_statistics_node_update>
#define pb push_back
#define pi pair<int,int>
#define f first
#define s second
#define int int64_t


int ckmin(int &a, int b) { return a > b ? ((a = b), true)
     : false; }

/**
 * @return the jobs of each worker in the optimal
      assignment,
 * or -1 if the worker is not assigned
 */
template <class T> vector<int> hungarian(const
     vector<vector<T>> &C) {
       int J = C.size();
       int W = C[0].size();
       assert(J <= W);

       // job[w] = job assigned to w-th worker, or -1 if
            no job assigned
       // note: a W-th worker was added for convenience
       vector<int> job(W + 1, -1);
       vector<T> h(W); // Johnson potentials

       const T inf = numeric_limits<T>::max();
       // assign j_cur-th job using Dijkstra with
            potentials
       for (int j_cur = 0; j_cur < J; j_cur++) {
              int w_cur = W; // unvisited worker with
                   minimum distance
              job[w_cur] = j_cur;

              vector<T> dist(W + 1, inf); //
                   Johnson-reduced distances
              dist[W] = 0;
              vector<bool> vis(W + 1);  // whether visited
                   yet
              vector<int> prv(W + 1, -1); // previous
                   worker on shortest path
              while (job[w_cur] != -1) { // Dijkstra step:
                   pop min worker from heap
                     T min_dist = inf;
                     vis[w_cur] = true;
                     int w_next = -1; // next unvisited
                          worker with minimum distance

                     // consider extending shortest path
                          by w_cur -> job[w_cur] -> w
                     for (int w = 0; w < W; w++) {
                            if (!vis[w]) {
                                   // sum of reduced edge
                                        weights w_cur ->
                                        job[w_cur] -> w
                                   T edge =
                                        C[job[w_cur]][w]
                                        - h[w];
                                   if (w_cur != W) {
                                     edge -=
                                          C[job[w_cur]][w_cur]
                                          - h[w_cur];
                                     assert(edge >=
                                          0);
                                   }
                                   if (ckmin(dist[w],
                                        dist[w_cur] +
                                        edge)) { prv[w] =
                                        w_cur; }
                                   if (ckmin(min_dist,
                                        dist[w])) {
                                        w_next = w; }
                            }
                     }
                     w_cur = w_next;
              }

              for (int w = 0; w < W; w++) { // update
                   potentials
                     ckmin(dist[w], dist[w_cur]);
                     h[w] += dist[w];
              }

              while (w_cur != W) { // update job assignment
                     job[w_cur] = job[prv[w_cur]];
                     w_cur = prv[w_cur];
              }
       }

       return job;
}

signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
      #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif
    int n;
    cin>> n;
    vector<vector<int>> table(n, vector<int>(n));
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cin>>table[j][i];
        }
    }
    vector<int> sol = hungarian(table);
    int cost=0;
    for(int i=0;i<n;i++) cost+=table[sol[i]][i];
    cout<< cost<<"\n";
    for(int i=0;i<n;i++){
        cout << sol[i]+1<<" "<< i+1;
        cout<< "\n";
    }
}
```

2.6 KOSARAJU

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


typedef long long ll;

using namespace std;
//using namespace __gnu_pbds;
#define ordered_set tree<int, null_type,less<int>,
     rb_tree_tag,tree_order_statistics_node_update>
#define pb push_back
#define f first
//#define s second
//#define int ll
#define pi pair<int,int>
#define pf pair<float,float>

vector<int> adj[500005];
vector<int> adjr[500005];
int visited[500005]={0};
vector<int> order;
vector<int> scc[500005];
int k = 0;

void dfs(int x){
    visited[x] = 1;
    for(auto nex : adj[x]){
        if(!visited[nex])dfs(nex);
```

```
    }
    order.push_back(x);
}

void dfsr(int x){
    visited[x] = k;
    scc[k].pb(x);
    for(auto nex : adjr[x]){
        if(!visited[nex]) dfsr(nex);
    }
}
signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    // #ifndef ONLINE_JUDGE
    // freopen("file.txt", "r", stdin);
    // #endif

    int n; int m; cin>> n >> m;
    for(int i=0;i<m;i++){
        int a; int b; cin >> a >> b;
        adj[a].pb(b);
        adjr[b].pb(a);
    }

    k=0;

    for(int i=0;i<n;i++){
        if(!visited[i]) dfs(i);
    }

    reverse(order.begin(),order.end());
    for(int i=0;i<500003;i++) visited[i]=0;

    for(int x : order){
        if(!visited[x]){
            k++;
            dfsr(x);
        }
    }

    cout << k << "\n";
    for(int i=1;i<=k;i++){
        cout << scc[i].size();
        for(auto x : scc[i]){
            cout << " "<<x;
        }
        if(i!=k) cout << "\n";
    }

}
```

2.7 two sat with kosaraju

```
/*
ID: sahajrastogi
LANG: C++11
*/
#include <iostream>
#include <bits/stdc++.h>
#include <unordered_set>
// #include <ext/pb_ds/assoc_container.hpp>
// #include <ext/pb_ds/tree_policy.hpp>

typedef long long ll;

using namespace std;
//using namespace __gnu_pbds;
#define ordered_set tree<int, null_type,less<int>,
     rb_tree_tag,tree_order_statistics_node_update>
#define pb push_back
#define f first
#define s second
#define int int64_t
#define pi pair<int,int>
#define pf pair<float,float>


pi pts[1005];


vector<int> adj[2005];
vector<int> adjr[2005];
int visited[2005]={0};
vector<int> order;
vector<int> scc[2005];
int k = 0;

void dfs(int x){
    visited[x] = 1;
    for(auto nex : adj[x]){
        if(!visited[nex])dfs(nex);
    }
    order.push_back(x);
}

void dfsr(int x){
    visited[x] = k;
    scc[k].pb(x);
    for(auto nex : adjr[x]){
        if(!visited[nex]) dfsr(nex);
    }
}
signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif

    int n; int r; int l; cin >> n >> r >> l;
    for(int i=0;i<l;i++){
        int a; int b; cin >> a >> b;
        pts[i] ={a,b};
    }

    for(int i=0;i<l;i++){
        for(int j=i+1;j<l;j++){
            if(pts[i].f == pts[j].f && abs(pts[j].s -
                 pts[i].s) <= 2*r){
                adj[i+l].pb(j);
                //adj[j+l].pb(i);
                //adjr[i].pb(j+l);
                adjr[j].pb(i+l);
            }

            if(pts[i].s == pts[j].s && abs(pts[j].f -
                 pts[i].f) <= 2*r){
                adj[i].pb(j+l);
                //adj[j].pb(i+l);
                //adjr[i+l].pb(j);
                adjr[j+l].pb(i);
            }
        }
    }



    for(int i=0;i<2*l;i++){
        if(!visited[i]) dfs(i);
    }

    reverse(order.begin(),order.end());
    for(int i=0;i<2*l;i++) visited[i]=0;

    for(int x : order){
        if(!visited[x]){
            k++;
            dfsr(x);
        }
    }

    bool good = true;
    for(int i=0;i<l;i++){
        if(visited[i] == visited[i+l]){
            good = false;
        }
    }
    if(good){
        cout << 1;
    } else {
        cout << 0;
    }

    // cout << k << "\n";
    // for(int i=1;i<=k;i++){
    //     cout << scc[i].size();
    //     for(auto x : scc[i]){
    //         cout << " "<<x;
    //     }
    //     if(i!=k) cout << "\n";
    // }



}
```

