Team notebook

November 7, 2024

Contents

1 Conjuntos **1**
- 1.1 generarConjuntos . . . 1
- 1.2 permutaciones . . . 2

2 data-structures **2**
- 2.1 BITRange . . . 2
- 2.2 fenwikTree . . . 2
- 2.3 segmentTree . . . 2
- 2.4 segmentTreeDinamicLazy . . . 3
- 2.5 sparceTable . . . 4
- 2.6 UnionFind . . . 4

3 ejerciciosCompl **4**
- 3.1 areaPoligonoYVerificarPoli . . . 4

4 geometry **6**
- 4.1 convexHull . . . 6
- 4.2 pointLine . . . 7
- 4.3 puntoDentroLog . . . 7

5 graphs **9**
- 5.1 bellman-ford . . . 9
- 5.2 bfs . . . 9
- 5.3 dijkstra . . . 10
- 5.4 eulerTour . . . 11
- 5.5 FindingBridge . . . 11
- 5.6 Floyd–Warshall . . . 11
- 5.7 fordFulkerson . . . 12
- 5.8 kruscal . . . 13
- 5.9 lca . . . 13
- 5.10 ordenTopologico . . . 14
- 5.11 PlanarGraph . . . 14

6 Math **16**
- 6.1 Exponentiation . . . 16
- 6.2 floorCeil . . . 17
- 6.3 sieveEratosthenes . . . 17

7 String **18**
- 7.1 KMP . . . 18
- 7.2 manacher . . . 18
- 7.3 rabinKarp . . . 19
- 7.4 stringHashing . . . 19

1 Conjuntos

1.1 generarConjuntos

```
void search(int k) {
    if (k == n) {
        // process subset
    } else {
        search(k+1);
        subset.push_back(k);
        search(k+1);
        subset.pop_back();
    }
}
```

1.2 permutaciones

```
void search() {
    if (permutation.size() == n) {
        // process permutation
    } else {
        for (int i = 0; i < n; i++) {
            if (chosen[i]) continue;
                chosen[i] = true;
                permutation.push_back(i);
                search();
                chosen[i] = false;
                permutation.pop_back();
        }
    }
}
```

2 data-structures

2.1 BITRange

```
struct BITRange {
        BIT a,b;//pos desde 1
    void build(int tam){
        a.build(tam+10);
        b.build(tam+10);
    }
        int sum(int x) {
                return a.sum(x)*x+b.sum(x);
        }
        void update(int l, int r, int v) {
                a.add(l,v), a.add(r+1,-v);
                b.add(l,-v*(l-1)), b.add(r+1,v*r);
        }
};
```

2.2 fenwikTree

```
struct BIT{
    int n;//pos desde 1
    vi bit;
    void build(int tam){
        n = tam+10;
        bit.resize(n,0);
    }
    int sum(int idx) {
        int ret = 0;
        for (++idx; idx > 0; idx -= idx & -idx)
            ret += bit[idx];
        return ret;
    }

    void add(int pos, int val){
        pos++;
        int tam = n+1;
        while(pos <= tam){
            bit[pos] += val;
            pos += pos & (-pos);
        }
    }

};
```

2.3 segmentTree

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
const int N = 10e5+2;
int n, st[4*N], v[N];
void build(int p, int l, int r){
    if(l == r) st[p] = v[l];
    else{
        build(2*p,l,(l+r)/2);
        build(2*p+1,(l+r)/2+1,r);
        st[p] = st[2*p]*st[2*p+1];
    }
}
int query(int a, int b, int p, int l, int r){
    if(a>b) return 1;
    if(l == a && r == b) return st[p];
    int mid = (l+r) / 2;
    return (query(a,min(mid,b),2*p,l,mid))
```

