Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos

1.1 Mochila

```
int val[MAXN], peso[MAXN], dp[MAXN][MAXS]

int knapsack(int N, int M) // N Objetos | Peso max
{
    for(int i=0;i<=N;i++){
       for(int j=0;j<=M;j++){
            if(i==0 or j==0)
                dp[i][j] = 0;
            else if(peso[i-1]<=j)
                dp[i][j] = max(val[i-1]+dp[i-1][j-peso[i-1]], dp[i-1][j]);
            else
                dp[i][j] = dp[i-1][j];
       }
    }
    return dp[N][M];
}
```

1.2 Unbounded-Knapsack

```
int w, n;
int c[MAX], v[MAX];

int unbounded_knapsack()
{

    int dp[w+1];
    memset(dp, 0, sizeof dp);

    for(int i=0;i<=w;i++)
        for(int j=0;j<n;j++)
            if(c[j] <= i)
                dp[i] = max(dp[i], dp[i-c[j]] + v[j]);

    return dp[w];

}
```

1.3 Kadane-DP

```
// Largest Sum Contiguous Subarray

int maxSubArraySum(vector<int> a)
{
    int size = a.size();
    int max_so_far = a[0];
    int curr_max = a[0];

    for (int i=1;i<size;i++)
    {
        curr_max = max(a[i], curr_max+a[i]);
        max_so_far = max(max_so_far, curr_max);
    }
    return max_so_far;
}
```

1.4 Iterative-BS

```
int l=1, r=N;
int res=-1;

while(l<=r){
    int m = (l+r)/2;
    if(!ver(m)){
        l = m+1;
    }
    else{
        res = m;
        r = m-1;
    }
}
cout << res << endl;
```

2 Grafos

2.1 BFS

```
//BFS (Breadth First Search) O(V+A)

vector<vector<int>> adj;  // adjacency list representation
int n; // number of nodes
int s; // source vertex

queue<int> q;
vector<int> d(n, INF);
d[s]=0;

q.push(s);
used[s] = true;
while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (int u : adj[v]) {
        if (d[u] > d[v] + 1) {
            q.push(u);
            d[u] = d[v] + 1;
        }
    }
}
```

2.2 Find-bridges

```
int n;
vector<vi> adj(n+1, vi());

vector<bool> visited;
vi tin, low;
int timer;

void dfs(int v, int p=-1){
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to: adj[v]){
        if(to == p) continue;
        if(visited[to])
            low[v] = min(low[v], tin[to]);
        else{
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if(low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}

void find_bridges(){
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for(int i=0;i<n;i++)
        if(!visited[i])
            dfs(i);
}
```

2.3 Dijkstra

```cpp
// Dijkstra - Shortest Path

#define pii pair<int, int>
#define vi vector<int>
#define vii vector< pair<int,int> >
#define INF 0x3f3f3f3f

vector<vii> grafo(N+1, vii());
vi distancia(N+1, INF);
priority_queue< pii, vii, greater<pii> > fila;

void dijkstra(int k){
    int dist, vert, aux;
    distancia[k]=0;

    fila.push(mp(k, 0));

    while(!fila.empty()){
        aux=fila.top().f;
        fila.pop();

        for(auto v: grafo[aux]){
            vert=v.f;
            dist=v.s;
            if(distancia[vert]>distancia[aux]+dist){
                distancia[vert]=distancia[aux]+dist;
                fila.push(mp(vert, distancia[vert]));

            }
        }
    }
}

int main()
{
    for(int i=0; i<M; i++){
        cin >> a >> b >> p;
        grafo[a].pb(mp(b, p));
        grafo[b].pb(mp(a, p));
    }
}
```

2.4 LCA

```cpp
const int K = 100;
int logv[MAX+1];
int st[MAX][K];
vector<vi> grafo(200010, vi());

void make(){
    logv[1] = 0; // pre-computar tabela de log
    for (int i = 2; i <= MAX; i++)
        logv[i] = logv[i/2] + 1;
}

void precompute(int N, int array[]) { //
  for (int i = 0; i < N; i++)
      st[i][0] = array[i];

  int k = logv[N];
  for (int j = 1; j <= k; j++)
      for (int i = 0; i + (1 << j) <= N; i++)
          st[i][j] = min(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);
}

int query(int L, int R) {
    int j = logv[R - L + 1];
    int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);

    return minimum;
}


int start[MAX+1], dfs_time;
int tour[2*MAX+1], id[2*MAX+1];

void dfs(int u, int pai=-1){
    start[u] = dfs_time;
    id[dfs_time] = u;
    tour[dfs_time++] = start[u];
    for(int v : grafo[u]){
        if(v==pai)
            continue;
        dfs(v, u);
        id[dfs_time] = u;
        tour[dfs_time++] = start[u];
    }
}

int LCA(int u, int v)
{
    if(start[u] > start[v])
        swap(u, v);
    return id[query(start[u], start[v])];
}

int main()
{
    int N, k, a, b;
    cin >> N;

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

    make();
    precompute(2*N, tour);


    cin >> k;
    for(int i=0;i<k;i++)
    {
        cin >> a >> b;
        cout << LCA(a, b) << endl;
    }

    return 0;
}
```

2.5 Floyd-Warshall

```cpp
// Floyd Warshall

int dist[MAX][MAX];

void Floydwarshall()
{
    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]);
}
```

2.6 Kruskal

```cpp
// deve-se ter dsu codada com as funcoes make_set, find_set e union_sets
struct Edge {
    int u, v, weight;
```

```
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

int n;
vector<Edge> edges;

int cost = 0;
vector<Edge> result;
for (int i = 0; i < n; i++)
    make_set(i);

sort(edges.begin(), edges.end());

for (Edge e : edges) {
    if (find_set(e.u) != find_set(e.v)) {
        cost += e.weight;
        result.push_back(e); // vector com as arestas da MST
        union_sets(e.u, e.v);
    }
}
```

2.7 DFS

```
//DFS (Depth First Search) O(V+A)

void DFS(int x){
    for(int i=0; i<(int)vizinhos[x].size(); i++){
        int v = vizinhos[x][i];
        if(componente[v] == -1){
            componente[v] = componente[x];
            DFS(v);
        }
    }
}
```

2.8 Topological-sort

```
vector<vi> grafo(MAX, vi());
int grau[MAX]; // Quantas arestas chegam no indice i

vi topological_sort(int N)
{
    vi resp;
    for(int i=0;i<N;i++)
        if(!grau[i])
            resp.push_back(i);

    int k=0;
    while(k < (int)resp.size()){
        int u = resp[k];
        k++;
        for(auto v: grafo[u]){
            grau[v]--;
            if(!grau[v])
                resp.pb(v);
        }
    }
    return resp;
}
```

2.9 Kosaraju

```
// KOSARAJU - O(V+E) - encontra componentes fortemente conexos
// g -> grafo, gt -> grafo tempo
// vis -> visitado, cor -> componente fortemente conexo ordenado topologicamente
vector<int> g[N], gt[N], S; int vis[N], cor[N];
void dfs(int u){
    vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
    S.push_back(u);
}
void dfst(int u, int e){
    cor[u] = e;
    for(int v : gt[u]) if(!cor[v]) dfst(v, e);
}
void kosaraju(){
    for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);
    for(int i = 1; i <= n; i++) for(int j : g[i])
        gt[j].push_back(i);
    int e = 0; reverse(S.begin(), S.end());
    for(int u : S) if(!cor[u]) dfst(u, ++e);
}
```

2.10 Centroid

```
vi g[MAX];
int size[MAX];
bool erased[MAX]; // vetor dos vertices apagados na decomp.

int sz(int u, int p) {
  int s = 1;
  for(auto prox : g[u]) {
    if(prox != p and !erased[prox])
      s += sz(prox, u);
  }
  return size[u] = s;
}

int centroid(int u, int p, int n) {
  // chamar funcao sz antes, n = size[u]
  for(auto prox : g[u]) {
    if(prox != p and !erased[prox]) {
      if(size[prox] > n/2) {
        return centroid(prox, u, n);
      }
    }
  }
  return u;
}
```