3 Matrix

3.1 MATRIX

```
#include <bits/stdc++.h>
using namespace std;
#define int int64_t
#define f first
#define s second
const int MN = 505;
const int mod = 998244353;


int power(int b, int e, int m){
    if(e >= 1){
        int p = power(b, e / 2, m) % m;
        if(e%2==0){
            return (p*p)%m;
        } else {
            return (b*((p*p)%m)%m);
        }
    } else if(e == 1) {
        return (b%m);
    } else {
        return 1;
    }
}
int inv(int b, int m){
    return power(b,m-2,m);
}
struct matrix {
    int r, c;
    double m[MN][MN];
    matrix (int _r, int _c) : r (_r), c (_c) {
        memset(m, 0, sizeof m);
    }
    void print() {
        for (int i = 0; i < r; ++i) {
            for (int j = 0; j < c; ++j)
                cout << m[i][j] << " ";
            cout << endl;
        }
    }

    matrix operator *(const matrix &b){
        matrix res(r, b.c);
        if(c!=b.r) cout<< "bad matrix multiplication";
        for(int i=0;i<r;i++){
            for(int j=0;j<b.c;j++){
                for(int k=0;k<c;k++){
                    res.m[i][j]+=m[i][k]*b.m[k][j];
                    //res.m[i][j]%=mod;
                }
            }
        }
        return res;
    }

    void operator *=(const matrix &b){
        *this = *this * b;
        //return *this;
    }

    matrix operator ^(int e){
        matrix res(r,r);
        //matrix id(r,r);
        matrix b = *this;
        for (int i = 0; i < r; ++i)
            res.m[i][i] = 1;
        if (e == 0) return res;
        while (true) {
            if (e & 1) res *= b;
            if ((e >>= 1) == 0) break;
            b *= b;
        }
        return res;
    }

    void operator ^=(int e){
        *this = *this ^ e;
        //return *this;
    }

};
int getRow(vector<vector<int>>& m, int R, int i, int nex) {
       for(int j =nex; j<R;j++) if (m[j][i] != 0) return j;
       return -1; }
int getRow(vector<vector<double>>& m, int R, int i, int
     nex) {
       pair<double,int> bes{0,-1}; // find row with max
            abs value
       for(int j = nex; j< R;j++) bes =
            max(bes,{abs(m[j][i]),j});
       return bes.f < 1e-9 ? -1 : bes.s; }

//for determinant and rank
pair<int,int> gauss(vector<vector<int>> &m) { // convert
     to reduced row echelon form
       if (!m.size()) return {1,0};
       int R = m.size(), C = m[0].size(), rank = 0, nex =
            0;
       int det = 1; // determinant
       for(int i=0;i<C;i++) {
              int row = getRow(m,R,i,nex);
              if (row == -1) { det = 0; continue; }
              if (row != nex) det *= -1,
                   swap(m[row],m[nex]);
              det *= m[nex][i]; rank++;
       det %= mod;
       //det = fmod(det,mod);
       //while(det < 0) det+= mod;
              int x = inv(m[nex][i],mod); for(int k = i;k
                   < C;k++){
          m[nex][k] *= x;
          m[nex][k] %= mod;
       }
              for(int j=0;j < R;j++) if (j != nex) {
                     int v = m[j][i]; if (v == 0) continue;
                     for(int k=i;k<C;k++){
              m[j][k] -= v*m[nex][k];
              m[j][k] %= mod;
              //m[j][k] = fmod(m[j][k],mod);
            }
                }
                nex++;
            }
        // for(int i = 0; i<R;i++){
        //     for(int j=0;j<C;j++){
        //         if(m[i][j] != 0){
        //             //rank++;
        //             break;
        //         }
        //     }
        // }
            return {det,rank};
}

//for system of linear equations with in double form
void slae(vector<vector<double>> &m) { // convert to
     reduced row echelon form
       if (!m.size()) return;
       int R = m.size(), C = m[0].size(), nex = 0;
       for(int i=0;i<C;i++) {
              int row = getRow(m,R,i,nex);
              if (row == -1) { continue; }
              if (row != nex) swap(m[row],m[nex]);
              double x = 1/m[nex][i];
       for(int k = i;k < C;k++){
           m[nex][k] *= x;
       }
              for(int j=0;j < R;j++) if (j != nex) {
                     double v = m[j][i]; if (v == 0)
                          continue;
                     for(int k=i;k<C;k++){
              m[j][k] -= v*m[nex][k];
            }
                }
                nex++;
       }
}

//returns -1 for no soln, 0 for 1 soln, and 1 for infinite
int checkSoln(vector<vector<double>> &m){
    int r = m.size(); int c= m[0].size();
    int cnt = 0; bool imp = false; bool broke = false;
    for(int i = 0;i<r;i++){
        broke = false;
        for(int j = 0;j<c-1;j++){
            if(abs(m[i][j]) > 1e-9) {
                cnt++; broke = true; break;
            }
        }
        if(!broke && abs(m[i][c-1]) > 1e-9) imp = true;
    }
    //cout << m[r-1][c-1] << "! ";
    //cout << cnt << "? ";

    if(imp) return -1;
    if(cnt < c-1) return 1;
    return 0;
}
signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
```

```
    #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif
    while(true){
        int n; cin >> n; if(n==0)break;

        vector<vector<double>>mat(n,vector<double>(n+1,0));
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                cin >> mat[i][j];
            }
        }
        for(int i=0;i<n;i++){
            cin >> mat[i][n];
        }
        slae(mat);

        // for(int i=0;i<n;i++){
        //     for(int j=0;j<n+1;j++){
        //         cout << mat[i][j] << " ";
        //     }
        //     cout << "\n";
        // }
        //cout << checkSoln(mat); cout << "\n";
        if(checkSoln(mat) == -1){
            cout << "inconsistent";
        } else if (checkSoln(mat) == 1){
            cout << "multiple";
        } else {
            for(int i=0;i<n;i++){
                cout << mat[i][n] << " ";
            }
        }
        cout << "\n";
    }

}
```