```
        *(query(max(a,mid+1),b,2*p+1,mid+1,r));
}
void update(int pos, int val, int p, int l, int r){
    if(l == r) st[p] = val;
    else{
        int mid = (l+r)/2;
    if(pos<=mid){
        update(pos, val, 2*p, l, mid);
    }else{
        update(pos, val, 2*p+1,mid+1,r);
    }
    st[p] = st[2*p] * st[2*p+1];
    }
}
int main(){
     return 0;
}
```

2.4 segmentTreeDinamicLazy

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

vector<int> e, d, mx, lazy;
//begin creating node 0, then start your segment tree creating node 1
int create(){
        mx.push_back(0);
        lazy.push_back(0);
        e.push_back(0);
        d.push_back(0);
        return mx.size() - 1;
}

void push(int pos, int ini, int fim){
        if(pos == 0) return;
        if (lazy[pos]) {
                mx[pos] += lazy[pos];
                // RMQ (max/min) -> update: = lazy[p],      incr: +=
                    lazy[p]
                // RSQ (sum)     -> update: = (r-l+1)*lazy[p], incr: +=
                    (r-l+1)*lazy[p]
                // Count lights on -> flip: = (r-l+1)-st[p];
                if (ini != fim) {
                        if(e[pos] == 0){
                                int aux = create();
                                e[pos] = aux;
                        }
                        if(d[pos] == 0){
                                int aux = create();
                                d[pos] = aux;
                        }
                        lazy[e[pos]] += lazy[pos];
                        lazy[d[pos]] += lazy[pos];
                        // update:   lazy[2*p] = lazy[p], lazy[2*p+1] =
                            lazy[p];
                        // increment: lazy[2*p] += lazy[p], lazy[2*p+1] +=
                            lazy[p];
                        // flip:     lazy[2*p] ^= 1,     lazy[2*p+1] ^= 1;
                }
                lazy[pos] = 0;
        }
}

void update(int pos, int ini, int fim, int p, int q, int val){
        if(pos == 0) return;

        push(pos, ini, fim);

        if(q < ini || p > fim) return;

        if(p <= ini and fim <= q){
                lazy[pos] += val;
                // update:   lazy[p] = k;
                // increment: lazy[p] += k;
                // flip:     lazy[p] = 1;
                push(pos, ini, fim);
                return;
        }

        int m = (ini + fim) >> 1;
        if(e[pos] == 0){
                int aux = create();
                e[pos] = aux;
        }
        update(e[pos], ini, m, p, q, val);
        if(d[pos] == 0){
                int aux = create();
                d[pos] = aux;
        }
```

```
        update(d[pos], m + 1, fim, p, q, val);
        mx[pos] = max(mx[e[pos]], mx[d[pos]]);
}

int query(int pos, int ini, int fim, int p, int q){
        if(pos == 0) return 0;

        push(pos, ini, fim);

        if(q < ini || p > fim) return 0;

        if(p <= ini and fim <= q) return mx[pos];

        int m = (ini + fim) >> 1;
        return max(query(e[pos], ini, m, p, q) , query(d[pos], m + 1, fim,
            p, q));
}
```

2.5 sparceTable

```
int n;
const int MAX_N = 1e5; // N <= 100,000
const int LOG = ceil(log2(MAX_N));
int sparse[MAX_N][LOG];
int v[MAX_N];

void build() {
    for (int i = 0; i < n; i++)
        sparse[i][0] = v[i];

    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 0; i + (1 << j) <= n; i++) {
            sparse[i][j] = min(sparse[i][j - 1], sparse[i + (1 << (j -
                1))][j - 1]);
        }
    }
}

int query(int a, int b) {
    int pot = 32 - __builtin_clz(b - a + 1) - 1;
    return min(sparse[a][pot], sparse[b - (1 << pot) + 1][pot]);
}
```

2.6 UnionFind

```
#include <bits/stdc++.h>
#define mp make_pair
#define pb push_back
#define f first
#define s second
#define st first
#define nd second
using namespace std;
const int N = 1000;
vector<int> par(N,-1),sz(N,1);
vector <pair<int, pair<int, int>>> edges;
int cost = 0;
int v,a;
//par: vector of parents                                              *
//sz: vector of subsets sizes, i.e. size of the subset a node is in
int find(int a) { return par[a] == -1 ? a : par[a] = find(par[a]); }

void unite(int a, int b) {
  if ((a = find(a)) == (b = find(b))) return;
  if (sz[a] < sz[b]) swap(a, b);
  par[b] = a; sz[a] += sz[b];
}


int main () {

    return 0;
}
```

3 ejerciciosCompl

3.1 areaPoligonoYVerificarPoli

```
#include <bits/stdc++.h>
#define int long long int
#define endl '\n'
#define pb push_back
using namespace std;
const int INF = 1e9;
```

```
struct pt {
    int x, y;
};

int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1;
    if (v > 0) return +1;
    return 0;
}

bool cw(pt a, pt b, pt c, bool bb) {
    int o = orientation(a, b, c);
    return o < 0 || (bb && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<pt>& a, bool bb) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (bb) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], bb))
            st.pop_back();
        st.push_back(a[i]);
    }

    a = st;
}
```

```
double polygonArea(const vector<pt>& p) {
    double area = 0;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        area += p[i].x * p[j].y;
        area -= p[j].x * p[i].y;
    }
    area = fabs(area) / 2.0;
    return area;
}

bool isIn(const vector<pt>& v, pt p) {
    int n = v.size();
    if (n < 3) return false;
    double angleSum = 0;

    for (int i = 0; i < n; i++) {
        pt a = v[i];
        pt b = v[(i + 1) % n];
        double angle = atan2(b.y - p.y, b.x - p.x) - atan2(a.y - p.y, a.x
            - p.x);
        if (angle >= M_PI) angle -= 2 * M_PI;
        if (angle <= -M_PI) angle += 2 * M_PI;
        angleSum += angle;
    }

    return fabs(fabs(angleSum) - 2 * M_PI) < 1e-9;
}



signed main(){
    int n;
    int p = 0;
    vector<vector<pt>> v(20);
    while(cin >> n and n){
        for(int i = 0,a,b;i<n;i++){
            cin >> a >> b;
            //if(i==0)continue;
            v[p].pb({a,b});
        }
        convex_hull(v[p],false);
        p++;
    }
```

```
    int x,y;
    double ans = 0;
    bool vis[p] = {false};
    while(cin >> x >> y){
        for(int i = 0;i<p;i++){
            //if(vis[i])continue;
            pt p = {x,y};
            vector<pt> pp = v[i];
            if(vis[i])continue;
            if(isIn(pp,p)){
                vis[i]=1;
                ans+=polygonArea(v[i]);
            }

        }
    }

    cout << fixed <<setprecision(2)<<ans << endl;

}
```