2.11 Prim

```
// Prim Algorithm
#define MAXN 10100
#define INFINITO 999999999

int n, m;
int distancia[MAXN];
int processado[MAXN];
vector<pii> vizinhos[MAXN];

int Prim()
{
    for(int i = 2;i <= n;i++) distancia[i] = INFINITO;
    distancia[1] = 0;

    priority_queue< pii, vector<pii>, greater<pii> > fila;
    fila.push( pii(distancia[1], 1) );

    while(1){
        int davez = -1;

        while(!fila.empty()){
            int atual = fila.top().second;
            fila.pop();

            if(!processado[atual]){
```

```
                davez = atual;
                break;
            }
        }

        if(davez == -1)
            break;

        processado[davez] = true;

        for(int i = 0;i < (int)vizinhos[davez].size();i++){
            int dist  = vizinhos[davez][i].first;
            int atual = vizinhos[davez][i].second;

            if( distancia[atual] > dist && !processado[atual])
            {
                distancia[atual] = dist;
                fila.push( pii(distancia[atual], atual) );
            }
        }
    }

    int custo_arvore = 0;
    for(int i = 1;i <= n;i++)
        custo_arvore += distancia[i];

    return custo_arvore;
}

int main(){

    cin >> n >> m;

    for(int i = 1;i <= m;i++){

        int x, y, tempo;
        cin >> x >> y >> tempo;

        vizinhos[x].pb( pii(tempo, y) );
        vizinhos[y].pb( pii(tempo, x) );
    }

    cout << Prim() << endl;

    return 0;
}
```

3 Geometria

3.1 Inside-polygon

```
bool inside(vector<point> &vet, point ext) //ccw
{
    int l=2, r=(int)vet.size()-1;
    int res=r;
    while(l<r){
        int mid = (l+r)/2;
        if(ccw(vet[0], vet[mid], ext)==1)
            l=mid+1;
        else{
            r=mid;
            res=mid;
        }
    }
    int a = ccw(vet[0], vet[res-1], ext);
    int b = ccw(vet[res-1], vet[res], ext);
    int c = ccw(vet[res], vet[0], ext);

    if((a==1 or b==1 or c==1) and (a==-1 or b==-1 or c==-1)) return false;
    else return true;
}
```

3.2 MinDistPair

```
ll MinDistPair(vp &vet){
    int n = vet.size();
    sort(vet.begin(), vet.end());
    set<point> s;

    ll best_dist = LLINF;
    int j=0;
    for(int i=0;i<n;i++){
        int d = ceil(sqrt(best_dist));
        while(j<n and vet[i].x-vet[j].x >= d){
            s.erase(point(vet[j].y, vet[j].x));
            j++;
        }

        auto it1 = s.lower_bound({vet[i].y - d, vet[i].x});
        auto it2 = s.upper_bound({vet[i].y + d, vet[i].x});

        for(auto it=it1; it!=it2; it++){
            ll dx = vet[i].x - it->y;
            ll dy = vet[i].y - it->x;
            if(best_dist > dx*dx + dy*dy){
                best_dist = dx*dx + dy*dy;
                // vet[i] e inv(it)
            }
        }

        s.insert(point(vet[i].y, vet[i].x));
    }
    return best_dist;
}
```

3.3 Center-polygon

```
point center(vector<point> A){
    point centerA = point();
    for(int i=0;i<(int)A.size();i++)
        centerA=centerA+A[i];
    return centerA/(int)A.size();
}
```

3.4 Intersect-polygon

```
bool intersect(vector<point> A, vector<point> B) //Ordered ccw
{
    for(auto a: A)
        if(inside(B, a))
            return true;
    for(auto b: B)
        if(inside(A, b))
            return true;

    if(inside(B, center(A)))
        return true;

    return false;
}
```

3.5 Sort-by-Angle

```
int quarter(point a)
{
```

```
    if(a.x>=0 and a.y>0) return 0;
    if(a.x<0 and a.y>=0) return 1;
    if(a.x<=0 and a.y<0) return 2;
    return 3;
}

point c;
bool comp(point a, point b) //ccw
{
    a=a-c;b=b-c;
    int qa = quarter(a);
    int qb = quarter(b);
    if(qa==qb)
        return cross(a,b)>0;
    else
        return qa<qb;
}

c = center(A);
sort(A.begin(), A.end(), comp);
```

3.6 Convex-Hull

```
vp convex_hull(const vp points)
{
    vp P(points);
    sort(P.begin(), P.end());
    vp lower, upper;
    for(const auto& p: P){
        int n = int(lower.size());
        while(n>=2 and ccw(lower[n-2], lower[n-1], p)!=-1){
            lower.pop_back();
            n = int(lower.size());
        }
        lower.push_back(p);
    }
    reverse(P.begin(), P.end());
    for(const auto& p: P){
        int n = int(upper.size());
        while(n>=2 and ccw(upper[n-2], upper[n-1], p)!=-1){
            upper.pop_back();
            n = int(upper.size());
        }
        upper.push_back(p);
    }
    lower.pop_back();
    lower.insert(lower.end(), upper.begin(), upper.end()-1);
    return lower;
}
```

3.7 Inter-Retangulos

```
bool doOverlap(point l1, point r1, point l2, point r2)
{
    if (l1.x>r2.x or l2.x>r1.x or l1.y<r2.y or l2.y<r1.y)
        return false;
    return true;
}
```

3.8 Tetrahedron-Distance3D

```
bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0));
}

ld misto(point p1, point p2, point p3){
    return (p1^p2)*p3;
}

ld dist_pt_face(point p, vp v){
    assert(v.size()==3);

    point v1 = v[1]-v[0];
    point v2 = v[2]-v[0];
    point n = (v1^v2);

    for(int i=0;i<3;i++){
        point va = p-v[i];
        point vb = v[(i+1)%3]-v[i];
        point ve = vb^n;
        ld d = ve*v[i];
        //se ponto coplanar com um dos lados do prisma (va^vb eh nulo),
        //ele esta dentro do prisma (poderia desconsiderar pois distancia
        //vai ser a msm da distancia do ponto ao segmento)
        if(!nulo(va^vb) and (v[(i+2)%3]*ve>d) ^ (p*ve>d)) return LLINF;
    }

    //se ponto for coplanar ao triangulo (e dentro do triangulo)
    //vai retornar zero corretamente
    return fabs(misto(p-v[0],v1,v2)/norm(n));
}

ld dist_pt_seg(point p, vp li){
    return norm((li[1]-li[0])^(p-li[0]))/norm(li[1]-li[0]);
}

ld dist_line(vp l1, vp l2){
    point n = (l1[1]-l1[0])^(l2[1]-l2[0]);
    if(nulo(n)) //retas paralelas - dist ponto a reta
        return dist_pt_seg(l2[0],l1);

    point o1o2 = l2[0]-l1[0];
    return fabs((o1o2*n)/norm(n));
}
// retas paralelas e intersecao nao nula
ld dist_seg(vp l1, vp l2){

    assert(l2.size()==2);
    assert(l1.size()==2);

    //pontos extremos do segmento
    ld ans = LLINF;
    for(int i=0;i<2;i++)
        for(int j=0;j<2;j++)
            ans = min(ans, norm(l1[i]-l2[j]));

    //verificando distancia de ponto extremo com ponto interno dos segs
    for(int t=0;t<2;t++){
        for(int i=0;i<2;i++){
            bool c=true;
            for(int k=0;k<2;k++){
                point va = l1[i]-l2[k];
                point vb = l2[!k]-l2[k];
                ld ang = atan2(norm((vb^va)), vb*va);
                if(ang>PI/2) c = false;
            }
            if(c)
                ans = min(ans,dist_pt_seg(l1[i],l2));
        }
        swap(l1,l2);
    }

```

```
    //ponto interno com ponto interno dos segmentos
    point v1 = l1[1]-l1[0], v2 = l2[1]-l2[0];
    point n = v1^v2;
    if(!nulo(n)){
        bool ok = true;
        for(int t=0;t<2;t++){
            point n2 = v2^n;
            point o1o2 = l2[0]-l1[0];
            ld escalar = (o1o2*n2)/(v1*n2);
            if(escalar<0 or escalar>1) ok = false;
            swap(l1,l2);
            swap(v1,v2);
        }
        if(ok) ans = min(ans,dist_line(l1,l2));
    }

    return ans;
}

ld ver(vector<vp> &vet){
    ld ans = LLINF;
    // vertice - face
    for(int k=0;k<2;k++)
        for(int pt=0;pt<4;pt++)
            for(int i=0;i<4;i++){
                vp v;
                for(int j=0;j<4;j++){
                    if(i!=j) v.pb(vet[!k][j]);
                }
                ans = min(ans, dist_pt_face(vet[k][pt
], v));
            }

    // edge - edge
    for(int i1=0;i1<4;i1++)
        for(int j1=0;j1<i1;j1++)
            for(int i2=0;i2<4;i2++)
                for(int j2=0;j2<i2;j2++)
                    ans = min(ans, dist_seg({vet[0][
i1], vet[0][j1]},
                                            {vet[1][
i2], vet[1][j2]}));

    return ans;
}
```