4 Misc

4.1 BigInt

```
#include <bits/stdc++.h>
using namespace std;
/**
 * Description: Big Integer
 * Source: https://github.com/indy256/codelibrary/
                    blob/master/cpp/numbertheory/bigint.cpp
 * Verification: https://oj.uz/problem/view/IOI11_parrots
 */

// base and base_digits must be consistent
constexpr int base = 1000000000;
constexpr int base_digits = 9;

struct bigint {
       // value == 0 is represented by empty z
       vector<int> z; // digits

       // sign == 1 <==> value >= 0
       // sign == -1 <==> value < 0
       int sign;

       bigint() : sign(1) {}
       bigint(long long v) { *this = v; }

       bigint &operator=(long long v) {
              sign = v < 0 ? -1 : 1; v *= sign;
              z.clear(); for (; v > 0; v = v / base)
                    z.push_back((int) (v % base));
              return *this;
       }

       bigint(const string &s) { read(s); }

       bigint &operator+=(const bigint &other) {
              if (sign == other.sign) {
                     for (int i = 0, carry = 0; i <
                           other.z.size() || carry; ++i) {
                       if (i == z.size())
                              z.push_back(0);
                       z[i] += carry + (i <
                             other.z.size() ?
                             other.z[i] : 0);
                       carry = z[i] >= base;
                       if (carry)
                              z[i] -= base;
                     }
              } else if (other != 0 /* prevent infinite
                    loop */) {
                       *this -= -other;
              }
              return *this;
       }

       friend bigint operator+(bigint a, const bigint &b)
            { return a += b; }

       bigint &operator-=(const bigint &other) {
              if (sign == other.sign) {
                     if (sign == 1 && *this >= other ||
                           sign == -1 && *this <= other) {
                       for (int i = 0, carry = 0; i
                             < other.z.size() ||
                             carry; ++i) {
                         z[i] -= carry + (i <
                               other.z.size() ?
                               other.z[i] : 0);
                         carry = z[i] < 0;
                         if (carry)
                                z[i] += base;
                       }
                       trim();
                     } else {
                       *this = other - *this;
                       this->sign = -this->sign;
                     }
              } else {
                     *this += -other;
              }
              return *this;
       }

       friend bigint operator-(bigint a, const bigint &b)
            { return a -= b; }

       bigint &operator*=(int v) {
              if (v < 0) sign = -sign, v = -v;
              for (int i = 0, carry = 0; i < z.size() ||
                    carry; ++i) {
                if (i == z.size())
                       z.push_back(0);
                long long cur = (long long) z[i] * v
                      + carry;
                carry = (int) (cur / base);
                z[i] = (int) (cur % base);
              }
              trim();
              return *this;
       }

       bigint operator*(int v) const { return
            bigint(*this) *= v; }

       friend pair<bigint, bigint> divmod(const bigint
            &a1, const bigint &b1) {
              int norm = base / (b1.z.back() + 1);
              bigint a = a1.abs() * norm;
              bigint b = b1.abs() * norm;
              bigint q, r;
              q.z.resize(a.z.size());

              for (int i = (int) a.z.size() - 1; i >= 0;
                    i--) {
                r *= base;
                r += a.z[i];
                int s1 = b.z.size() < r.z.size() ?
                      r.z[b.z.size()] : 0;
                int s2 = b.z.size() - 1 < r.z.size()
                      ? r.z[b.z.size() - 1] : 0;
                int d = (int) (((long long) s1 * base
                      + s2) / b.z.back());
                r -= b * d;
                while (r < 0)
                       r += b, --d;
                q.z[i] = d;
              }

              q.sign = a1.sign * b1.sign;
              r.sign = a1.sign;
              q.trim();
              r.trim();
              return {q, r / norm};
       }

       friend bigint sqrt(const bigint &a1) {
              bigint a = a1;
              while (a.z.empty() || a.z.size() % 2 == 1)
                     a.z.push_back(0);

              int n = a.z.size();
```

```
int firstDigit = (int) ::sqrt((double) a.z[n
     - 1] * base + a.z[n - 2]);
int norm = base / (firstDigit + 1);
a *= norm;
a *= norm;
while (a.z.empty() || a.z.size() % 2 == 1)
       a.z.push_back(0);

bigint r = (long long) a.z[n - 1] * base +
     a.z[n - 2];
firstDigit = (int) ::sqrt((double) a.z[n -
     1] * base + a.z[n - 2]);
int q = firstDigit;
bigint res;

for (int j = n / 2 - 1; j >= 0; j--) {
       for (;; --q) {
              bigint r1 = (r - (res * 2 *
                   base + q) * q) * base *
                   base +
                                 (j > 0 ?
                                      (long
                                      long)
                                      a.z[2
                                      *
                                      j
                                      -
                                      1]
                                      *
                                      base
                                      +
                                      a.z[2
                                      *
                                      j
                                      -
                                      2]
                                      :
                                      0);
              if (r1 >= 0) {
                     r = r1;
                     break;
              }
       }
       res *= base;
       res += q;

       if (j > 0) {
              int d1 = res.z.size() + 2 <
                   r.z.size() ?
                   r.z[res.z.size() + 2] :
                   0;
              int d2 = res.z.size() + 1 <
                   r.z.size() ?
                   r.z[res.z.size() + 1] :
                   0;
              int d3 = res.z.size() <
                   r.z.size() ?
                   r.z[res.z.size()] : 0;
              q = (int) (((long long) d1 *
                   base * base + (long
                   long) d2 * base + d3) /
                                 (firstDigit * 2));
              }
       }

       res.trim();
       return res / norm;
}

bigint operator/(const bigint &v) const { return
     divmod(*this, v).first; }

bigint operator%(const bigint &v) const { return
     divmod(*this, v).second; }

bigint &operator/=(int v) {
       if (v < 0) sign = -sign, v = -v;
       for (int i = (int) z.size() - 1, rem = 0; i
            >= 0; --i) {
              long long cur = z[i] + rem * (long
                   long) base;
              z[i] = (int) (cur / v);
              rem = (int) (cur % v);
       }
       trim();
       return *this;
}

bigint operator/(int v) const { return
     bigint(*this) /= v; }

int operator%(int v) const {
       if (v < 0) v = -v;
       int m = 0;
       for (int i = (int) z.size() - 1; i >= 0; --i)
              m = (int) ((z[i] + m * (long long)
                   base) % v);
       return m * sign;
}

bigint &operator*=(const bigint &v) { return *this
     = *this * v; }
bigint &operator/=(const bigint &v) { return *this
     = *this / v; }

bool operator<(const bigint &v) const {
       if (sign != v.sign)
              return sign < v.sign;
       if (z.size() != v.z.size())
              return z.size() * sign < v.z.size() *
                   v.sign;
       for (int i = (int) z.size() - 1; i >= 0; i--)
              if (z[i] != v.z[i])
                     return z[i] * sign < v.z[i] *
                          sign;
       return false;
}

bool operator>(const bigint &v) const { return v <
     *this; }
bool operator<=(const bigint &v) const { return !(v
     < *this); }

bool operator>=(const bigint &v) const { return
     !(*this < v); }

bool operator==(const bigint &v) const { return
     !(*this < v) && !(v < *this); }

bool operator!=(const bigint &v) const { return
     *this < v || v < *this; }

void trim() {
       while (!z.empty() && z.back() == 0)
            z.pop_back();
       if (z.empty()) sign = 1;
}

bool isZero() const { return z.empty(); }

friend bigint operator-(bigint v) {
       if (!v.z.empty()) v.sign = -v.sign;
       return v;
}

bigint abs() const {
       return sign == 1 ? *this : -*this;
}

long long longValue() const {
       long long res = 0;
       for (int i = (int) z.size() - 1; i >= 0; i--)
              res = res * base + z[i];
       return res * sign;
}

friend bigint gcd(const bigint &a, const bigint &b)
     {
       return b.isZero() ? a : gcd(b, a % b);
}

friend bigint lcm(const bigint &a, const bigint &b)
     {
       return a / gcd(a, b) * b;
}

void read(const string &s) {
       sign = 1;
       z.clear();
       int pos = 0;
       while (pos < s.size() && (s[pos] == '-' ||
            s[pos] == '+')) {
              if (s[pos] == '-')
                     sign = -sign;
              ++pos;
       }
       for (int i = (int) s.size() - 1; i >= pos; i
            -= base_digits) {
              int x = 0;
              for (int j = max(pos, i - base_digits
                   + 1); j <= i; j++)
                     x = x * 10 + s[j] - '0';
              z.push_back(x);
       }
       trim();
```

```
}

friend istream &operator>>(istream &stream, bigint
    &v) {
  string s; stream >> s;
  v.read(s);
  return stream;
}

friend ostream &operator<<(ostream &stream, const
    bigint &v) {
  if (v.sign == -1)
        stream << '-';
  stream << (v.z.empty() ? 0 : v.z.back());
  for (int i = (int) v.z.size() - 2; i >= 0;
       --i)
    stream << setw(base_digits) <<
         setfill('0') << v.z[i];
  return stream;
}

static vector<int> convert_base(const vector<int>
    &a, int old_digits, int new_digits) {
  vector<long long> p(max(old_digits,
       new_digits) + 1);
  p[0] = 1;
  for (int i = 1; i < p.size(); i++)
        p[i] = p[i - 1] * 10;
  vector<int> res;
  long long cur = 0;
  int cur_digits = 0;
  for (int v : a) {
        cur += v * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
              res.push_back(int(cur %
                   p[new_digits]));
              cur /= p[new_digits];
              cur_digits -= new_digits;
        }
  }
  res.push_back((int) cur);
  while (!res.empty() && res.back() == 0)
        res.pop_back();
  return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const
    vll &b) {
  int n = a.size();
  vll res(n + n);
  if (n <= 32) {
        for (int i = 0; i < n; i++)
              for (int j = 0; j < n; j++)
                    res[i + j] += a[i] *
                         b[j];
        return res;
  }

  int k = n >> 1;

        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

        for (int i = 0; i < k; i++)
              a2[i] += a1[i];
        for (int i = 0; i < k; i++)
              b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < a1b1.size(); i++)
              r[i] -= a1b1[i];
        for (int i = 0; i < a2b2.size(); i++)
              r[i] -= a2b2[i];

        for (int i = 0; i < r.size(); i++)
              res[i + k] += r[i];
        for (int i = 0; i < a1b1.size(); i++)
              res[i] += a1b1[i];
        for (int i = 0; i < a2b2.size(); i++)
              res[i + n] += a2b2[i];
        return res;
  }

  bigint operator*(const bigint &v) const {
        vector<int> a6 = convert_base(this->z,
             base_digits, 6);
        vector<int> b6 = convert_base(v.z,
             base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size())
              a.push_back(0);
        while (b.size() < a.size())
              b.push_back(0);
        while (a.size() & (a.size() - 1))
              a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < c.size();
             i++) {
              long long cur = c[i] + carry;
              res.z.push_back((int) (cur %
                   1000000));
              carry = (int) (cur / 1000000);
        }
        res.z = convert_base(res.z, 6, base_digits);
        res.trim();
        return res;
  }

};


signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif

    int t; cin >> t;
    bigint a; bigint b;
    cin >> a >> b;

}
```

4.2 Dates

```
#include<bits/stdc++.h>
//
 // Time- Leap years
 //
 // A[i] has the accumulated number of days from months
      previous to i
 const int A[13] = { 0, 0, 31, 59, 90, 120, 151, 181, 212,
      243,
 273, 304, 334 };
 // same as A, but for a leap year
 const int B[13] = { 0, 0, 31, 60, 91, 121, 152, 182, 213,
      244,
 274, 305, 335 };
 // returns number of leap years up to, and including, y
 int leap_years(int y) { return y / 4- y / 100 + y / 400; }
 bool is_leap(int y) { return y % 400 == 0 || (y % 4 == 0
      && y %
 100 != 0); }
 // number of days in blocks of years
 const int p400 = 400*365 + leap_years(400);
 const int p100 = 100*365 + leap_years(100);
 const int p4 = 4*365 + 1;
 const int p1 = 365;
 int date_to_days(int d, int m, int y)
 {
 return (y- 1) * 365 + leap_years(y- 1) + (is_leap(y) ?
      B[m]
 : A[m]) + d;
 }
 void days_to_date(int days, int &d, int &m, int &y)
 {
 bool top100; // are we in the top 100 years of a 400
      block?
 bool top4; // are we in the top 4 years of a 100 block?
 bool top1; // are we in the top year of a 4 block?
 y = 1;
 top100 = top4 = top1 = false;
 y += ((days-1) / p400) * 400;
 d = (days-1) % p400 + 1;
 if (d > p100*3) top100 = true, d-= 3*p100, y += 300;
 else y += ((d-1) / p100) * 100, d = (d-1) % p100 + 1;
 if (d > p4*24) top4 = true, d-= 24*p4, y += 24*4;
 else y += ((d-1) / p4) * 4, d = (d-1) % p4 + 1;
 if (d > p1*3) top1 = true, d-= p1*3, y += 3;
 else y += (d-1) / p1, d = (d-1) % p1 + 1;
 const int *ac = top1 && (!top4 || top100) ? B : A;
```

```
for (m = 1; m < 12; ++m) if (d <= ac[m + 1]) break;
d-= ac[m];
}
```