4 geometry

4.1 convexHull

```
#include <bits/stdc++.h>
#define fast ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define fastIO ios_base::sync_with_stdio(false); cin.tie(NULL);
    cout.tie(NULL);
#define int long long int
#define mod (int)1e9+7
#define endl '\n'
#define arr array
#define pb push_back
using namespace std;
const int INF = 1e9;
//typedef long double ld;
struct pt {
    int x, y;
};

int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<pt>& a, bool include_collinear) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i],
            include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    a = st;
}
signed main(){
    int n;cin >> n;
    vector<pt> v;
    for(int i = 0;i<n;i++){
        int a,b;
        cin >> a >> b;
```

```
        v.push_back({a,b});
    }
    //for(auto it:v)cout << it.x << " " << it.y << endl;
    convex_hull(v,true);
    //cout << "sssssssssss" << endl;
    cout << v.size() << endl;
    for(auto it:v)cout << it.x << " " << it.y << endl;
}
```

4.2 pointLine

```
#include <bits/stdc++.h>
using namespace std;
#define conDec(numero,i) fixed << setprecision(i) << numero
const double EPS = 1e-9;
struct point{
    double x, y;
    point(double x,double y): x(x),y(y){}
    point operator+(point b){return {x+b.x,y+b.y};}
    point operator -(point b){return {x-b.x,y-b.y};}
    point operator*(double d) {return {x*d, y*d};}
    point operator/(double d) {return {x/d, y/d};}
    point translate(point v,point p){return p+v;}
    point scale(point c, double factor, point p) {return c+(p-c)*factor;}
};

double dot(point a,point b){
    return a.x*b.x + a.y*b.y;
}
double abs(point a){
    return sqrt(dot(a,a));
}
double proj(point a,point b){
    return dot(a,b)/abs(b);
}
double angle(point a, point b) {
    double ans = acos(dot(a, b) / abs(a) / abs(b));//rad
    //ans *= 180.0/M_PI;
    return ans;
}
```

```
double dist(point a,point b){
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
bool operator==(point a, point b) {return a.x == b.x && a.y == b.y;}
bool operator!=(point a,point b) {return !(a==b);}


double roundToThreeDecimals(double value) {
    return round(value * 1000.0) / 1000.0;
}

double cross(point a, point b){
    return a.x*b.y-a.y*b.x;
}
struct line{
    point v; double c;
    // From direction vector v and offset c
    line(point v, double c) : v(v), c(c) {}
    // From equation ax+by=c
    line(double a, double b, double c) : v({b,-a}), c(c) {}
    // From points P and Q
    line(point p, point q) : v(q-p), c(cross(v,p)) {}

    bool contains(const point& r){
        // Comprueba si el producto cruzado de v y (r - cualquier punto en
            la lnea) es cero
        return fabs(cross(v, r) - c) < EPS;
    }
};
point inter(line l1, line l2) {
    double d = cross(l1.v, l2.v);
    //if (d == 0) return false;
    point res = ((l2.v*l1.c - l1.v*l2.c) / d);
    return ((l2.v*l1.c - l1.v*l2.c) / d);
}
int main(){
    point a(3,4),b(3,0);
    cout << angle(a,b) << endl;
}
```

4.3 puntoDentroLog

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
struct pt {
    long long x, y;
    pt() {}
    pt(long long _x, long long _y) : x(_x), y(_y) {}
    pt operator+(const pt &p) const { return pt(x + p.x, y + p.y); }
    pt operator-(const pt &p) const { return pt(x - p.x, y - p.y); }
    long long cross(const pt &p) const { return x * p.y - y * p.x; }
    long long dot(const pt &p) const { return x * p.x + y * p.y; }
    long long cross(const pt &a, const pt &b) const { return (a -
        *this).cross(b - *this); }
    long long dot(const pt &a, const pt &b) const { return (a -
        *this).dot(b - *this); }
    long long sqrLen() const { return this->dot(*this); }
};

bool lexComp(const pt &l, const pt &r) {
    return l.x < r.x || (l.x == r.x && l.y < r.y);
}

int sgn(long long val) { return val > 0 ? 1 : (val == 0 ? 0 : -1); }

vector<pt> seq;
pt translation;
int n;

bool pointInTriangle(pt a, pt b, pt c, pt point) {
    long long s1 = abs(a.cross(b, c));
    long long s2 = abs(point.cross(a, b)) + abs(point.cross(b, c)) +
        abs(point.cross(c, a));
    return s1 == s2;
}

void prepare(vector<pt> &points) {
    n = points.size();
    int pos = 0;
    for (int i = 1; i < n; i++) {
        if (lexComp(points[i], points[pos]))
            pos = i;
    }
    rotate(points.begin(), points.begin() + pos, points.end());
    n--;
    seq.resize(n);
    for (int i = 0; i < n; i++)
        seq[i] = points[i + 1] - points[0];
    translation = points[0];
}