3.9 3D

```
// typedef int cod;
// bool eq(cod a, cod b){ return (a==b); }

#define vp vector<point>
typedef ld cod;
bool eq(cod a, cod b){ return fabs(a - b) <= EPS; }

struct point
{
    cod x, y, z;
    point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z
    ){}

    point operator+(const point &o) const{
        return {x+o.x, y+o.y, z+o.z};
    }
    point operator-(const point &o) const{
        return {x-o.x, y-o.y, z-o.z};
    }
    point operator*(cod t) const{
        return {x*t, y*t, z*t};
    }
    point operator/(cod t) const{
        return {x/t, y/t, z/t};
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y) and eq(z, o.
    z);
    }
    cod operator*(const point &o) const{ // dot
        return x*o.x + y*o.y + z*o.z;
    }
    point operator^(const point &o) const{ // cross
        return point(y*o.z - z*o.y,
                     z*o.x - x*o.z,
                     x*o.y - y*o.x);
    }
};

ld dist(point a, point b){
    return sqrt((a-b)*(a-b));
}
bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0))
    ;
}

ld norm(point a){ // Modulo
    return sqrt(a*a);
}
ld proj(point a, point b){ // a sobre b
    return (a*b)/norm(b);
}
ld angle(point a, point b){ // em radianos
    return acos((a*b) / norm(a) / norm(b));
}

cod triple(point a, point b, point c){
    return dot(a, b^c); // Area do paralelepipedo
}


struct plane{
    point p1, p2, p3;
    plane(point p1=0, point p2=0, point p3=0): p1(p1)
    , p2(p2), p3(p3){}

    point aux = (p1-p3)^(p2-p3);
    cod a = aux.x, b = aux.y, c = aux.z;
    cod d = -a*p1.x - b*p1.y - c*p1.z;
    // ax+by+cz+d = 0;
};

cod dist(plane pl, point p){
    return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d
    ) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
}
```

3.10 Heron

```
ld heron(int a, int b, int c){
    ld s = (a+b+c)/2.0;
    return sqrtl(s*(s-a)*(s-b)*(s-c));
}

ld heron(int a, int b, int c, int d){
    ld s = (a+b+c+d)/2.0;
    return sqrtl((s-a)*(s-b)*(s-c)*(s-d));
}
```

3.11 Uniao-segmentos

```
int length_union(const vector<pii> &a){
    int n = a.size();
    vector<pair<int, bool>> x(n*2);
    for(int i = 0; i < n; i++){
        x[i*2] = {a[i].ff, false};
```

```
        x[i*2+1] = {a[i].ss, true};
    }

    sort(x.begin(), x.end());

    int result=0;
    int c=0;
    for(int i=0;i<2*n;i++){
        if(i and c and x[i].ff>x[i-1].ff)
            result += x[i].ff-x[i-1].ff;

        if(x[i].ss) c--;
        else c++;
    }
    return result;
}
```

3.12 Minkowski-Sum

```
vp mk(const vp &a,const vp &b){
    int i = 0, j = 0;
    for(int k = 0; k < (int)a.size(); k++)if(a[k]<a[i
    ])
        i = k;
    for(int k = 0; k < (int)b.size(); k++)if(b[k]<b[j
    ])
        j = k;

    vp c;
    c.reserve(a.size() + b.size());
    for(int k = 0; k < int(a.size()+b.size()); k++){
        point pt{a[i] + b[j]};
        if((int)c.size() >= 2 and !ccw(c[c.size()-2],
     c.back(), pt))
            c.pop_back();
        c.pb(pt);
        int q = i+1, w = j+1;
        if(q == int(a.size())) q = 0;
        if(w == int(b.size())) w = 0;
        if(ccw(c.back(), a[i]+b[w], a[q]+b[j]) < 0) i
     = q;
        else j = w;
    }

    if(!ccw(c[0], c[(int)c.size()-1], c[(int)c.size()
    -2]))
        c.pop_back();
    if(!ccw(c.back(), c[0], c[1])){
        c[0]=c.back();
        c.pop_back();
    }
    c.shrink_to_fit();

    return c;
}
```

3.13 Simetria-central

```
bool simetric(vector<point> &a){ //ordered
    int n = a.size();
    c = center(a);
    if(n&1) return false;
    for(int i=0;i<n/2;i++)
        if(!collinear(a[i], a[i+n/2], c))
            return false;
    return true;
}
```

3.14 2D

```
// typedef int cod;
// bool eq(cod a, cod b){ return (a==b); }

#define vp vector<point>
typedef ld cod;
bool eq(cod a, cod b){ return fabsl(a - b) <= EPS; }

struct point{
    cod x, y;
    int id;
    point(cod x=0, cod y=0): x(x), y(y){}


    point operator+(const point &o) const{
        return {x+o.x, y+o.y};
    }
    point operator-(const point &o) const{
        return {x-o.x, y-o.y};
    }
    point operator*(cod t) const{
        return {x*t, y*t};
    }
    point operator/(cod t) const{
        return {x/t, y/t};
    }
    cod operator*(const point &o) const{ // dot
        return x * o.x + y * o.y;
    }
    cod operator^(const point &o) const{ // cross
        return x * o.y - y * o.x;
    }
    bool operator<(const point &o) const{
        if(!eq(x, o.x)) return x < o.x;
        return y < o.y;
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y);
    }

};

bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0));
}
ld norm(point a){ // Modulo
    return sqrt(a*a);
}
ld proj(point a, point b){ // a sobre b
    return a*b/norm(b);
}
ld max(ld a, ld b){ return (a>b ? a:b); }
ld min(ld a, ld b){ return (a<b ? a:b); }
ld angle(point a, point b){ // em radianos
    ld ang = a*b / norm(a) / norm(b);
    return acos(max(min(ang, 1), -1));
}
ld angle_vec(point v){
    // return 180/PI*atan2(v.x, v.y);
    return atan2(v.x, v.y);
}
int ccw(point &a, point &b, point &e){ //-1=dir; 0=
    collinear; 1=esq;
    cod tmp = (b-a)^(e-a); // from a to b
    return (tmp > EPS) - (tmp < -EPS);
}
ld order_angle(point a, point b){ // from a to b ccw
    (a in front of b)
    ld aux = angle(a,b)*180/PI;
    return ((a^b)<=0 ? aux:360-aux);
}

bool collinear(point a, point b, point c){
    return eq((a-c)^(b-c), 0);
}

```

```
point rotccw(point p, ld a){
    // a = PI*a/180; // graus
    return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
    +p.x*sin(a)));
}

point rot90cw(point a) { return point(a.y, -a.x); };
point rot90ccw(point a) { return point(-a.y, a.x); };

// Area de um poligono (pontos ordenados por
    adjacencia)
ld area(vp &p){
    ld ret = 0;
    for(int i=2;i<(int)p.size();i++)
        ret += (p[i]-p[0])^(p[i-1]-p[0]);
    return fabsl(ret/2);
}




struct line{
    point p1, p2;
    line(point p1=0, point p2=0): p1(p1), p2(p2){}

    cod a = p1.y-p2.y;
    cod b = p2.x-p1.x;
    cod c = -(a*p1.x + b*p1.y);
    // ax+by+c = 0;

    bool inside(point p){
        return eq(norm(p1-p)+norm(p2-p) - norm(p2-p1)
    , 0);
    }
};

point intersection(line l1, line l2){
    ld det = l1.a*l2.b - l1.b*l2.a;
    if(det==0) return point(INF, INF);
    ld x = (l1.b*l2.c - l1.c*l2.b)/det;
    ld y = (l1.c*l2.a - l1.a*l2.c)/det;
    return point(x, y);
}

// Dist entre ponto e segmento de reta
cod distr(point p, point a, point b){
    if(((b-a)^(p-b)) > 0)
        return norm(p-b);
    if(((a-b)^(p-a)) > 0)
        return norm(p-a);
    return fabs((b-a)^(p-a))/norm(a-b);
}

struct circle{
    point c;
    cod r;
    circle(point c=0, cod r=0): c(c), r(r){}
};
```

4 ED

4.1 Trie

```
class Trie {
private:
   struct Node {
      map<char, Node*> children;
      int qt = 0;
      ll size = 0;
   };

   Node* root;

   void dfs(Node* cur) {
     ll sz = 1;

     for(auto prox : cur->children) {
         dfs(prox.second);
         sz += (prox.second)->size;
     }

     cur->size = sz;
   }

    void del(Node* cur, int dep, string &s) {
        if(dep >= 32)
            return;

        Node* prox = cur->children[s[dep]];
        prox->qt--;
        del(prox, dep+1, s);

        if(prox->qt == 0)
           cur->children.erase(s[dep]);
    }

public:
   Trie() {
      root = new Node();
      root->qt = 1;
   }

   void add(string s) {
      Node* cur = root;

      for(auto c : s) {
         if(cur->children.count(c) == 0) {
            cur->children[c] = new Node();
         }
         cur->children[c]->qt++;
         cur = cur->children[c];
      }
   }

   void del(string &s) {
       Node* cur = root;
       del(cur, 0, s);
   }

   void size() {
       this->dfs(root);
   }
};
```