4.3 Divide and Conquer

```
/*
Consider a dynamic programming problem with the following
      formula

$$dp(i,j) = \min_{0\leq k \leq j} ( dp(i-1, k-1) +
      C(k,j)),$$

where $C(i,j)$ is a cost function and you can compute it
      in $O(1)$ time.
Furthermore, $dp(i,j) =0$ for $j<0$.

The straightforward implementation gives a runtime of
      $O(MN^2)$ if $0\leq i < M$ and $0\leq j < N$.
Divide & Conquer DP allows this to be optimized to $O(M N
      \log N)$.

For each $i,j$, let $\text{opt}(i,j)$ be the value of $k$
      that minimizes the right hand side of the equation.
Divide & Conquer DP only applies if

$$\text{opt}(i,j) \leq \text{opt}(i,j+1).$$

Often, proving this with the given cost function is
      challenging,
but if the cost function satisfies the quadrangle
      inequality, the condition holds.

We can then apply the idea behind Divide & Conquer.
Fix a given $i$. First, compute $\text{opt}(i,n/2)$.
Then compute $\text{opt}(i, n/4)$ using the fact that it
      is less than or equal to $\text{opt}(i, n/2)$.
Similarly, we can compute $\text{opt}(i, 3n/4)$ and
      recursively split the ranges in half, keeping track
      on the lower and upper bounds.

Since each possible value of $\text{opt}(i, j)$ appears
      $O(\log n)$ times, this gives a final runtime of
      $O(mn \log n)$.*/
#include <bits/stdc++.h>
using namespace std;
#define ll long long

const int MAX_N = 1000;
const int MAX_K = 7;

// calc[i][j] stores the # of steps to get all cows
      distance j away to door i
// to make implementing a cyclic array easier, we double
      the size
vector<vector<ll>> calc(2 * MAX_N, vector<ll>(MAX_N + 1,
      0));
// dp[i][j] stores the answer for doors 0,1,.., j and i
      doors open
vector<vector<ll>> dp(MAX_K + 1, vector<ll>(MAX_N + 1,
      INT64_MAX));

int rot;

void compdp(int k, int begin, int end, int rl, int rr) {
       // fixed k, begin and end are the ends of the
            array, rl and rr are the
       // bounds on the last door used
       int mid = (begin + end) / 2;

       ll best = INT64_MAX;
       int best_last = -1;
       // best is min amount moved, last is the last door
            used
       for (int last = rl; last <= min(mid, rr); last++) {
              ll cost = dp[k - 1][last - 1] + calc[last +
                   rot][mid - last + 1];
              if (cost < best) {
                     best = cost;
                     best_last = last;
              } else if (cost == best && last < best_last)
                   {
                     best_last = last;
              }
       }
       dp[k][mid] = best;
       if (begin == end) { return; }
       compdp(k, begin, mid, rl, best_last);
       compdp(k, mid + 1, end, best_last, rr);
}

int main() {
       freopen("cbarn.in", "r", stdin);
       int n, k;
       cin >> n >> k;

       vector<int> a(2 * n);
       for (int i = 0; i < n; i++) {
              cin >> a[i];
              a[i + n] = a[i];
       }

       for (int i = 0; i < n; i++) {
              for (int j = 1; j <= n; j++) {
                     calc[i][j] = calc[i + n][j] =
                          calc[i][j - 1] + (ll)a[i + j -
                          1] * (j - 1);
              }
       }

       // rotate stores where we start in the linear
            representation
       ll ans = INT64_MAX;
       for (rot = 0; rot < n; rot++) {
              for (int i = 0; i < n; i++) { dp[0][i] =
                   calc[rot][i + 1]; }
              for (int i = 1; i < k; i++) {
                     for (int j = 0; j < n; j++) {
                          dp[i][j] = INT64_MAX; }
                     compdp(i, i, n - 1, i, n - 1);
              }
              ans = min(ans, dp[k - 1][n - 1]);
       }

       freopen("cbarn.out", "w", stdout);
       cout << ans << endl;
}

void divide(int lo, int hi, int L, int R) {
    if (lo > hi) return;
    int mid = (lo+hi)/2;
    pair<ll,int> tmp = {1e18,-1};
    FOR(i,max(mid+1,L),R+1)
        tmp = min(tmp,{calc(0,mid)+calc(mid+1,i)
               +calc(i+1,n),i}); // find lowest optimal
                    index
    ans = min(ans,tmp.f);
    divide(lo,mid-1,L,tmp.s); // assume optimal index is
         non-decreasing
    divide(mid+1,hi,tmp.s,R);
}
```

4.4 LIS

```
/*
ID: sahajrastogi
LANG: C++11
*/

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

typedef long long ll;

using namespace std;
//using namespace __gnu_pbds;
#define ordered_set tree<int, null_type,less<int>,
     rb_tree_tag,tree_order_statistics_node_update>
#define pb push_back
#define pi pair<int,int>
#define f first
#define s second
#define int int64_t



//bool visited[200005];
signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif
    int t; cin>>t;
    while(t--){
        int n; int m; cin>>n>> m;
```

```
        vector<int> a(n);
        vector<int> b(m);
        for(int i=0;i<n;i++) cin>>a[i];
        for(int i=0;i<m;i++) cin>>b[i];
        vector<int> dp(n+5);
        vector<int> res(n);
        fill(dp.begin(),dp.end(),-1);
        dp[0]= INT_MAX;
        for(int i=n-1;i>=0;i--){
            int idx = (upper_bound(dp.begin(),dp.end(),
                  a[i], greater<int>())-dp.begin())-1;
            if (dp[idx] == a[i]) idx--;
            res[i]=idx+1;
            dp[idx+1]=max(a[i], dp[idx+1]);
        }
        int ma = 0;
        for(int i=0;i<n;i++) ma=max(ma, res[i]);
        vector<int> inds;
        for(int i=0;i<n;i++) if(res[i]==ma) inds.pb(i);
        inds.pb(n);
        a.pb(-1);
        sort(b.begin(), b.end(), greater<int>());
        int j = 0;
        for(int i=0;i<inds.size();i++){
            while(j<m && b[j]>=a[inds[i]]){
                cout<< b[j]<<" ";
                j++;
            }
            if(i!=inds.size()-1){for(int
                  k=inds[i];k<inds[i+1];k++){
                if(a[k] != -1) cout<< a[k]<<" ";
            }}
        }
        cout<< "\n";

    }


}
```

4.5 Simplex

```
// Simplex Method for Linear Programming
// m - number of (less than) inequalities
// n - number of variables
// C - (m+1) by (n+1) array of coefficients:
// row 0 - objective function coefficients
// row 1:m - less-than inequalities
// column 0:n-1 - inequality coefficients
// column n - inequality constants (0 for objective
     function)
// X[n] - result variables
// return value - maximum value of objective function
// (-inf for infeasible, inf for unbounded)

#include <vector>
#include <cmath>

#define MAXM 400
#define MAXN 400

#define EPS 1e-9
#define INF 1.0/0.0

double A[MAXM][MAXN];
int basis[MAXM], out[MAXN];

void pivot(int m, int n, int a, int b) {
    int i, j;
    for(i = 0; i <= m; i ++) if(i != a) for(j = 0; j <= n;
         j ++) if(j != b) {
        A[i][j] -= A[a][j] * A[i][b] / A[a][b];
    }
    for(j = 0; j <= n; j ++) if(j != b) A[a][j] /= A[a][b];
    for(i = 0; i <= m; i ++) if(i != a) A[i][b] =
         -A[i][b]/A[a][b];

    A[a][b] = 1/A[a][b];

    i = basis[a];
    basis[a] = out[b];
    out[b] = i;
}

double simplex(int m, int n, double C[][MAXN], double X[])
     {
    int i, j, ii, jj;

    for(i = 1; i <= m; i ++) for(j = 0; j <= n; j ++)
         A[i][j] = C[i][j];
    for(j = 0; j <= n; j ++) A[0][j] = -C[0][j];
    for(i = 0; i <= m; i ++) basis[i] = -i;
    for(j = 0; j <= n; j ++) out[j] = j;

    for(;;) {
        for(i = ii = 1; i <= m; i ++) {
            if(A[i][n] < A[ii][n]
                || (A[i][n] == A[ii][n] && basis[i] <
                     basis[ii])) ii = i;
        }

        if(A[ii][n] >= -EPS) {
            break;
        }

        for(j = jj = 0; j < n; j ++) {
            if(A[ii][j] < A[ii][jj]-EPS
                || (A[ii][j] < A[ii][jj]+EPS &&
                     out[i]<out[j])) jj=j;
        }

        if(A[ii][jj] >= -EPS) return -INF;
        pivot(m,n,ii,jj);
    }

    for(;;) {
        for(j = jj = 0; j < n; j ++)
            if(A[0][j] < A[0][jj]
                || (A[0][j] == A[0][jj] && out[j] <
                     out[jj])) jj = j;
        if(A[0][jj] > -EPS) break;
        for(i=1,ii=0; i <= m; i ++)
            if(A[i][jj] > EPS &&
                (!ii || A[i][n]/A[i][jj] <
                     A[ii][n]/A[ii][jj]-EPS
                    || (A[i][n]/A[i][jj] <
                         A[ii][n]/A[ii][jj]+EPS
                        && basis[i]<basis[ii]))) ii = i;

        if(A[ii][jj] <= EPS) return INF;
        pivot(m,n,ii,jj);
    }

    for(j = 0; j < n; j ++) X[j] = 0;
    for(i = 1; i <= m; i ++) if(basis[i] >= 0) X[basis[i]]
         = A[i][n];

    return A[0][n];
}
```

4.6 SlopeTrick

```
/*
From the latter link (modified):

Slope trick is a way to represent a function that
     satisfies the following conditions:

It can be divided into multiple sections, where each
     section is a linear function (usually) with an
     integer slope.
It is a convex/concave function. In other words, the slope
     of each section is non-decreasing or non-increasing
     when scanning the function from left to right.
It's generally applicable as a DP optimization.
Let $dp[i][j]$ denote the maximum amount of money you can
     have on day $i$ if you
have exactly $j$ shares of stock on that day. The final
     answer will be
$dp[N][0]$. This solution runs in $\mathcal{O}(N^2)$ time.

Copyvector<vl> dp = {{0}};int N;
int main() {  re(N); FOR(i, N) {          int x;
     re(x);        dp.pb(vl(i + 2, -INF));      FOR(j, i +
     1) {

If we run this on the first sample case, then we get the
     following table:

Input:

9
10 5 4 7 9 12 6 2 10

Output:

dp[0] = { 0}
dp[1] = { 0, -10}
dp[2] = { 0,  -5, -15}
```

dp[3] = { 0, -4, -9, -19}
dp[4] = { 3, -2, -9, -16, -26}
dp[5] = { 7, 0, -7, -16, -25, -35}
dp[6] = { 12, 5, -4, -13, -23, -35, -47}
dp[7] = { 12, 6, -1, -10, -19, -29, -41, -53}
dp[8] = { 12, 10, 4, -3, -12, -21, -31, -43, -55}
dp[9] = { 20, 14, 7, -2, -11, -21, -31, -41, -53, -65}

However, the DP values look quite special! Specifically, let

$$dif[i][j]=dp[i][j]-dp[i][j+1]\ge 0.$$

Then $dif[i][j]\le dif[i][j+1]$ for all $j\ge 0$. In other words, $dp[i][j]$ as
a function of j is concave down.