bool pointInConvexPolygon(pt point) {
    point = point - translation;
    if (seq[0].cross(point) != 0 &&
            sgn(seq[0].cross(point)) != sgn(seq[0].cross(seq[n - 1])))
        return false;
    if (seq[n - 1].cross(point) != 0 &&
            sgn(seq[n - 1].cross(point)) != sgn(seq[n - 1].cross(seq[0])))
        return false;

    if (seq[0].cross(point) == 0)
        return seq[0].sqrLen() >= point.sqrLen();

    int l = 0, r = n - 1;
    while (r - l > 1) {
        int mid = (l + r) / 2;
        int pos = mid;
        if (seq[pos].cross(point) >= 0)
            l = mid;
        else
            r = mid;
    }
    int pos = l;
    return pointInTriangle(seq[pos], seq[pos + 1], pt(0, 0), point);
}

int main() {
    // Definir el polgono convexo
    vector<pt> polygon = {pt(0, 0), pt(4, 0), pt(4, 4), pt(0, 4)};

    // Preparar la estructura para el polgono convexo
    prepare(polygon);

    // Definir algunos puntos de prueba
    vector<pt> testPoints = {pt(2, 2), pt(5, 5), pt(0, 0), pt(4, 4),
        pt(2, 0)};

    // Probar si los puntos estn dentro del polgono convexo
```

```
    for (pt point : testPoints) {
        bool inside = pointInConvexPolygon(point);
        cout << "Point (" << point.x << ", " << point.y << ") is "
             << (inside ? "inside" : "outside") << " the polygon." << endl;
    }

    return 0;
}
```

5 graphs

5.1 bellman-ford

```
#include <bits/stdc++.h>
using namespace std;
#define int long long int
const int INF = 1e10;
const int N =1e4; // Mximo nmero de nodos
vector<int> adj[N], adjw[N]; // Lista de adyacencia y pesos de las aristas
int dist[N]; // Distancias desde la fuente
int n, m; // Nmero de nodos y aristas
void bellmanFord(int source) {
   fill(dist, dist + N, INF); // Inicializar distancias con un valor
       grande
    dist[source] = 0; // Distancia desde la fuente a s mismo es 0
    for (int i = 0; i < n - 1; ++i) {
        for (int u = 1; u <= n; ++u) {
            for (size_t j = 0; j < adj[u].size(); ++j) {
                int v = adj[u][j], w = adjw[u][j];
                if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                }
            }
        }
    }
}
signed main() {
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back(v);
        adjw[u].push_back(w);
    }
    bellmanFord(1);
}
```

5.2 bfs

```
#include <bits/stdc++.h>
#define fast ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define fastIO ios_base::sync_with_stdio(false); cin.tie(NULL);
    cout.tie(NULL);
#define int long long int
#define mod (int)1e9+7
#define endl '\n'
#define arr array
#define pb push_back
using namespace std;
int const nxm = 1e5;
signed main(){
    fastIO;
    int n,m;
    cin >> n >> m;
    vector<vector<int>> g(n);
    int a,b;
    for(int i=0;i<m;i++){
        cin >> a >> b;a--,b--;
        g[a].push_back(b);
        g[b].push_back(a);
    }

    vector<int> p(n,-1);
    queue<int> qu;
    p[0]=-2;
    qu.push(0);
    while(qu.size()){
        int u = qu.front();
        qu.pop();
        for(int v:g[u]){
            if(p[v]<0){
                p[v] = u;
                qu.push(v);
            }
        }
```

```
    }
    vector<int> ans;
    if(p[n-1]<0){
        cout << "IMPOSSIBLE";
    }else{
        int v = n-1;
        while(v){
            ans.pb(v);
            v=p[v];
        }
        ans.pb(0);
        reverse(ans.begin(),ans.end());
        cout << ans.size() << endl;
        for(int i:ans){
            cout << i+1 << " ";
        }
    }

}
```

5.3 dijkstra

```
#include <bits/stdc++.h>
#define int ll
#define s second
#define f first
typedef long long ll;
using namespace std;
typedef pair<int,int> ii;
const int N = 200010;
//dijkstra
//complejidad O(n + m log m)
vector<int> dis(N, 1e12);//almacena las distancias
vector<int> dijkstra(vector<vector<pair<int,int>>> &g,int r){
    int n = g.size();
    vector<int> par(n,-1);
    vector<bool> vis(n);
    priority_queue<pair<int,int>> cola;
    cola.push({0,r});
    dis[r] = 0;
    while(!cola.empty()){
        int node = cola.top().second;
        cola.pop();
        if(vis[node]) continue;
        vis[node] = 1;
        for(ii ed: g[node]){
            if(dis[ed.f] > dis[node] + ed.s){
                par[ed.f] = node;
                dis[ed.f] = dis[node] + ed.s;
                cola.push({-dis[ed.f], ed.f});
            }
        }
    }
    return par;