4.2 Range-query-bigger-than-k-BIT

```
// C++ program to print the number of elements
// greater than k in a subarray of range L-R.
#include <bits/stdc++.h>
using namespace std;

// Structure which will store both
// array elements and queries.
struct node{
    int pos;
    int l;
    int r;
    int val;
};

// Boolean comparator that will be used
// for sorting the structural array.
bool comp(node a, node b){
    // If 2 values are equal the query will
    // occur first then array element
```

```
    if (a.val == b.val)
        return a.l > b.l;

    // Otherwise sorted in descending order.
    return a.val > b.val;
}

// Updates the node of BIT array by adding
// 1 to it and its ancestors.
void update(int* BIT, int n, int idx){
    while (idx <= n){
        BIT[idx]++;
        idx += idx & (-idx);
    }
}
// Returns the count of numbers of elements
// present from starting till idx.
int query(int* BIT, int idx){
    int ans = 0;
    while (idx){
        ans += BIT[idx];

        idx -= idx & (-idx);
    }
    return ans;
}

// Function to solve the queries offline
void solveQuery(int arr[], int n, int QueryL[],
                int QueryR[], int QueryK[], int q){
    // create node to store the elements
    // and the queries
    node a[n + q + 1];
    // 1-based indexing.

    // traverse for all array numbers
    for(int i = 1; i <= n; ++i){
        a[i].val = arr[i - 1];
        a[i].pos = 0;
        a[i].l = 0;
        a[i].r = i;
    }

    // iterate for all queries
    for(int i = n + 1; i <= n + q; ++i){
        a[i].pos = i - n;
        a[i].val = QueryK[i - n - 1];
        a[i].l = QueryL[i - n - 1];
        a[i].r = QueryR[i - n - 1];
    }

    // In-built sort function used to
    // sort node array using comp function.
    sort(a + 1, a + n + q + 1, comp);

    // Binary Indexed tree with
    // initially 0 at all places.
    int BIT[n + 1];

    // initially 0
    memset(BIT, 0, sizeof(BIT));

    // For storing answers for each query( 1-based
    indexing ).
    int ans[q + 1];

    // traverse for numbers and query
    for (int i = 1; i <= n + q; ++i){
        if (a[i].pos != 0) {

            // call function to returns answer for
    each query
            int cnt = query(BIT, a[i].r) - query(BIT,
     a[i].l - 1);

            // This will ensure that answer of each
    query
            // are stored in order it was initially
    asked.
            ans[a[i].pos] = cnt;
        }
        else{
            // a[i].r contains the position of the
            // element in the original array.
            update(BIT, n, a[i].r);
        }
    }
    // Output the answer array
    for (int i = 1; i <= q; ++i){
        cout << ans[i] << endl;
    }
}

// Driver Code
int main()
{
    int arr[] = { 7, 3, 9, 13, 5, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // 1-based indexing
    int QueryL[] = { 1, 2 };
    int QueryR[] = { 4, 6 };

    // k for each query
    int QueryK[] = { 6, 8 };

    // number of queries
    int q = sizeof(QueryL) / sizeof(QueryL[0]);

    // Function call to get
    solveQuery(arr, n, QueryL, QueryR, QueryK, q);

    return 0;
}
```

4.3 Prefixsum2D

```
ll find_sum(vector<vi> &mat, int a, int b, int c, int
     d){
    // superior-esq(c,d) (a,b)inferior-dir
    return mat[a][b]-mat[a][d-1]-mat[c-1][b]+mat[c
    -1][d-1];
}

int main(){

    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            mat[i][j]+=mat[i-1][j]+mat[i][j-1]-mat[i
    -1][j-1];

}
```

4.4 Iterative-SegTree

```
// Segment Tree Iterativa - Max

struct Segtree{
    vi t;
    int n;

    Segtree(int n){
        this->n = n;
        t.assign(2*n, 0);
    }
```

```
    void build(){
        for(int i=n-1; i>0; i--)
            t[i]=max(t[i<<1], t[i<<1|1]);
    }

    int query(int l, int r){ // idx 0
        int ans=0;
        for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
            if(l&1)
                ans=max(ans, t[l++]);
            if(r&1)
                ans=max(t[--r], ans);
        }
        return ans;
    }

    void update(int p, int value){
        for(t[p+=n]=value; p >>= 1;)
            t[p] = max(t[p<<1], t[p<<1|1]);
    }

};

int main()
{
    Segtree st(n);

    for(int i=0;i<n;i++){
        cin >> aux;
        st.t[n+i]=aux; //Leaves are stored in
    continuous nodes with indices starting with N
    }

    st.build();
    x = st.query(inicio, fim);
    st.update(ind, value);

}
```

4.5 Recursive-SegTree

```
// Segment Tree Recursiva - Range maximum query

int val[4*MAX];
int vet[MAX];

void monta(int i, int j, int no){
    if(i==j){
        val[no]=vet[i];
        return;
    }

    int esq = 2*no;
    int dir = 2*no+1;
    int meio = (i+j)/2;

    monta(i, meio, esq);
    monta(meio+1, j, dir);

    val[no]=max(val[esq], val[dir]);
}

void atualiza(int no, int i, int j, int pos, int
    novo_valor){
    if(i==j){
        val[no]=novo_valor;
    }else{
        int esq = 2*no;
        int dir = 2*no+1;
        int meio = (i+j)/2;

        if(pos<=meio)
            atualiza(esq, i, meio, pos, novo_valor);
        else
            atualiza(dir, meio+1, j, pos, novo_valor)
    ;

        if(val[esq]>val[dir])
            val[no]=val[esq];
        else
            val[no]=val[dir];
    }
}

int consulta(int no, int i, int j, int A, int B){
    if(i>B || j<A)
        return -1;
    if(i>=A and j<=B)
        return val[no];

    int esq = 2*no;
    int dir = 2*no+1;
    int meio = (i+j)/2;

    int resp_esq = consulta(esq, i, meio, A, B);
    int resp_dir = consulta(dir, meio+1, j, A, B);

    if(resp_dir==-1)
        return resp_esq;
    if(resp_esq==-1)
        return resp_dir;

    if(resp_esq>resp_dir)
        return resp_esq;
    else
        return resp_dir;
}

int main()
{
    monta(1, N, 1);
    atualiza(1, 1, N, pos, valor);
    x = consulta(1, 1, N, inicio, fim);
}
```

4.6 Seg-Tree-MaxSubArray

```
// Subarray with maximum sum
struct no{
    ll pref, suff, total, best;
    no(ll x=0): pref(x), suff(x), total(x), best(x){}
};

no merge(no l, no r){

    no ans;
    ans.pref = max(0LL, max(l.pref, l.total+r.pref));
    ans.suff = max(0LL, max(r.suff, l.suff+r.total));
    ans.total = l.total+r.total;
    ans.best = max(max(l.best, r.best), l.suff+r.pref
    );
    return ans;
}

struct Segtree{
    vector<no> t;
    int n;

    Segtree(int n){
        this->n = n;
        t.assign(2*n, no(0));
    }

    void build(){
        for(int i=n-1; i>0; i--)
```

```
            t[i]=merge(t[i<<1], t[i<<1|1]);
    }

    no query(int l, int r){ // idx 0
        no a(0), b(0);
        for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
            if(l&1)
                a=merge(a, t[l++]);
            if(r&1)
                b=merge(t[--r], b);
        }
        return merge(a, b);
    }

    void update(int p, int value){
        for(t[p+=n] = no(value); p >>= 1;)
            t[p] = merge(t[p<<1], t[p<<1|1]);
    }

};
```

4.7 Delta-Encoding

```
// Delta encoding

for(int i=0;i<q;i++){
    int l,r,x;
    cin >> l >> r >> x;
    delta[l] += x;
    delta[r+1] -= x;
}

int atual = 0;

for(int i=0;i<n;i++){
    atual += delta[i];
    v[i] += atual;
}
```

4.8 BIT-2D

```
// BIT 2D

int bit[MAX][MAX];

int sum(int x, int y)
{
    int resp=0;

    for(int i=x;i>0;i-=i&-i)
        for(int j=y;j>0;j-=j&-j)
            resp+=bit[i][j];

    return resp;
}

void update(int x, int y, int delta)
{
    for(int i=x;i<MAX;i+=i&-i)
        for(int j=y;j<MAX;j+=j&-j)
            bit[i][j]+=delta;
}

int query(int x1, y1, x2, y2)
{
    return sum(x2,y2) - sum(x2,y1) - sum(x1,y2) + sum
    (x1,y1);
}
```