Full Solution

We'll process the shares in order. Suppose that we are currently considering the
i-th day, where shares are worth p_i. We can replace (buy or sell a share)
in the statement with (buy, then sell somewhere between 0 and 2 shares).

If we currently have j shares and overall balance b, then after buying,
j increases by one and b decreases by p_i. So we set
$dp[i][j]=dp[i-1][j-1]-p_i$ for all j. Note that the differences between
every two consecutive elements of $dp[i]$ have not changed.

If we choose to sell a share, this is equivalent to setting
$dp[i][j]=\max(dp[i][j],dp[i][j+1]+p_i)$ for all j at the same time. By the
concavity condition, $dp[i][j]=dp[i][j+1]+p_i$ will hold for all j less than
a certain threshold while $dp[i][j]$ will remain unchanged for all others. So
this is equivalent to inserting p_i into the list of differences while
maintaining the condition that the differences are in sorted order.

So choosing to sell between 0 and 2 shares is represented by adding p_i to
the list of differences two times. After that, we should pop the smallest
difference in the list because we can't end up with a negative amount of
shares.

Example

The implementation is quite simple; maintain a priority queue representing
$dif[i]$ that allows you to pop the minimum element. After adding i elements,
ans stores the current value of $dp[i][i]$. At the end, you add all the
differences in $dif[N]$ to go from $dp[N][N]$ to $dp[N][0]$.

Copy#include <bits/stdc++.h>using namespace std;
int main() { int N; cin >> N; priority_queue<int, vector<int>, greater<int>> pq; long long ans = 0; for (int i = 0; i < N; ++i) { int p;

Extension

Stock Trading (USACO Camp): What if your amount of shares can go negative, but
you can never have more than L shares or less than $-L$?

Potatoes & Fertilizers

2019 - Potatoes & FertilizersLMiO - NormalFocus Problem
try your best to solve this problem before continuing!

Simplifying the Problem

Instead of saying that moving fertilizer from segment i to segment j costs
$|i-j|$, we'll say that it costs 1 to move fertilizer from a segment to an
adjacent segment.

Let the values of $a_1,a_2,\ldots,a_N$ after all the transfers be
$a_1',a_2',\ldots,a_N'$. If we know this final sequence, how much did the
transfers cost (in the best case scenario)? It turns out that this is just

$$C=\sum_{i=1}^{N-1}\left|\sum_{j=1}^i(a_j-a_j')\right|.$$

We can show that this is a lower bound and that it's attainable. The term
$D=\sum_{j=1}^i(a_j-a_j')$ denotes the number of units of fertilizer that move
from segment i to segment $i+1$. Namely, if D is positive then D units of
fertilizer moved from segment i to segment $i+1$; otherwise, $-D$ units of
fertilizer moved in the opposite direction. Note that it is never optimal to
have fertilizer moving in both directions.

Let $dif_i=a_i-b_i$ and define $d_j=\sum_{i=1}^jdif_i$ for each $0\le j\le N$.
Similarly, define $dif_i'=a_i'-b_i$ and $d_j'=\sum_{i=1}^jdif_i'$. Since we want
$dif_i'\ge 0$ for all i, we should have
$d_0=d_0'\le d_1'\le \cdots\le d_N'=d_N.$ Conversely, every sequence
$(d_0',d_1',\ldots,d_N')$ that satisfies this property corresponds to a valid
way to assign values of $(a_1',a_2',\ldots,a_N')$.

Now you can verify that $C=\sum_{i=1}^{N-1}|d_i-d_i'|$. This makes sense since
moving one unit of fertilizer one position is equivalent to changing one of the
d_i by one (although d_0,d_N always remain the same).

Slow Solution

For each $0\le i\le N$ and $0\le j\le d_N$, let $dp[i][j]$ be the minimum cost
to determine $d_0',d_1',\ldots,d_i'$ such that $d_i'\le j$. Note that by
definition, $dp[i][j]\ge dp[i][j+1]$. We can easily calculate these values in
$\mathcal{O}(N\cdot d_N)$ time.

Full Solution

Similar to before, this DP is concave up for a fixed i! Given a piecewise
linear function $f_i(x)$ that takes as input x and outputs $dp[i][x]$, we need
to support the following two operations to transform this function into
f_{i+1}.

Add $|x-k|$ to the function for some k
Set $f(x)=\min(f(x),f(x-1))$ for all x

Again, these can be done with a priority queue. Instead of storing the
consecutive differences, we store the points where the slope of the piecewise
linear function changes by one.

The first operation corresponds to inserting k into the priority queue two
times because the slope increases by two at $x=k$.
The latter operation just corresponds to removing the greatest element of the
priority queue.

This solution runs in $\mathcal{O}(N\log N)$ time.

Copy#include <bits/stdc++.h>using namespace std;
typedef long long ll;
int N;ll fst = 0; // value of DP function at 0priority_queue<ll> points; // points where DP function changes slope
int main() {

*/

```
vector<vl> dp = {{0}};
int N;

int main() {
	re(N);
```

```
	FOR(i, N) {
		int x;
		re(x);
		dp.pb(vl(i + 2, -INF));
		FOR(j, i + 1) {
			ckmax(dp.bk[j + 1], dp[sz(dp) - 2][j]
				- x);
			ckmax(dp.bk[j], dp[sz(dp) - 2][j]);
			if (j) ckmax(dp.bk[j - 1], dp[sz(dp)
				- 2][j] + x);
		}
	}
	int cnt = 0;
	trav(t, dp) {
		pr("dp[", cnt++, "] = ");
		pr('{');
		FOR(i, sz(t)) {
			if (i) cout << ", ";
			cout << setw(3) << t[i];
		}
		ps('}');
	}
}
```

4.7 sos

```
void sos (vi& dp, int x = 1) { // x = -1 reverses
    int SZ = 31-__builtin_clz(sz(dp));
    FOR(i,SZ) FOR(j,1<<SZ) if (j&(1<<i))
        dp[j^(1<<i)] += x*dp[j];
}

vi andConv(vi a, vi b) { // a[i]*b[j] contributes to
     result[i&j]
    sos(a), sos(b);
    FOR(i,sz(a)) a[i] *= b[i];
    sos(a,-1); return a;
}
```

4.8 Treap

```
#include <stdlib.h>

struct Node {
	// the value and priority of the node respectively
	int val, pri;
	// pointer to left and right child (NULL means no
		child)
	Node *left, *right;
	Node(int val) : val(val), pri(rand()), left(NULL),
		right(NULL){};
} *root;

/**
 * pass in root as pointer, left and right as references
 * to a node pointer so we can modify them
 * (alternatively, we can return left and right pointers
 * as an std::pair)
 */
void split(Node *root, int x, Node *&left, Node *&right) {
	if (!root) {
		left = right = NULL;
		return;
	}
	if (root->val <= x) {
		split(root->right, x, root->right, right);
		left = root;
	} else {
		split(root->left, x, left, root->left);
		right = root;
	}
}

/**
 * merge left and right pointers into root which
 * is a reference to a pointer to enable
 * modification within the function
 */
void merge(Node *&root, Node *left, Node *right) {
	if (!left || !right) {
		root = left ? left : right;
		return;
	}
	if (left->pri > right->pri) {
		merge(left->right, left->right, right);
		root = left;
	} else {
		merge(right->left, left, right->left);
		root = right;
	}
}
```

5 Number Theory

5.1 BinaryExponentiation

```
//switch * to + for safe ll multiplication
int power(int b, int e, int m) {
    int res = 1;
    while (e > 0) {
        if (e & 1){
            res = res * b;res%=m;
        } b = b * b;b%= m;e >>= 1;
    } return res;
}
int inv(int b, int m){ return power(b,m-2,m);}
```