}
signed main(){
    //v->vertices
    //a->aristas
    //o->origen
    //d->destino
    int v,a,o,d,c;cin >> v >> a;
    vector<vector<pair<int,int>>> g(v);
    for(int i = 0; i < a;i++){
        cin >> o >> d >> c;
        o--;d--;
        g[o].push_back({d,c});
        //g[d].push_back({o,c});
    }
    vector<int> par = dijkstra(g,0);
    //for(int &i:par) cout << i << endl;//mostrar padres
    if(dis[v - 1] == 1e12)
        cout<<-1<<'\n';
    else
    {
        vector<int> path;
        int nodo = v - 1;//camino que queremos retornar
        while(nodo != -1)
        {
            path.push_back(nodo + 1);
            nodo = par[nodo];
        }
        reverse(path.begin(),path.end());
        for(int x : path)
            cout<<x<<' ';
        cout<<'\n';
    }
```

```
    for(int i = 0;i<v;i++){
        cout << dis[i] << endl;
    }
    return 0;
}
```

5.4 eulerTour

```
vi w(N), start(N),fin(N);
int t = 0;//N 2e5+5 sbt
void euler_tour(int r, int p, vector<vi> &v){
    start[r] = ++t;
    for(int i:v[r]){
        if(i==p)continue;
        euler_tour(i,r,v);
    }
    fin[r] = t;
}
```

5.5 FindingBridge

```
#include <bits/stdc++.h>
#define vi vector<int>
using namespace std;
const int N = 1e5+5, INF = 1e18;
vector<vi> g(N);
int ans,timer = 1,n;
vi vis,low;
vector<vi> v;
//Buscar puentes en un grafo en O(n+m)
void dfs(int i,int p = -1){
    //int c = 1;
    vis[i] = low[i] = timer++;
    for(int j:v[i]){
        if(j!=p){
            if(vis[j]) low[i] = min(low[i],vis[j]);
            else{
                int count = dfs(j,i);
                if(vis[i]<low[j]){
                    //proceso con el puente
                }else low[i] = min(low[i],low[j]);
                //c+=count;
            }
        }
    }
    //return c;
}
```

5.6 Floyd–Warshall

```
#include <bits/stdc++.h>
#define int ll
#define s second
#define f first
#define ing long long int
typedef long long ll;
using namespace std;
//algoritmo de FloydWarshall
//complejidad O(n*n*n)
typedef pair<int,int> ii;
const int INF = 1e18;
signed main(){
    int n,m,q;
    cin >> n >> m >> q;
    int v[n+10][n+10] = {0},dist[n+10][n+10] = {0};
    for(int i = 0,a,b,c;i<m;i++){
        cin >> a >> b >> c;
        if(v[a][b] && c<v[a][b]){
            v[a][b]=c;
            v[b][a]=c;
        }else if(!v[a][b]){
            v[a][b]=c;
            v[b][a]=c;
        }
    }
    //llean la matriz con lo que debe estar
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == j) dist[i][j] = 0;
            else if (v[i][j]) dist[i][j] = v[i][j];
            else dist[i][j] = INF;
        }
    }
```

```
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                dist[i][j] = min(dist[i][j],dist[i][k]+dist[k][j]);
            }
        }
    }
    for(int i = 0,a,b; i<q;i++){
        cin >> a >> b;
        cout << (dist[a][b]>=1e18?-1:dist[a][b]) << endl;
    }

}
```

5.7 fordFulkerson

```
#include<bits/stdc++.h>
#define endl '\n'
#define int long long int
#define fast ios_base::sync_with_stdio(0), cin.tie(0), cout.tie(0)
#define pb push_back
#define debug cout<<'*'<<'\n'
#define arr array
using namespace std;
const int INF = 1e19;
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});
    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
signed main() {
    fast;
    int m;
    cin >> n >> m;
    capacity.assign(n, vector<int>(n, 0));
    adj.assign(n, vector<int>());
    for (int i = 0; i < m; i++) {
        int u, v, c;
        cin >> u >> v >> c;
        u--,v--;
        capacity[u][v] += c;
        adj[u].pb(v);
        adj[v].pb(u); // Agregar la inversa tambin para el grafo residual
    }

    cout << maxflow(0, n-1) << endl;
```

```
}
```

5.8 kruscal

```
#include <bits/stdc++.h>
#define fast ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define fastIO ios_base::sync_with_stdio(false); cin.tie(NULL);
    cout.tie(NULL);
#define int long long
#define endl '\n'
#define arr array
#define nd second
#define st first
using namespace std;
const int mod = 1e9+7,INF=1e9;
const int N = 2e5+100;