4.9 BIT

```
struct FT {
    vector<int> bit;  // indexado em 1
    int n;

    FT(int n) {
        this->n = n + 1;
        bit.assign(n + 1, 0);
    }

    int sum(int idx) {
        int ret = 0;
        for (++idx; idx > 0; idx -= idx & -idx)
            ret += bit[idx];
        return ret;
    }

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

    void add(int idx, int delta) {
        for (++idx; idx <= n; idx += idx & -idx)
            bit[idx] += delta;
    }
};
```

4.10 BIT-kth

```
struct FT {
    vector<int> bit;  // indexado em 1
    int n;

    FT(int n) {
        this->n = n + 1;
        bit.assign(n + 1, 0);
    }

    int kth(int x){
        int resp = 0;
        x--;
        for(int i=26;i>=0;i--){
            if(resp + (1<<i) >= n) continue;
            if(bit[resp + (1<<i)] <= x){
                x -= bit[resp + (1<<i)];
                resp += (1<<i);
            }
        }
        return resp + 1;
    }

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

4.11 DQUERY-Seg-Persistente

```
// distinct elements in the range [l, r]
struct node{
    int val;
    int l, r;
    node(int a=-1, int b=-1, int c=0){
        val=c;l=a;r=b;
    }
};

node tree[8600010]; // nlog(n) space = 8600010
int idx=0;

int build(int l, int r){
    if(l==r)
        return idx++;
```

```
    int mid = (l+r)/2;

    tree[idx].l = build(l, mid);
    tree[idx].r = build(mid+1, r);

    return idx++;
}

int update(int l, int r, int root, int e, int o){
    int plus=0;
    if((l>e or r<e) and (l>o or r<o or o==-1))
        return root;
    if(l<=e and e<=r) plus++;
    if(l<=o and o<=r and o!=-1) plus--;

    if(l==e and r==e){
        tree[idx]=node(-1, -1, 1);
        return idx++;
    }
    if(l==o and r==o){
        tree[idx]=node(-1, -1, 0);
        return idx++;
    }
    int mid = (l+r)/2;
    tree[idx]=node(update(l, mid, tree[root].l, e, o),
                   update(mid+1, r, tree[root].r, e, o), tree[root].val+plus);
    return idx++;
}

int query(int a, int b, int l, int r, int root){
    if(l>b or r<a)
        return 0;
    if(l<=a and b<=r)
        return tree[root].val;
    int mid = (a+b)/2;

    return query(a, mid, l, r, tree[root].l) + query(mid+1, b, l, r, tree[root].r);
}


int main()
{sws;

    int n, m, a, b;
    int v[MAX], aux[MAX];
    int root[MAX];

    cin >> n >> m;

    for(int i=0;i<n;i++){
        cin >> v[i]; aux[i]=v[i];
    }

    sort(v, v+n);

    map<int, int> comp;
    for(int i=0, j=0;i<n;i++)
        if(i==0 or v[i]!=v[i-1])
            comp[v[i]]=j++;

    root[0]=build(0, n-1);

    int last[MAX];
    memset(last, -1, sizeof(last));

    for(int i=0;i<n;i++){
        root[i+1] = update(0, n-1, root[i], i, last[comp[aux[i]]]);
        last[comp[aux[i]]]=i;
    }

    for(int i=0;i<m;i++){
        cin >> a >> b;
        cout << query(0, n-1, a-1, n-1, root[b]) << endl;
    }

    return 0;
}
```

4.12 Sparse-Table

```
int logv[MAX+1];
void make_log() {
    logv[1] = 0; // pre-computar tabela de log
    for (int i = 2; i <= MAX; i++)
        logv[i] = logv[i/2] + 1;
}

struct Sparse {
    int n;
    vector<vector<int>> st;

    Sparse(int n, vi array) {
        this->n = n;
        int k = logv[n];
        st.assign(n+1, vector<int>(k+1, 0));

        for (int i = 0; i < n; i++)
            st[i][0] = array[i];

        for (int j = 1; j <= k; j++)
            for (int i = 0; i + (1 << j) <= n; i++)
                st[i][j] = f(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);
    }

    int f(int a, int b) {
        return min(a, b);
    }

    int query(int L, int R) {
        int j = logv[R - L + 1];
        int res = f(st[L][j], st[R - (1 << j) + 1][j]);
        return res;
    }
};
```

4.13 Union-Find

```
struct DSU {
    int n;
    vi parent, size;

    DSU(int n) {
        this->n = n;
        parent.assign(n+1, 0);
        size.assign(n+1, 1);

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

    int find(int v) {
        if(v==parent[v])
            return v;
        return parent[v]=find(parent[v]);
    }

    void join(int a, int b) {
```

```
        a = find(a);
        b = find(b);
        if(a!=b) {
            if(size[a]<size[b])
                swap(a, b);

            parent[b]=a;
            size[a]+=size[b];
        }
    }
};
```

4.14 CHT

```
const ll is_query = -LLINF;
struct Line{
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const{
        if(rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if(!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct Cht : public multiset<Line>{ // maintain max
    bool bad(iterator y){
        auto z = next(y);
        if(y == begin()){
            if(z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if(z == end()) return y->m == x->m && y->b <= x->b;
        return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b){
        auto y = insert({ m, b });
        y->succ = [=]{ return next(y) == end() ? 0 : &*next(y); };
        if(bad(y)){ erase(y); return; }
        while(next(y) != end() && bad(next(y))) erase(next(y));
        while(y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x){
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

4.15 MKTHNUM-Seg-Persistente

```
// kth number in range [l, r] if it was ordered
struct node{
    int val;
    int l, r;
    node(int a=-1, int b=-1, int c=0){
        val=c;l=a;r=b;
    }
};

node tree[8600010]; // 4*nlog(4*n) space = 8600010
int idx=0;

int build(int l, int r){
    if(l==r)
        return idx++;

    int mid = (l+r)/2;

    tree[idx].l = build(l, mid);
    tree[idx].r = build(mid+1, r);

    return idx++;
}

int update(int l, int r, int root, int e){
    if(l>e or r<e)
        return root;
    if(l==e and r==e){
        tree[idx]=node(-1, -1, tree[root].val+1);
        return idx++;
    }
    int mid = (l+r)/2;
    tree[idx]=node(update(l, mid, tree[root].l, e),
                   update(mid+1, r, tree[root].r, e),
     tree[root].val+1);
    return idx++;
}

int query(int l, int r, int root1, int root2, int k){
    while(l!=r)
    {
        int mid=(l+r)/2;
        if(tree[tree[root2].l].val-tree[tree[root1].l].val>=k)
        {
            r = mid;
            root1 = tree[root1].l;
            root2 = tree[root2].l;
        }else
        {
            l = mid+1;
            k-=tree[tree[root2].l].val-tree[tree[root1].l].val;
            root1 = tree[root1].r;
            root2 = tree[root2].r;
        }
    }
    return l;
}


int main()
{sws;

    int n, m, a, b, k;
    int v[MAX], aux[MAX];
    int root[MAX];

    cin >> n >> m;

    for(int i=0;i<n;i++){
        cin >> v[i]; aux[i]=v[i];
    }

    sort(v, v+n);

    map<int, int> comp;
    for(int i=0, j=0;i<n;i++)
        if(i==0 or v[i]!=v[i-1])
            comp[v[i]]=j++;

    root[0]=build(0, n-1);

    for(int i=1;i<=n;i++)
        root[i] = update(0, n-1, root[i-1], comp[aux[i-1]]);

```

```
    for(int i=0;i<m;i++){
        cin >> a >> b >> k;
        cout << v[query(0, n-1, root[a-1], root[b], k
    )] << endl;
    }

    return 0;
}
```

4.16 Mo

```
const int BLK = 500; // tamanho do bloco, algo entre
    300 e 500 e nice

struct Query {
    int l, r, idx;
    Query(int l, int r, int idx) {
        this->l = l;
        this->r = r;
        this->idx = idx;
    }
    bool operator<(Query other) const {
        return make_pair(l / BLK, r) <
        make_pair(other.l / BLK, other.r);
    }
};

void add() void remove() // implementar operacoes de
    acordo com o problema, cuidado com TLE ao
    utilizar MAP

vector<int> mo(vector<Query> queries) {
    vector<int> res(queries.size());
    sort(queries.begin(), queries.end());
    resposta = 0;

    int l = 0, r = -1;
    for(Query q : queries) {
        while(l > q.l) {
                l--;
                add(l);
        }
        while(r < q.r) {
                r++;
                add(r);
        }
        while(l < q.l) {
                remove(l);
                l++;
        }
        while(r > q.r) {
                remove(r);
                r--;
        }
        res[q.idx] = resposta; // adicionar resposta
    de acordo com o problema
     }
     return res; // ordernar o vetor pelo indice e
    responder queries na ordem
}
```