5.2 CRT

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
 /**
 * Chinese remainder theorem.
 * Find z such that z % x[i] = a[i] for all i.
 * */
long long crt(vector<long long> &a, vector<long long> &x) {
    long long z = 0;
    long long n = 1;
    for (int i = 0; i < x.size(); ++i)
    n *= x[i];
    for (int i = 0; i < a.size(); ++i) {
        long long tmp = (a[i] * (n / x[i])) % n;
        tmp = (tmp * mod_inv(n / x[i], x[i])) % n;
        z = (z + tmp) % n;
    }
    return (z + n) % n;
}
```

5.3 Diophantine

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

long long gcd(long long a, long long b, long long &x, long
     long &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    long long x1, y1;
    long long d = gcd(b % a, a, x1, y1);
    x = y1- (b / a) * x1;
    y = x1;
    return d;
}
bool find_any_solution(long long a, long long b, long long
     c,long long &x0, long long &y0, long long &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
    return false;
    }
     x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 =-x0;
    if (b < 0) y0 =-y0;
    return true;
}
void shift_solution(long long &x, long long &y, long long
     a, long long b, long long cnt) {
    x += cnt * b;
    y-= cnt * a;
}
long long find_all_solutions(long long a, long long b,
     long long c,
long long minx, long long maxx, long long miny, long long
     maxy) {
    long long x, y, g;
```

```
    if (!find_any_solution(a, b, c, x, y, g)) return 0;
    a /= g;
    b /= g;
    long long sign_a = a > 0 ? +1 :-1;
    long long sign_b = b > 0 ? +1 :-1;
    shift_solution(x, y, a, b, (minx- x) / b);
    if (x < minx) shift_solution(x, y, a, b, sign_b);
    if (x > maxx) return 0;
    long long lx1 = x;
    shift_solution(x, y, a, b, (maxx- x) / b);
    if (x > maxx) shift_solution(x, y, a, b,-sign_b);
    long long rx1 = x;
    shift_solution(x, y, a, b,-(miny- y) / a);
    if (y < miny) shift_solution(x, y, a, b,-sign_a);
    if (y > maxy) return 0;
    long long lx2 = x;
    shift_solution(x, y, a, b,-(maxy- y) / a);
    if (y > maxy) shift_solution(x, y, a, b, sign_a);
    long long rx2 = x;
    if (lx2 > rx2) swap(lx2, rx2);
    long long lx = max(lx1, lx2);
    long long rx = min(rx1, rx2);
    if (lx > rx) return 0;
    return (rx- lx) / abs(b) + 1;
}
```

5.4 ExtendedEuclidean

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
//tested very little
void ext_euclid(int a, int b, int &x, int &y, int &g) {
    x = 0, y = 1, g = b;
    int m, n, q, r;
    for (int u = 1, v = 0; a != 0; g = a, a = r) {
    q = g / a, r = g % a;
    m = x- u * q, n = y- v * q;
    x = u, y = v, u = m, v = n;
    }
 }
int mod_inv(int n, int m) {
    int x, y, gcd;
    ext_euclid(n, m, x, y, gcd);
    if (gcd != 1)
    return 0;
    return (x + m) % m;
}
```

5.5 MillerRabin

```
#include <bits/stdc++.h>
using namespace std;
const int rounds = 20;
 // checks whether a is a witness that n is not prime, 1 <
      a < n
bool witness(long long a, long long n) {
// check as in Miller Rabin Primality Test described
   long long u = n- 1;
   int t = 0;
   while (u % 2 == 0) {
       t++;
       u >>= 1;
   }
   long long next = mod_pow(a, u, n);
   if (next == 1) return false;
   long long last;
   for (int i = 0; i < t; ++i) {
       last = next;
       next = mod_mul(last, last, n);
       if (next == 1) {
       return last != n- 1;
       }
   }
   return next != 1;
}
// Checks if a number is prime with prob 1- 1 / (2 ^ it)

// D(miller_rabin(99999999999999997LL) == 1);
// D(miller_rabin(9999999999971LL) == 1);
// D(miller_rabin(7907) == 1);
bool miller_rabin(long long n, int it = rounds) {
   if (n <= 1) return false;
   if (n == 2) return true;
   if (n % 2 == 0) return false;
   for (int i = 0; i < it; ++i) {
       long long a = rand() % (n- 1) + 1;
       if (witness(a, n)) {
           return false;
       }
   }
   return true;
}
```

5.6 PollardRho

```
#include <bits/stdc++.h>
using namespace std;
long long pollard_rho(long long n) {
   long long x, y, i = 1, k = 2, d;
   x = y = rand() % n;
   while (1) {
       ++i;
       x = mod_mul(x, x, n);
       x += 2;
       if (x >= n) x-= n;
       if (x == y) return 1;
       d = __gcd(abs(x- y), n);
       if (d != 1) return d;
       if (i == k) {
           y = x;
           k *= 2;
       }
   }
   return 1;
}
// Returns a list with the prime divisors of n
vector<long long> factorize(long long n) {
   vector<long long> ans;
   if (n == 1)
   return ans;
   if (miller_rabin(n)) {
       ans.push_back(n);
   } else {
       long long d = 1;
       while (d == 1)
       d = pollard_rho(n);
       vector<long long> dd = factorize(d);
       ans = factorize(n / d);
       for (int i = 0; i < dd.size(); ++i)
       ans.push_back(dd[i]);
   }
   return ans;
}
```

5.7 Sieve+Totient

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

typedef long long ll;
using namespace std;
#define pb push_back
#define int ll
#define pi pair<int,int>

vector<int> primes;
int sieve[1000005] = {0};
int phi[1000005];
signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif
    sieve[0] =0; sieve[1] = 1;
    cout << sieve[5];
    for(int i=2;i<100000;i++){
        if(sieve[i]) continue;
        primes.pb(i);
        for(int j=i*i;j<100000;j+=i){
            sieve[j] = i;
        }
    }
    for(int i=1;i<1000000;i++) phi[i] = i;
    for(int i=1;i<1000000;i++){
        if(sieve[i]) continue;
        for(int j=i;j<1000000;j+=i){
            phi[j] -= phi[j]/i;
        }
    }
}
```

6 Range Query

6.1 BIT

```
#include <bits/stdc++.h>
using namespace std;
int sum(int i, vector<int> &bit){
    int res = 0; while(i>=0) res+=bit[i]; i=((i+1)&i)-1;
        return res;
}
void upd(int i, int wt, vector<int> &bit){
    while(i<bit.size()) bit[i]+=wt; i=(i+1)|i;
}
int range(int a, int b,vector<int>&bit){
    if(a == 0) return sum(b,bit); // care for indexing
    return sum(b,bit) - sum(a-1,bit);
}
```

6.2 SEGTREEBigStepper

```
#include <bits/stdc++.h>
using namespace std;
template <class T> struct SegTree { // cmb(ID,b) = b
       const T ID{0};
       T cmb(T a, T b) { }
       int n; vector<T> seg;
       void init(int _n) { // upd, query also work if n =
            _n
              for (n = 1; n < _n; ) n *= 2;
              seg.assign(2*n,ID);
   }
       void pull(int p) {
       seg[p] = cmb(seg[2*p],seg[2*p+1]);
   }
       void upd(int p, T val) { // set val at position p
              seg[p += n] += val;
       for (p /= 2; p; p /= 2) pull(p);
       }
       T query(int l, int r) { // zero-indexed, inclusive
              T ra = ID, rb = ID;
              for (l += n, r += n+1; l < r; l /= 2, r /=
                   2) {
                    if (l&1) ra = cmb(ra,seg[l++]);
                    if (r&1) rb = cmb(seg[--r],rb);
              }
              return cmb(ra,rb);
       }

       int bSearch(int target){
              int p = 1;
              if(seg[p] < target) return 0;
              while(p < n){
                    if(seg[2*p] < target){
                           p = 2*p+1;
                    } else {
                           p = 2*p;
                    }
              }
              return p-n+1;
       }
       // int first_at_least(int lo, int val, int ind, int
            l, int r) { // if seg stores max across range
       //     if (r < lo || val > seg[ind]) return -1;
       //     if (l == r) return l;
       //     int m = (l+r)/2;
       //     int res = first_at_least(lo,val,2*ind,l,m);
            if (res != -1) return res;
       //     return first_at_least(lo,val,2*ind+1,m+1,r);
       // }
};
```