/*******************************************************************************
* KRUSKAL'S ALGORITHM (MINIMAL SPANNING TREE - INCREASING EDGE SIZE)
    *
* Time complexity: O(ElogE)
    *
* Usage: cost, sz[find(node)]
    *
* Notation: cost: sum of all edges which belong to such MST
    *
*          sz:  vector of subsets sizes, i.e. size of the subset a node
    is in *
*******************************************************************************/
vector<int> par(N,-1),sz(N,1);

int n,m;


int cost = 0;
vector <pair<int, pair<int, int>>> edges; //mp(dist, mp(node1, node2))
int find(int a) { return par[a] == -1 ? a : par[a] = find(par[a]); }

void unite(int a, int b) {
  if ((a = find(a)) == (b = find(b)))return;
  if (sz[a] < sz[b]) swap(a, b);
  par[b] = a; sz[a] += sz[b];
}

signed main(){
    fast;
    cin >> n >> m;
    for(int i = 0;i < m;i++){
         int a,b,tam;
         cin >> a >> b >> tam;
         edges.push_back({tam,{a,b}});
    }


    sort(edges.begin(), edges.end());
    for (auto e : edges)
        if (find(e.nd.st) != find(e.nd.nd))
            unite(e.nd.st, e.nd.nd), cost += e.st;
    int ans = 0;
    for(int i = 1; i<=n;i++)ans+=(par[i]==-1?1:0);
    if(ans>1){
        cout << "IMPOSSIBLE" << endl;
        return 0;
    }
    cout << cost << endl;
}
```

5.9 lca

```
int timer = 1; //al final timer es 2*n;build(1,1,timer);
//depth[a]+depth[b]-2*depth[lca(a,b)] = dist
int ini[N], euler[N], st[4*N];
int mnTin(int x,int y){
    if (x == -1)return y;
    if( y == -1)return x;
    return (ini[x]<ini[y]?x:y);
}
void build(int p,int l, int r){
    if(l==r)st[p] = euler[l];
    else{
        build(p*2,l,(l+r)/2);
        build(p*2+1,(l+r)/2+1,r);
        st[p] = mnTin(st[p*2],st[2*p+1]);
    }
}

int query(int a,int b,int p, int l, int r){
```

```
    if(l>b || r<a)return -1;
    if(l>=a and r<= b)return st[p];
    int mid = (l+r)/2;
    return mnTin(query(a,b,2*p,l,mid),query(a,b,2*p+1,mid+1,r));
}
void dfs(int r, int p,vector<vi> &v){
    ini[r] = timer;
    euler[timer++] = r;
    for(int i:v[r]){
        if(i==p)continue;
        dfs(i,r,v);
        euler[timer++] = r;
    }
}
int lca(int a,int b){
    if(ini[a]>ini[b])swap(a,b);
    return query(ini[a],ini[b],1,1,timer);
}
```

5.10 ordenTopologico

```
#include <bits/stdc++.h>
#define fast ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define fastIO ios_base::sync_with_stdio(false); cin.tie(NULL);
    cout.tie(NULL);
#define int long long int
#define mod (int)1e9+7
#define endl '\n'
#define arr array
#define pb push_back
using namespace std;
const int INF = 1e9;
vector<int> ans;
vector<bool> visited, inStack;
vector<vector<int>> g;
int n, m;
bool cycle;

void dfs(int v) {
    visited[v] = true;
    inStack[v] = true;
    for (int u : g[v]) {
        if (!visited[u])
            dfs(u);
        else if (inStack[u])
            cycle = true;
    }
    inStack[v] = false;
    ans.push_back(v+1);
}

void topological_sort() {
    visited.assign(n, false);
    inStack.assign(n, false);
    ans.clear();
    cycle = false;
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());

    if (cycle) {
        cout << "IMPOSSIBLE" << endl;
        return;
    }
}

signed main(){
    cin >> n >> m;
    g.resize(n);
    for(int i = 0, a, b; i < m; i++){
        cin >> a >> b, a--, b--;
        g[a].push_back(b);
    }
    topological_sort();
    if (!cycle) {
        for(int i : ans){
            cout << i << " ";
        }
    }
    return 0;
}
```

5.11 PlanarGraph

```
const int tam = 1000 + 10;

struct Point {
    int x, y;

    Point(int x_ = 0, int y_ = 0) : x(x_), y(y_) {}

    Point operator - (const Point & p) const {
        return Point(x - p.x, y - p.y);
    }

    bool operator < (const Point & other) const {
        if (x != other.x) return x < other.x;
        return y < other.y;
    }

    int cross(const Point & p) const {
        return x * p.y - y * p.x;
    }

    int cross(const Point & p, const Point & q) const {
        return (p - *this).cross(q - *this);
    }

    int half() const {
        return int(y < 0 || (y == 0 && x < 0));
    }
};



vector<vector<int>> find_faces(vector<Point> vertices,
     vector<vector<int>> adj) {
    int n = vertices.size();
    vector<vector<char>> used(n);
    for (int i = 0; i < n; i++) {
        used[i].resize(adj[i].size(), 0);
        auto compare = [&](int l, int r) {
            Point pl = vertices[l] - vertices[i];
            Point pr = vertices[r] - vertices[i];
            if (pl.half() != pr.half())
                return pl.half() < pr.half();
            return pl.cross(pr) > 0;
        };
        sort(adj[i].begin(), adj[i].end(), compare);
    }