5 Math

5.1 Totient

```
// phi(p^k) = (p^(k-1))*(p-1) com p primo
// O(sqrt(m))
ll phi(ll m){
    ll res = m;
    for(ll d=2;d*d<=m;d++){
        if(m % d == 0){
            res = (res/d)*(d-1);
            while(m%d == 0)
                m /= d;
        }
    }
    if(m > 1) {
        res /= m;
        res *= (m-1);
    }
    return res;
}

// modificacao do crivo, O(n*log(log(n)))
vl phi_to_n(ll n){
    vector<bool> isprime(n+1, true);
    vl tot(n+1);
    tot[0] = 0; tot[1] = 1;
    for(ll i=1;i<=n; i++){
        tot[i] = i;
    }

    for(ll p=2;p<=n;p++){
        if(isprime[p]){
            tot[p] = p-1;
            for(ll i=p+p;i<=n;i+=p){
                isprime[i] = false;
                tot[i] = (tot[i]/p)*(p-1);
            }
        }
    }
    return tot;
}
```

5.2 Sqrt-BigInt

```
public static BigInteger isqrtNewton(BigInteger n) {
    BigInteger a = BigInteger.ONE.shiftLeft(n.
    bitLength() / 2);
    boolean p_dec = false;
    for (;;) {
        BigInteger b = n.divide(a).add(a).shiftRight
    (1);
        if (a.compareTo(b) == 0 || a.compareTo(b) < 0
     && p_dec)
            break;
        p_dec = a.compareTo(b) > 0;
        a = b;
    }
    return a;
}
```

5.3 Linear-Diophantine-Equation

```
// Linear Diophantine Equation
int gcd(int a, int b, int &x, int &y)
{
    if (a == 0)
    {
        x = 0; y = 1;
        return b;
    }
    int x1, y1;
    int d = gcd(b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0,
    int &y0, int &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;
}

//  All solutions
//  x = x0 + k*b/g
//  y = y0 - k*a/g
```

5.4 Factorization-sqrt

```
// Factorization of a number in sqrt(n)

vi fact(ll n){
    vector<int> div;
    for(ll i=2;i*i<=n;i++)
        if(n%i==0){
            vet.pb(i);
            while(n%i==0)
                n/=i;
        }
    if(n!=1) vet.pb(n);
    return div;
}
```

5.5 FFT-simple

```
struct num{
    ld a {0.0}, b {0.0};
    num(){}
    num(ld na) : a{na}{}
    num(ld na, ld nb) : a{na}, b{nb} {}
    const num operator+(const num &c) const{
        return num(a + c.a, b + c.b);
    }
    const num operator-(const num &c) const{
        return num(a - c.a, b - c.b);
    }
    const num operator*(const num &c) const{
        return num(a*c.a - b*c.b, a*c.b + b*c.a);
    }
    const num operator/(const int &c) const{
         return num(a/c, b/c);
    }
};

void fft(vector<num> &a, bool invert){
    int n = a.size();
    for(int i=1,j=0;i<n;i++){
        int bit = n>>1;
        for(; j&bit; bit>>=1)
            j^=bit;
        j^=bit;
        if(i<j)
            swap(a[i], a[j]);
    }
    for(int len = 2; len <= n; len <<= 1){
        ld ang = 2 * PI / len * (invert ? -1 : 1);
        num wlen(cos(ang), sin(ang));
        for(int i=0;i<n;i+=len){
            num w(1);
            for (int j=0;j<len/2;j++){
                num u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w = w * wlen;
            }
        }
    }
    if(invert)
      for(num &x: a)
          x = x/n;

}

vl multiply(vi const& a, vi const& b){
    vector<num> fa(a.begin(), a.end());
    vector<num> fb(b.begin(), b.end());
    int n = 1;
    while(n < int(a.size() + b.size()) )
        n <<= 1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for(int i=0;i<n;i++)
        fa[i] = fa[i]*fb[i];
    fft(fa, true);
    vl result(n);
    for(int i=0;i<n;i++)
        result[i] = round(fa[i].a);
    while(result.back()==0) result.pop_back();
    return result;
}
```

5.6 Modular-Exponentiation

```
ll fexp(ll b, ll e, ll mod) {
    if(e == 0) return 1LL;
    ll res = fexp(b, e/2LL, mod);
    res = (res*res)%mod;
    if(e%2LL)
        res = (res*b)%mod;

    return res%mod;
}
```

5.7 Lagrange-interpolation

```
// Lagrange's interpolation (n+1 points)
ld interpolate(vii d, ld x){
    ld y = 0;
    int n = d.size();
    for(int i=0;i<n;i++){
        ld yi = d[i].ss;
        for(int j=0;j<n;j++)
            if(j!=i)
                yi = yi*(x - d[j].ff)/(ld)(d[i].ff - d
    [j].ff);

        y += yi;
    }
    return y;
}

ld inv_interpolate(vii d, ld y){
    ld x = 0;
    int n = d.size();
    for(int i=0;i<n;i++){
        ld xi = d[i].ff;
        for(int j=0;j<n;j++)
            if(j!=i)
                xi = xi*(y - d[j].ss)/(ld)(d[i].ss -
    d[j].ss);

        x += xi;
    }
    return x;
}
```

5.8 Miller-Habin

```
ll llrand()
{
    ll tmp = rand();
    return (tmp << 31) | rand();
}

ll add(ll a, ll b, ll c)
{
    return (a + b)%c;
}

ll mul(ll a, ll b, ll c)
{
    ll ans = 0;
    while(b)
    {
        if(b & 1)
            ans = add(ans, a, c);
        a = add(a, a, c);
        b /= 2;
    }
    return ans;
}

ll fexp(ll a, ll b, ll c)
{
    ll ans = 1;
    while(b)
    {
        if(b & 1)
            ans = mul(ans, a, c);
        a = mul(a, a, c);
        b /= 2;
    }
    return ans;
}

bool rabin(ll n)
{
    if(n <= 1)
        return 1;
    if(n <= 3)
        return 1;

    ll s=0, d=n-1;
    while(d%2==0)
    {
        d/=2;
        s++;
    }

    for(int k = 0; k < 64*4; k++)
    {
        ll a = (llrand()%(n - 3)) + 2;
        ll x = fexp(a, d, n);
        if(x != 1 and x != n-1)
        {
            for(int r = 1; r < s; r++)
            {
                x = mul(x, x, n);
                if(x == 1)
                    return 0;
                if(x == n-1)
                    break;
            }
            if(x != n-1)
                return 0;
        }
    }

    return 1;
}


int main()
{

    ll N;
    cin >> N;

    cout << rabin(N) << endl;

    return 0;

}
```

5.9 Inverso-Mult

```
// gcd(a, m) = 1 para existir solucao
// ax + my = 1, ou a*x = 1 (mod m)
ll inv(ll a, ll m) { // com gcd
    ll x, y;
    gcd(a, m, x, y);
    return (((x % m) +m) %m);
}

ll inv(ll a, ll phim) { // com phi(m), se m for primo
     entao phi(m) = p-1
    ll e = phim-1;
    return fexp(a, e);
}
```

5.10 Pollard-Rho

```
// Pollard Rho Algorithm

#include <bits/stdc++.h>
#define ll long long

using namespace std;

ll llrand()
{
    ll tmp = rand();
    return (tmp << 31) | rand();
}

ll add(ll a, ll b, ll c)
{
    return (a + b)%c;
}

ll mul(ll a, ll b, ll c)
{
    ll ans = 0;
    while(b)
    {
        if(b & 1)
            ans = add(ans, a, c);
        a = add(a, a, c);
        b /= 2;
    }
    return ans;
}

ll rho(ll n)
{
    ll x, c, y, d, k;
    int i;
    do{
        i = 1;
        x = llrand()%n;
        c = llrand()%n;
```

```
        y = x, k = 4;
        do{
            if(++i == k)
            {
                y = x;
                k *= 2;
            }
            x = add(mul(x, x, n), c, n);
            d = __gcd(abs(x - y), n);
        }
        while(d == 1);
    }
    while(d == n);

    return d;
}

int main()
{
    srand(time(0));

    ll N;
    cin >> N;

    ll div = rho(N);
    cout << div << " " << N/div << endl;


    // Finding all divisors

    vector<ll> div;

    while(N>1 and !rabin(N))
    {
        ll d = rho(N);
        div.pb(d);
        while(N%d==0)
            N/=d;
    }
    if(N!=1)
        div.pb(N);

    return 0;

}
```

5.11 Verif-primo

```
// prime verification sqrt(N)

bool eh_primo(long long N)
{
    if(N==2)
        return true;
    else if(N==1 or N%2==0)
        return false;
    for(long long i=3;i*i<=N;i+=2)
        if(N%i==0)
            return false;
    return true;
}
```