6.3 SEGTREELazy

```
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
#define pb push_back
#define f first
#define s second
#define int ll
#define pi pair<int,int>

struct Node{
       bool isID = false;
       int sum =0;
       Node(bool x) : isID(x){}
       Node(bool x, int s) : isID(x), sum(s){}
};

struct lNode{
       bool isID = false;
       int set = -1;
       int inc = 0;
       lNode(bool x) : isID(x){}
       lNode(bool x, int a, int b): isID(x), set(a),
            inc(b){}
};

Node idnode(true,0);
lNode lazynode(true);
template <class T, class Q> struct SegTree { // cmb(ID,b)
     = b
       const T ID{idnode}; const Q IDQ{lazynode};
       T cmb(T a, T b) {
       if(a.isID) return b;
              if(b.isID) return a;
              Node res(false,0);
              res.sum = (a.sum+b.sum);
              return res;
   }

       Q lazycmb(Q a, Q b){
              if(a.isID) return b;
              if(b.isID) return a;
              lNode res(false);
              if(a.set != -1) return a;
              res.set = b.set;
              res.inc = b.inc + a.inc;
              return res;
   }

   T cmbTQ(T a, Q b,int l,int r){
              if(b.isID) return a;
              Node res(false);
              if(a.isID) {
                     res.sum = 0;
              } else {
                     res.sum = a.sum;
              }
              if(b.set != -1) res.sum = b.set*(r-l+1);
              res.sum += b.inc*(r-l+1);
              return res;
   }
   int n; vector<T> seg; vector<Q> lazy;
   void init(int _n) { // upd, query also work if n =
        _n
              for (n = 1; n < _n; ) n *= 2;
              seg.assign(2*n,ID);
              lazy.assign(2*n,IDQ);
}


   void printTree(){
              for(int i=1;i<2*n;i++){
                     cout << seg[i].sum << " ";
              }
              cout << "\n";
   }
   void push(int node, int l, int r){
              //seg[node].sum =
                   ((seg[node].sum*lazy[node].m)%mod +
                   (lazy[node].c*(r-l+1))%mod)%mod; //
                   operation dependent
              seg[node] = cmbTQ(seg[node],lazy[node], l,
                   r);
              if(l != r){
                     lazy[2*node] =
                          lazycmb(lazy[node],lazy[2*node]);
                     lazy[2*node+1] =
                          lazycmb(lazy[node],lazy[2*node+1]);
              }
              lazy[node] = IDQ;
   }
   void pull(int p) {
   seg[p] = cmb(seg[2*p],seg[2*p+1]);
}

   void upd(int l, int r, Q val){
              upd(l,r,val,0,n-1,1);
   }

   void upd(int l, int r, Q val, int start, int end,
        int node) {
              push(node,start,end);
              if(r < start || l > end) return; // maybe
                   not needed
```

```
                if(l <= start && end <= r){
                        lazy[node] = val;
                        push(node,start,end);
                        return;
                }
                int mid = (start + end)/2;
                //if(start <=l && r <= mid){
                        upd(l,r,val,start,mid,2*node);
                //} else {
                        upd(l,r,val,mid+1,end,2*node+1);
                //}
                pull(node);
    }

        T query(int l, int r){
                return query(l,r,0,n-1,1);
        }
        T query(int l, int r, int start, int end, int node)
              { // zero-indexed, inclusive
                push(node,start,end);
                if(r < start || l > end){
                        return ID;
                }
                if(l <= start && end <= r){
                        return seg[node];
                } else {
                        int mid = (start + end)/2;
                        T x = query(l,r, start, mid,2*node);
                        T y = query(l,r, mid+1, end,2*node+1);
                        return cmb(x,y);
                }
        }
};

signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif
        int n; int q; cin >> n >> q;
        SegTree<Node,lNode> seg;
        seg.init(n+5);
        for(int i=0;i<n;i++){
                int x; cin >> x;
                     seg.upd(i,i,lNode(false,x,0));
        }
        // for(int i=0;i<n;i++){
        //      cout << seg.seg[0].sum;
        // }
        for(int i=0;i<q;i++){
                int k; cin >> k;
                if(k==1){
                        int a; int b; int x; cin >> a >> b
                              >>x;a--;b--;
                        seg.upd(a,b,lNode(false, -1 , x));
                } else if(k==2){
                        int a; int b; int x; cin >> a >> b
                              >>x;a--;b--;
                        seg.upd(a,b,lNode(false, x , 0));
                } else {
                        int a; int b; cin >> a >> b;a--;b--;
                        cout << seg.query(a,b).sum << "\n";
                }
        }

}
```

6.4 SEGTREERecursive

```
#include <bits/stdc++.h>
template <class T> struct SegTree { // cmb(ID,b) = b
        const T ID{0}; T cmb(T a, T b) {
        if(a == ID){
                        return b;
                }
                if(b == ID){
                        return a;
                }
                return min(a,b);
    }

        int n; vector<T> seg;
        void init(int _n) { // upd, query also work if n =
              _n
                for (n = 1; n < _n; ) n *= 2;
                seg.assign(2*n,ID);
    }
        void pull(int p) {
        seg[p] = cmb(seg[2*p],seg[2*p+1]);
    }
        void upd(int p,T val) upd(p, val,0,n-1,1);
        void upd(int p, T val, int start, int end, int
              node) { // set val at position p

                if(p < start || p > end) return; // maybe
                     not needed

                if(start == end){
                        seg[node] = val;
                        return;
                }

                int mid = (start + end)/2;
                if(start <=p && p <= mid){
                        upd(p,val,start,mid,2*node);
                } else {
                        upd(p,val,mid+1,end,2*node+1);
                }
                pull(node);
    }

        T query(int l, int r) query(l,r,1,0,n-1)
        T query(int l, int r, int node, int start, int end)
              { // zero-indexed, inclusive

                if(r < start || l > end){
                        return ID;
                }
                if(l <= start && end <= r){
                        return seg[node];
                } else {
                        int mid = (start + end)/2;
                        T x = query(l,r,2*node, start, mid);
                        T y = query(l,r,2*node+1, mid+1, end);
                        return cmb(x,y);
                }
        }
};
```

6.5 Sparse Table

```
#include <bits/stdc++.h>
using namespace std;
const int MN = 100000 + 10; // Max number of elements
const int ML = 18; // ceil(log2(MN));
struct st {
    int data[MN];
    int M[MN][ML];
    int n;
    void init(const vector<int> &d) {
        n = d.size();
        for (int i = 0; i < n; ++i)
        data[i] = d[i];
        build();
    }
    void build() {
        for (int i = 0; i < n; ++i)
        M[i][0] = data[i];
        for (int j = 1, p = 2, q = 1; p <= n; ++j, p <<= 1,
              q <<= 1)
        for (int i = 0; i + p- 1 < n; ++i)
            M[i][j] = max(M[i][j- 1], M[i + q][j- 1]);
        }
        int query(int b, int e) {
        int k = log2(e- b + 1);
        return max(M[b][k], M[e + 1- (1<<k)][k]);
    }
};
```

7 Strings

7.1 Zalgorithm

```
using namespace std;
#include<bits/stdc++.h>
vector<int> compute_z(const string &s){
    int n = s.size();
    vector<int> z(n,0);
    int l,r;
    r = l = 0;
    for(int i = 1; i < n; ++i){
        if(i > r) {
            l = r = i;
```

```
            while(r < n and s[r- l] == s[r])r++;
            z[i] = r- l;r--;
        }else{
            int k = i-l;
            if(z[k] < r- i +1) z[i] = z[k];
            else {
                l = i;
                while(r < n and s[r- l] == s[r])r++;
                z[i] = r- l;r--;
            }
        }
    }
    return z;
}

signed main(){
    //string line;cin>>line;
    string line = "alfalfa";
    vector<int> z = compute_z(line);
    for(int i = 0; i < z.size(); ++i ){
        if(i)cout<<" ";
        cout<<z[i];
    }
    cout<<endl;
    // must print "0 0 0 4 0 0 1"
    return 0;
}
```

8 Syntax and Headers

8.1 CustomComparator

```
#include <bits/stdc++.h>
using namespace std;
struct cc{
    bool operator()(const int &a, const int &b)
         const{return b<a;}
};
set<int,cc> S;
```

8.2 CustomHash

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define f first
#define s second
#define pl pair<ll, ll>
struct pair_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(const pair<pl, ll>& p) const {
        auto hash1 = hash<ll>{}(p.f.f);
        auto hash2 = hash<ll>{}(p.f.s);
        auto hash3 = hash<ll>{}(p.s);
        return hash1 ^ (hash2 << 1) ^ (hash3 << 2);
        //return splitmix64(x);
    }
};
unordered_map<pair<pl,ll>, ll, pair_hash> dp;
```

8.3 StringBitsetOperations

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

9 Trees

9.1 Centroid Decomposition

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

/*
Xenia the programmer has a tree consisting of n nodes. We
     will consider the tree nodes indexed from 1 to n. We
     will also consider the first node to be initially
     painted red, and the other nodes  to be painted blue.

The distance between two tree nodes v and u is the number
     of edges in the shortest path between v and u.

Xenia needs to learn how to quickly execute queries of two
     types:

paint a specified blue node in red;
calculate which red node is the closest to the given one
     and print the shortest distance to the closest red
     node.
Your task is to write a program which will execute the
     described queries.*/
// a number that is large enough while not causing overflow
const int INF = 1e9;

vector<vector<int>> adj;
vector<int> subtree_size;
// min_dist[v] := the minimal distance between v and a red
     node
vector<int> min_dist;
vector<bool> is_removed;
vector<vector<pair<int, int>>> ancestors;

int get_subtree_size(int node, int parent = -1) {
       subtree_size[node] = 1;
       for (int child : adj[node]) {
            if (child == parent || is_removed[child]) {
                 continue; }
            subtree_size[node] +=
                 get_subtree_size(child, node);
       }
       return subtree_size[node];
}

int get_centroid(int node, int tree_size, int parent = -1)
     {
       for (int child : adj[node]) {
            if (child == parent || is_removed[child]) {
                 continue; }
            if (subtree_size[child] * 2 > tree_size) {
                 return get_centroid(child, tree_size,
                      node);
            }
       }
       return node;
}

/**
 * Calculate the distance between current `node` and the
      `centroid` it belongs
 * to. The distances between a node and all its centroid
      ancestors are stored
 * in the vector `ancestors`.
 * @param cur_dist the distance between `node` and
      `centroid`
 */
void get_dists(int node, int centroid, int parent = -1,
     int cur_dist = 1) {
       for (int child : adj[node]) {
            if (child == parent || is_removed[child]) {
                 continue; }
            cur_dist++;
            get_dists(child, centroid, node, cur_dist);
            cur_dist--;
       }
       ancestors[node].push_back({centroid, cur_dist});
}

void build_centroid_decomp(int node = 0) {
       int centroid = get_centroid(node,
            get_subtree_size(node));

       /*
        * For all nodes in the subtree rooted at
             `centroid`, calculate their
        * distances to the centroid
        */
       for (int child : adj[centroid]) {
            if (is_removed[child]) { continue; }
            get_dists(child, centroid, centroid);
       }

       is_removed[centroid] = true;
       for (int child : adj[centroid]) {
            if (is_removed[child]) { continue; }
            // build the centroid decomposition for all
                 child components
            build_centroid_decomp(child);
```

```
	}
}

/**
 * Paint `node` red by updating all of its ancestors' minimal distances
 * to a red node
 */
void paint(int node) {
	for (auto &[ancestor, dist] : ancestors[node]) {
		min_dist[ancestor] = min(min_dist[ancestor], dist);
	}
	min_dist[node] = 0;
}

/** Print the minimal distance between `node` to a red node */
void query(int node) {
	int ans = min_dist[node];
	for (auto &[ancestor, dist] : ancestors[node]) {
		if (!dist) { continue; }
		/*
		 * The distance between `node` and a red painted node is the sum of
		 * the distance from `node` to one of its ancestors (`dist`) and the
		 * distance from this ancestor to the nearest red node
		 * (`min_dist[ancestor]`).
		 */
		ans = min(ans, dist + min_dist[ancestor]);
	}

	cout << ans << "\n";
}

int main() {
	int N, M;
	cin >> N >> M;

	adj.assign(N, vector<int>());
	for (int i = 0; i < N - 1; i++) {
		int a, b;
		cin >> a >> b;
		a--, b--;
		adj[a].push_back(b);
		adj[b].push_back(a);
	}

	subtree_size.assign(N, 0);
	ancestors.assign(N, vector<pair<int, int>>());
	is_removed.assign(N, false);
	build_centroid_decomp();

	min_dist.assign(N, INF);
	paint(0);
	for (int i = 0; i < M; i++) {
		int t, v;
		cin >> t >> v;
		v--;
		if (t == 1) {
			paint(v);
		} else {
			query(v);
		}
	}
}
```