    vector<vector<int>> faces;
    for (int i = 0; i < n; i++) {
        for (int edge_id = 0; edge_id < adj[i].size(); edge_id++) {
            if (used[i][edge_id]) {
                continue;
            }
            vector<int> face;
            int v = i;
            int e = edge_id;
            while (!used[v][e]) {
                used[v][e] = true;
                face.push_back(v);
                int u = adj[v][e];
                int e1 = -1;
                for (int j = 0; j < adj[u].size(); j++) {
                    if (adj[u][j] == v) {
                        e1 = (j + 1) % adj[u].size();
                        break;
                    }
                }
                if (e1 == -1) break;
                v = u;
                e = e1;
            }
            if (!face.empty()) {
                faces.push_back(face);
            }
        }
    }
    vector<pair<double, vector<int>>> face_areas;
    for (auto &face : faces) {
        vector<Point> polygon;
        for (int idx : face) {
            polygon.push_back(vertices[idx]);
        }
        double area = polygonArea(polygon);
        face_areas.push_back({area, face});
    }


    sort(face_areas.begin(), face_areas.end());
    face_areas.pop_back();
```

```
    vector<vector<int>> result;
    for (const auto &entry : face_areas) {
        result.push_back(entry.second);
    }

    return result;
}
double polygonArea(const vector<Point>& p) {
    double area = 0;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        area += p[i].x * p[j].y;
        area -= p[j].x * p[i].y;
    }
    area = fabs(area) / 2.0;
    return area;
}
void solve() {
    int n;
    cin >> n;
    vector<Point> vertices;
    vector<vector<int>> adj(n);
    map<Point, int> mp;
    map<int,Point> mp2;
    int nodo = 0;
    for (int i = 0, a, b, c, d; i < n; i++) {
        cin >> a >> b >> c >> d;
        Point a1(a, b), a2(c, d);
        int p1, p2;

        if (mp.find(a1) != mp.end()) {
            p1 = mp[a1];
        } else {
            p1 = nodo;
            mp[a1] = p1;
            mp2[p1] = a1;
            nodo++;
            vertices.pb(a1);
        }

        if (mp.find(a2) != mp.end()) {
            p2 = mp[a2];
        } else {
            p2 = nodo;
            mp2[p2] = a2;
            mp[a2] = p2;
            nodo++;
            vertices.pb(a2);
        }
        adj[p1].pb(p2);
        adj[p2].pb(p1);
    }

    vector<vi> faces = find_faces(vertices, adj);

    double ans = 0;
    for(auto v:faces){
        vector<Point> pol;
        for(auto i:v){
            pol.pb(mp2[i]);
        }
        double aux = polygonArea(pol);
        ans+=(aux*aux);
    }
    cout << fixed << setprecision(6) << ans << endl;
}
```

6 Math

6.1 Exponentiation

```
#include <bits/stdc++.h>
#define fast ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define fastIO ios_base::sync_with_stdio(false); cin.tie(NULL);
    cout.tie(NULL);
#define int long long int
#define endl '\n'
#define arr array
#define pb push_back
using namespace std;
int const mod= 1e9+7;
int pt(int a,int b,int m){
    int r = 1;
    while(b){
        if(b&1)
```

```
            r=r*a%m;
        a=a*a%m;
        b/=2;
    }
    return r;
}
signed main(){
    fastIO;
    int n;
    cin >> n;
    while(n--){
        int a,b;cin >> a >> b;
        cout << pt(a,b,mod) << endl;
    }
}
```

6.2 floorCeil

```
#include <bits/stdc++.h>
#define fast ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define fastIO ios_base::sync_with_stdio(false); cin.tie(NULL);
    cout.tie(NULL);
#define int long long int
#define endl '\n'
#define arr array
#define pb push_back
using namespace std;
int const mod= 1e9+7;
int pt(int a,int b,int m){
    int r = 1;
    while(b){
        if(b&1)
            r=r*a%m;
        a=a*a%m;
        b/=2;
    }
    return r;
}
signed main(){
    fastIO;
    int n;
    cin >> n;
    while(n--){
        int a,b;cin >> a >> b;
        cout << pt(a,b,mod) << endl;
    }
}
```

6.3 sieveEratosthenes

```
#include <bits/stdc++.h>
#define fast ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define fastIO ios_base::sync_with_stdio(false); cin.tie(NULL);
    cout.tie(NULL);
#define int long long int
#define endl '\n'
#define arr array
#define pb push_back
using namespace std;
int const mod= 1e9+7;
int const N = 1e6;
//vector<int> lpf(N+1,0);
int lpf[N+1] ;
vector<int> pfs;
signed main(){
    fastIO;
    for(int i=2;i<=N;i++){
        if(!lpf[i]){
            pfs.pb(i);
            lpf[i] = i;
        }
        for(int j = 0;j<pfs.size() && pfs[j]<=lpf[i]&&i*pfs[j]<=N;j++)
            lpf[i*pfs[j]] = pfs[j];
    }
    int n;
    cin >> n;
    //cout << "hola" << endl;
    /*for(int i=0;i<100;i++){
        cout << lpf[i] << endl;
    }*/
    while(n--){
        map<int,int> mp;
        int x;
        cin >> x;
        while(x>1){
            mp[lpf[x]]++;
```