5.12 Crivo

```
// Sieve of Eratosthenes

int N;
vector<bool> primos(100010, true);
cin >> N;

primos[0]=false;
primos[1]=false;

for(int i=2;i<=N;i++)
    if(primos[i])
        for(int j=i+i; j<=N; j+=i)
            primos[j]=false;
```

5.13 Bigmod

```
ll mod(string a, ll p) {
    ll res = 0, b = 1;
    reverse(all(a));

    for(auto c : a) {
        ll tmp = (((ll)c-'0')*b) % p;
        res = (res + tmp) % p;

        b = (b * 10) % p;
    }

    return res;
}
```

5.14 Simpson's-formula

```
inline ld simpson(ld fl, ld fr, ld fmid, ld l, ld r){
    return (fl+fr+4*fmid)*(r-l)/6;
}

ld rsimpson(ld slr, ld fl, ld fr, ld fmid, ld l, ld r
    )
{
    ld mid = (l+r)/2;
    ld fml = f((l+mid)/2), fmr = f((mid+r)/2);
    ld slm = simpson(fl,fmid,fml,l,mid);
    ld smr = simpson(fmid,fr,fmr,mid,r);
    if(fabsl(slr-slm-smr) < EPS) return slm+smr; //
    aprox. good enough
    return rsimpson(slm,fl,fmid,fml,l,mid)+rsimpson(
    smr,fmid,fr,fmr,mid,r);
}

ld integrate(ld l, ld r)
{
    ld mid = (l+r)/2;
    ld fl = f(l), fr = f(r);
    ld fmid = f(mid);
    return rsimpson(simpson(fl,fr,fmid,l,r),fl,fr,
    fmid,l,r);
}
```

5.15 FFT-tourist

```
struct num{
    ld x, y;
    num() { x = y = 0; }
    num(ld x, ld y) : x(x), y(y) {}
};

inline num operator+(num a, num b) { return num(a.x +
     b.x, a.y + b.y); }
inline num operator-(num a, num b) { return num(a.x -
     b.x, a.y - b.y); }
inline num operator*(num a, num b) { return num(a.x *
     b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
inline num conj(num a) { return num(a.x, -a.y); }

int base = 1;
vector<num> roots = {{0, 0}, {1, 0}};
vi rev = {0, 1};

void ensure_base(int nbase){
    if(nbase <= base)
```

```
        return;

    rev.resize(1 << nbase);
    for(int i = 0; i < (1 << nbase); i++)
        rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));

    roots.resize(1 << nbase);

    while(base < nbase){
        ld angle = 2*PI / (1 << (base + 1));
        for(int i = 1 << (base - 1); i < (1 << base); i++){
            roots[i << 1] = roots[i];
            ld angle_i = angle * (2 * i + 1 - (1 << base));
            roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
        }
        base++;
    }
}

void fft(vector<num> &a, int n = -1){
    if(n == -1)
        n = a.size();

    assert((n & (n-1)) == 0);
    int zeros = __builtin_ctz(n);
    ensure_base(zeros);
    int shift = base - zeros;
    for(int i = 0; i < n; i++)
        if(i < (rev[i] >> shift))
            swap(a[i], a[rev[i] >> shift]);

    for(int k = 1; k < n; k <<= 1)
        for(int i = 0; i < n; i += 2 * k)
            for(int j = 0; j < k; j++){
                num z = a[i+j+k] * roots[j+k];
                a[i+j+k] = a[i+j] - z;
                a[i+j] = a[i+j] + z;
            }
}

vector<num> fa, fb;
vi multiply(vi &a, vi &b){
    int need = a.size() + b.size() - 1;
    int nbase = 0;
    while((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    if(sz > (int) fa.size())
        fa.resize(sz);

    for(int i = 0; i < sz; i++){
        int x = (i < (int) a.size() ? a[i] : 0);
        int y = (i < (int) b.size() ? b[i] : 0);
        fa[i] = num(x, y);
    }
    fft(fa, sz);
    num r(0, -0.25 / sz);
    for(int i = 0; i <= (sz >> 1); i++){
        int j = (sz - i) & (sz - 1);
        num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
        if(i != j) {
            fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
        }
        fa[i] = z;
    }
    fft(fa, sz);
    vi res(need);
    for(int i = 0; i < need; i++)
        res[i] = fa[i].x + 0.5;

    return res;
}


vi multiply_mod(vi &a, vi &b, int m, int eq = 0){
    int need = a.size() + b.size() - 1;
    int nbase = 0;
    while((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    if(sz > (int) fa.size())
        fa.resize(sz);

    for(int i=0;i<(int)a.size();i++){
        int x = (a[i] % m + m) % m;
        fa[i] = num(x & ((1 << 15) - 1), x >> 15);
    }
    fill(fa.begin() + a.size(), fa.begin() + sz, num{0, 0});
    fft(fa, sz);
    if(sz > (int) fb.size())
        fb.resize(sz);
    if(eq)
        copy(fa.begin(), fa.begin() + sz, fb.begin());
    else{
        for(int i = 0; i < (int) b.size(); i++){
            int x = (b[i] % m + m) % m;
            fb[i] = num(x & ((1 << 15) - 1), x >> 15);
        }
        fill(fb.begin() + b.size(), fb.begin() + sz, num{0, 0});
        fft(fb, sz);
    }
    ld ratio = 0.25 / sz;
    num r2(0, -1);
    num r3(ratio, 0);
    num r4(0, -ratio);
    num r5(0, 1);
    for(int i=0;i<=(sz >> 1);i++) {
        int j = (sz - i) & (sz - 1);
        num a1 = (fa[i] + conj(fa[j]));
        num a2 = (fa[i] - conj(fa[j])) * r2;
        num b1 = (fb[i] + conj(fb[j])) * r3;
        num b2 = (fb[i] - conj(fb[j])) * r4;
        if(i != j){
            num c1 = (fa[j] + conj(fa[i]));
            num c2 = (fa[j] - conj(fa[i])) * r2;
            num d1 = (fb[j] + conj(fb[i])) * r3;
            num d2 = (fb[j] - conj(fb[i])) * r4;
            fa[i] = c1 * d1 + c2 * d2 * r5;
            fb[i] = c1 * d2 + c2 * d1;
        }
        fa[j] = a1 * b1 + a2 * b2 * r5;
        fb[j] = a1 * b2 + a2 * b1;
    }
    fft(fa, sz);
    fft(fb, sz);
    vi res(need);
    for(int i=0;i<need;i++){
        ll aa = fa[i].x + 0.5;
        ll bb = fb[i].x + 0.5;
        ll cc = fa[i].y + 0.5;
        res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
    }
    return res;
}
```

```

int main()
{sws;

    //FFT
    vi fx{1, 2, 3}; // 1+2x+3x^2
    vi gx{4, 5}; // 4+5x
    vi res;

    res = multiply(fx,gx); //4 + 13x + 22x^2 + 15x^3

    return 0;

}
```

5.16 Next-Permutation

```
vector<int> a = {1, 2, 3};
int n = a.size();
do{
    display(a, n);// 1,2,3; 1,3,2; 2,1,3; 3,1,2;
    2,3,1; 3,2,1;
}while(next_permutation(a.begin(), a.begin() + n));
```

5.17 Fast-Exponentiation

```
// Modular exponentiaion - (x^y)%mod in O(log y)
ll power(ll x, ll y, ll mod){
    ll res = 1;
    x%=mod;
    while(y){
        if(y&1)
            res=(res*x)%mod;
        y=y>>1;
        x=(x*x)%mod;
    }
    return res;
}
```

5.18 Recursao-linear

```
vector<vl> id(int n) {
    vector<vl> res(n, vl(n, 0));
    for(int i = 0; i < n; i++) res[i][i] = 1;
    return res;
}

vector<vl> mult(vector<vl> a, vector<vl> b, int n) {
    vector<vl> res(n, vl(n, 0));

    for(int row = 0; row < n; row++) {
        for(int col = 0; col < n; col++) {
            ll val = 0;
            for(int k = 0; k < n; k++) {
                ll delta = (a[row][k] * b[k][col]) %
    MOD;
                val = (val + delta) % MOD;
            }
            res[row][col] = val;
        }
    }

    return res;
}

vector<vl> fexp(vector<vl> b, ll e, int n) {
    if(e == 0) {
        return id(n);
    }
    vector<vl> res = fexp(b, e/2, n);
    res = mult(res, res, n);

    if(e%2)
        res = mult(res, b, n);

    return res;
}

// k = tamanho da recorrencia/matriz, n = n-esimo
    termo
// f(n) = c1*f(n-1) + c2*f(n-2) + ... + ck*f(n-k)
// base -> [f(k-1), f(k-2), ..., f(0)]
// coeficientes -> [c1, c2, ..., ck]
vl solve(int k, int n, vl base, vl coef) {
    vector<vl> inicial;
    inicial.pb(coef);
    for(int row = 0; row < k-1; row++) {
        vl tmp;
        for(int col = 0; col < k; col++) {
            if(col == row)
                tmp.pb(1);
            else
                tmp.pb(0);
        }
        inicial.pb(tmp);
    }

    vector<vl> matexp = fexp(inicial, max(0, n-k+1),
    k);
    vl res(k);

    for(int row = 0; row < k; row++) {
        ll val = 0;
        for(int aux = 0; aux < k; aux++) {
            val += matexp[row][aux]*base[aux];
        }
        res[row] = val; // res = (f(n), f(n-1), ...,
    f(n-k+1))
    }

    return res;
}
```