9.2 HLD Complete

```
template<int SZ, bool VALS_IN_EDGES> struct HLD {
	int N; vi adj[SZ];
	int par[SZ], root[SZ], depth[SZ], sz[SZ], ti;
	int pos[SZ]; vi rpos; // rpos not used but could be useful
	void ae(int x, int y) { adj[x].pb(y), adj[y].pb(x); }
	void dfsSz(int x) {
		sz[x] = 1;
		each(y,adj[x]) {
			par[y] = x; depth[y] = depth[x]+1;
			adj[y].erase(find(all(adj[y]),x)); /// remove parent from adj list
			dfsSz(y); sz[x] += sz[y];
			if (sz[y] > sz[adj[x][0]])
				swap(y,adj[x][0]);
		}
	}
	void dfsHld(int x) {
		pos[x] = ti++; rpos.pb(x);
		each(y,adj[x]) {
			root[y] = (y == adj[x][0] ? root[x] : y);
			dfsHld(y); }
	}
	void init(int _N, int R = 0) { N = _N;
		par[R] = depth[R] = ti = 0; dfsSz(R);
		root[R] = R; dfsHld(R);
	}
	int lca(int x, int y) {
		for (; root[x] != root[y]; y = par[root[y]])
			if (depth[root[x]] > depth[root[y]])
				swap(x,y);
		return depth[x] < depth[y] ? x : y;
	}
	/// int dist(int x, int y) { // # edges on path
	///	return depth[x]+depth[y]-2*depth[lca(x,y)]; }
	LazySeg<ll,SZ> tree; // segtree for sum
	template <class BinaryOp>
	void processPath(int x, int y, BinaryOp op) {
		for (; root[x] != root[y]; y = par[root[y]]) {
			if (depth[root[x]] > depth[root[y]])
				swap(x,y);
			op(pos[root[y]],pos[y]); }
		if (depth[x] > depth[y]) swap(x,y);
		op(pos[x]+VALS_IN_EDGES,pos[y]);
	}
	void modifyPath(int x, int y, int v) {
		processPath(x,y,[this,&v](int l, int r) {
			tree.upd(l,r,v); }); }
	ll queryPath(int x, int y) {
		ll res = 0; processPath(x,y,[this,&res](int l, int r) {
			res += tree.query(l,r); });
		return res; }
	void modifySubtree(int x, int v) {
		tree.upd(pos[x]+VALS_IN_EDGES,pos[x]+sz[x]-1,v);
	}
};
```

9.3 HLD Easy

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

const int N = 2e5 + 5;
const int D = 19;
const int S = (1 << D);

int n, q, v[N];
vector<int> adj[N];

int sz[N], p[N], dep[N];
int st[S], id[N], tp[N];

void update(int idx, int val) {
	st[idx += n] = val;
	for (idx /= 2; idx; idx /= 2) st[idx] = max(st[2 * idx], st[2 * idx + 1]);
}

int query(int lo, int hi) {
	int ra = 0, rb = 0;
	for (lo += n, hi += n + 1; lo < hi; lo /= 2, hi /= 2) {
		if (lo & 1) ra = max(ra, st[lo++]);
		if (hi & 1) rb = max(rb, st[--hi]);
	}
	return max(ra, rb);
}

int dfs_sz(int cur, int par) {
	sz[cur] = 1;
	p[cur] = par;
	for (int chi : adj[cur]) {
		if (chi == par) continue;
		dep[chi] = dep[cur] + 1;
		p[chi] = cur;
		sz[cur] += dfs_sz(chi, cur);
	}
	return sz[cur];
}

int ct = 1;

void dfs_hld(int cur, int par, int top) {
	id[cur] = ct++;
```

```
	tp[cur] = top;
	update(id[cur], v[cur]);
	int h_chi = -1, h_sz = -1;
	for (int chi : adj[cur]) {
		if (chi == par) continue;
		if (sz[chi] > h_sz) {
			h_sz = sz[chi];
			h_chi = chi;
		}
	}
	if (h_chi == -1) return;
	dfs_hld(h_chi, cur, top);
	for (int chi : adj[cur]) {
		if (chi == par || chi == h_chi) continue;
		dfs_hld(chi, cur, chi);
	}
}

int path(int x, int y) {
	int ret = 0;
	while (tp[x] != tp[y]) {
		if (dep[tp[x]] < dep[tp[y]]) swap(x, y);
		ret = max(ret, query(id[tp[x]], id[x]));
		x = p[tp[x]];
	}
	if (dep[x] > dep[y]) swap(x, y);
	ret = max(ret, query(id[x], id[y]));
	return ret;
}

int main() {
	scanf("%d%d", &n, &q);
	for (int i = 1; i <= n; i++) scanf("%d", &v[i]);
	for (int i = 2; i <= n; i++) {
		int a, b;
		scanf("%d%d", &a, &b);
		adj[a].push_back(b);
		adj[b].push_back(a);
	}
	dfs_sz(1, 1);
	dfs_hld(1, 1, 1);
	while (q--) {
		int t;
		scanf("%d", &t);
		if (t == 1) {
			int s, x;
			scanf("%d%d", &s, &x);
			v[s] = x;
			update(id[s], v[s]);
		} else {
			int a, b;
			scanf("%d%d", &a, &b);
			int res = path(a, b);
			printf("%d ", res);
		}
	}
}
```

9.4 LCA

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

int n; int q;
int par[200005][21];
int depth[200005];
vector<int> adj[200005];

void buildArr(int node, int p){
    par[node][0] = p;
    for(int i=1;i<20;i++){
        if(par[node][i-1] != -1){
            par[node][i] = par[par[node][i-1]][i-1];
        }
    }
    if(p == -1) depth[node] = 0;
    else depth[node] = depth[p] + 1;

    for(auto x : adj[node]){
        if(x == p) continue;
        buildArr(x,node);
    }
}


int bigStepper(int node, int k){
    int x = 0;
    for(int i=0;i<20;i++){
        if(k%2==1) node = par[node][i];
        k /= 2;
    }
    return node;
}


int lca(int a, int b){
    if (depth[a] > depth[b]) swap(a,b);

    b = bigStepper(b,depth[b] - depth[a]);
    //cout << b;
    if(a == b) return a;
    for(int i=19;i>=0;i--){
        if(par[a][i] != par[b][i]){
            a = par[a][i];
            b = par[b][i];
        }
    }
    return par[a][0];
}
signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    #ifndef ONLINE_JUDGE
    freopen("file.txt", "r", stdin);
    #endif
    cin >> n >> q;

    for(int i=0;i<=n;i++){
        for(int j=0;j<20;j++){
            par[i][j] = -1;
        }
    }

    for(int i=0;i<n-1;i++){
        int a; int b; cin >> a >> b;
        adj[a].pb(b);
        adj[b].pb(a);
    }

    buildArr(1,-1);
    for(int i =0;i<q;i++){
        int a; int b; cin >> a >>b;
        cout << depth[a] + depth[b] - 2*depth[lca(a,b)] <<
             "\n";
    }
}
```