```
            x/=lpf[x];
           //cout << x << endl;
        }
        int ans = 1;
        for(auto it:mp)
            ans*=(it.second+1);
        cout << ans << endl;
    }
}
```

7 String

7.1 KMP

```
#include <bits/stdc++.h>
using namespace std;
int const N = 1000;
string s,p;
int b[N], n, m; // n = strlen(s), m = strlen(p);

void kmppre() {
       b[0] = -1;
       for (int i = 0, j = -1; i < m; b[++i] = ++j)
              while (j >= 0 and p[i] != p[j])
                     j = b[j];
}

void kmp() {
       for (int i = 0, j = 0; i < n;) {
              while (j >= 0 and s[i] != p[j]) j=b[j];
              i++, j++;
              if (j == m) {
                     // match position i-j
           cout << i-j << endl;

                     j = b[j];
              }
       }

}
int main(){
    cin >> s >> p;
    n = s.size(),m = p.size();
    kmppre();

    for(int i = 0;i<=m;i++)cout << b[i] << " ";
    cout << endl;

    kmp();
}
```

7.2 manacher

```
#include <bits/stdc++.h>
#define fast ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define fastIO ios_base::sync_with_stdio(false); cin.tie(NULL);
    cout.tie(NULL);
#define int long long int
#define mod (int)1e9+7
#define endl '\n'
#define arr array
#define s second
#define pb push_back
#define ii pair<int,int>
#define vii vector<ii>
#define vi vector<int>
using namespace std;
const int INF = 1e9, N = 1e6+10;
// Mancher O(n)

vector<int> d1,d2;
// d1 -> odd : size = 2 * d1[i] - 1, palindrome from i - d1[i] + 1 to i +
    d1[i] - 1
// d2 -> even : size = 2 * d2[i], palindrome from i - d2[i] to i + d2[i]
    - 1

void manacher(string &s) {
    int n = s.size();
       d1.resize(n), d2.resize(n);
       for(int i = 0, l1 = 0, l2 = 0, r1 = -1, r2 = -1; i < n; i++) {
              if(i <= r1) {
                     d1[i] = min(d1[r1 + l1 - i], r1 - i + 1);
              }

              if(i <= r2) {
```

```
            d2[i] = min(d2[r2 + l2 - i + 1], r2 - i + 1);
        }
        while(i - d1[i] >= 0 and i + d1[i] < n and s[i - d1[i]] ==
            s[i + d1[i]]) {
            d1[i]++;
        }

        while(i - d2[i] - 1 >= 0 and i + d2[i] < n and s[i - d2[i]
            - 1] == s[i + d2[i]]) {
            d2[i]++;
    }

        if(i + d1[i] - 1 > r1) {
            l1 = i - d1[i] + 1;
            r1 = i + d1[i] - 1;
        }

        if(i + d2[i] - 1 > r2) {
            l2 = i - d2[i];
            r2 = i + d2[i] - 1;
        }
    }
}


signed main(){
}
```

7.3 rabinKarp

```
#include <bits/stdc++.h>
#define int long long int
using namespace std;
const int p = 31;
const int m = 1e9 + 9;
int S,T;//s patron, t texto
//Rabin Karp O(n+m)
vector<int> rabin_karp(string const& s, string const& t) {
    S = s.size(), T = t.size();
    vector<int> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;
    vector<int> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;


    //for(int i:h)cout << i << " ";
    //cout << endl;

    int h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

    vector<int> ocurrencias;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            ocurrencias.push_back(i);
    }
    return ocurrencias;
}
signed main(){

    string s = "hola";
    string t= "holaaqueetal";
    auto v = rabin_karp(s,t);
    for(int i:v)cout << i << " ";
    cout << endl;
}
```

7.4 stringHashing

```
#include <bits/stdc++.h>
#define fore(i,b,n) for(int i = b; i < n; i++)
#define endl "\n"
#define MOD                 1000000007
#define INF                 2e18
#define fast_cin()          ios::sync_with_stdio(0); cin.tie(0);
    cout.tie(0);
const int B = 257;//259

using namespace std;
```

```
struct hashing{
    string s;
    int n, b,mod;
    vector<int> hash, base;
    hashing(){}
    void init(string &s1, int b1, int mod1){
        s = s1; n = s.size(); b = b1; mod = mod1;
        hash.resize(n); base.resize(n);
        build();
    }
    void build(){
        base[0] = 1;
        for(int i = 1; i < n; i++)
            base[i] = 1ll*base[i-1]*b % mod;
        int h = 0;
        for(int i = 0; i < n; i++){
            h = (1ll*h*b + s[i]) % mod;
            hash[i] = h;
        }
    }
    int stringH(){return hash[n-1];}
    int substringH(int l, int r){return (l==0)?hash[r]:
    (hash[r] - (1ll*hash[l-1]*base[r-l+1]) % mod + mod) % mod;}
};
int main(){
    string s = "ab";
    hashing h1;
    h1.init(s,257,MOD);
    cout << h1.stringH() << endl;
    return 0;
}
```