5.19 Raiz-primitiva

```
ll fexp(ll b, ll e, ll mod) {
    if(e == 0) return 1LL;
    ll res = fexp(b, e/2LL, mod);
    res = (res*res)%mod;
    if(e%2LL)
        res = (res*b)%mod;

    return res%mod;
}

vl fatorar(ll n) { // fatora em primos
    vl fat;
    for(int i = 2; i*i <= n; i++) {
        if(n%i == 0) {
            fat.pb(i);
            while(n%i == 0)
                n /= i;
        }
    }
    return fat;
}

// O(log(n) ^ 2)
bool raiz_prim(ll a, ll mod, ll phi, vl fat) {
    if(__gcd(a, mod) != 1 or fexp(a, phi/2, mod) ==
    1) // phi de euler sempre eh PAR
        return false;
```

```
    for(auto f : fat) {
        if(fexp(a, phi/f, mod) == 1)
            return false;
    }

    return true;
}

// mods com raizes primitivas: 2, 4, p^k, 2*p^k, p eh primo impar, k inteiro --- O(n log^2(n))
ll achar_raiz(ll mod, ll phi) {
    if(mod == 2) return 1;
    vl fat, elementos;
    fat = fatorar(phi);

    for(ll i = 2; i <= mod-1; i++) {
        if(raiz_prim(i, mod, phi, fat))
            return i;
    }

    return -1; // retorna -1 se nao existe
}

vl todas_raizes(ll mod, ll phi, ll raiz) {
    vl raizes;
    if(raiz == -1) return raizes;
    ll r = raiz;
    for(ll i = 1; i <= phi-1; i++) {
        if(__gcd(i, phi) == 1) {
            raizes.pb(r);
        }
        r = (r * raiz) % mod;
    }

    return raizes;
}
```

5.20 Kamenetsky

```
// Number of digits in n! O(1)

#define Pi 3.14159265358979311599796346854
#define Eul 2.71828182845904509079559829842

long long findDigits(int n)
{
    double x;

    if (n < 0)
        return 0;
    if (n == 1)
        return 1;

    x = ((n * log10(n / euler) + log10(2 * Pi * n)/2.0));

    return floor(x) + 1;
}
```

6 Misc

6.1 2SAT

```
vector<int> g[MAX], gt[MAX], S; int vis[MAX], cor[MAX];

int val(int n, bool tvalue) {
    if(tvalue) return 2*n;
    return 2*n +1;
}

void dfs(int u) {
    vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
    S.push_back(u);
}

void dfst(int u, int e) {
    cor[u] = e;
    for(int v : gt[u]) if(!cor[v]) dfst(v, e);
}

void kosaraju(int n) {
    for(int i = 0; i <= n; i++) if(!vis[i]) dfs(i);
    for(int i = 0; i <= n; i++) for(int j : g[i])
        gt[j].push_back(i);
    int e = 0; reverse(S.begin(), S.end());
    for(int u : S) if(!cor[u]) dfst(u, ++e);
}

// antes de chamar essa funcao, colocar as arestas do grafo
bool solve(int n, vi &res) {
    kosaraju(2*n); // MAX > 2*N
    vi r;

    forn(i, n) {
        int t = val(i, true), f = val(i, false);
        if(cor[t] == cor[f]) {
            return false;
        }
        else {
            if(cor[t] > cor[f])
                r.pb(1);
            else
                r.pb(0);
        }
    }
    swap(r, res);
    return true;
}
```

6.2 LIS

```
multiset<int> S;
for(int i = 0; i < n; i++){
    auto it = S.upper_bound(vet[i]); // low for inc
    if(it != S.end())
        S.erase(it);
    S.insert(vet[i]);
}
// size of the lis
int ans = S.size();
```

6.3 Bitwise

```
// Bitwise
    #pragma GCC target("popcnt")
    unsigned char a = 5, b = 9; // a = (00000101), b = (00001001)

    AND -           a&b   // The result is 00000001 (1)
    OR -            a|b   // The result is 00001101 (13)
    XOR -           a^b   // The result is 00001100 (12)
    NOT -           ~a    // The result is 11111010 (250)
    Left shift -    b<<1  // The result is 00010010 (18)
    Right shift -   b>>1  // The result is 00000100 (4)
```

```
// Exchange two int variables

    a^=b;
    b^=a;
    a^=b;

// Even or Odd

    (x & 1)? printf("Odd"): printf("Even");

// Turn on the j-th bit

    int S = 34; //(100010)
    int j = 3;

    S = S | (1<<j);

// Turn off the j-th bit

    int S = 42; //(101010)
    int j = 1;

    S &= ~(1<<j)

    S == 40 //(101000)

// Check the j-th element

    int S = 42; //(101010)
    int j = 3;

    T = S & (1<<j); // T = 0

// Least significant bit (lsb)

    int lsb(int x){ return x&-x; }

// Exchange o j-th element

    S ^= (1<<j)

// Position of the first bit on

    T = (S & (-S))
    T -> 4 bit ligado //(1000)

// Most significant digit of N

    double K = log10(N);
    K = K - floor(K);
    int X = pow(10, K);

// Number of digits in N

    X =floor(log10(N)) + 1;

// Power of two

    bool isPowerOfTwo(int x){ return x && (!(x&(x-1))); }

// Turn off the first bit 1
    m = m & (m-1);

// Built-in functions

    // Number of bits 1
    __builtin_popcount()
    __builtin_popcountll()

    // Number of leading zeros
    __builtin_clz()
    __builtin_clzll()

    // Number of trailing zeros
    __builtin_ctz()
    __builtin_ctzll()

// floor(log2(x))

    int flog2(int x){ return 32-1-__builtin_clz(x); }

    int flog2ll(ll x){ return 64-1-__builtin_clzll(x); }
```

6.4 All-Subsets

```
vi a, subset;
vector<vi> subsets;

// Iterative
void search(int k){
    if(k==(int)a.size())
        subsets.pb(subset);
    else{
        search(k+1);
        subset.pb(a[k]);
        search(k+1);
        subset.pop_back();
    }
}
search(0);

// Binary
for(int b=0;b<(1<<n);b++){
    vi subset;
    for(int i=0;i<n;i++)
        if(b&(1<<i)) subset.pb(a[i]);
    subsets.pb(subset);
}
```

6.5 Template

```
#include <bits/stdc++.h>
#define ff first
#define ss second
#define ll long long
#define ld long double
#define pb push_back
#define eb emplace_back
#define mp make_pair
#define mt make_tuple
#define pii pair<int, int>
#define vi vector<int>
#define vl vector<ll>
#define vii vector<pii>
#define sws ios_base::sync_with_stdio(false);cin.tie(NULL)
#define endl '\n'
#define teto(a, b) (a+b-1)/(b)

const int MAX = 300010;
const int MOD = 1e9;
const int INF = 0x3f3f3f3f;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;
const ld EPS = 1e-8;
const ld PI = acosl(-1.0);

using namespace std;
```

6.6 Rand

```
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
uniform_int_distribution<int> distribution(1,n);

num = distribution(rng); // num no range [1, n]
shuffle(vec.begin(), vec.end(), rng); // shuffle
```

6.7 Meet-in-the-middle

```
// Subsequence with the biggest sum%m value O(2^(n/2)
    *n)

int n, m, a[40];

void comb(int l, int r, vi &v){
    int sz = r-l+1;
    for(int i=0;i<(1<<sz);i++){
        int sum = 0;
        for(int j=0;j<sz;j++)
            if(i & (1<<j))
                sum = (sum + a[l+j])%m;
        v.pb(sum);
    }

    sort(v.begin(), v.end());
}

int merge(vi &x, vi &y){
    int k=y.size()-1, ans=0;
    for(auto v: x){
        while(k>0 and v+y[k]>=m)
            k--;
        ans = max(ans, v+y[k]);
    }
    return ans;
}


int main()
{sws;

    vi x, y;
    cin >> n >> m;

    for(int i=0;i<n;i++)
        cin >> a[i];

    comb(0, n/2, x);
    comb(n/2 + 1, n-1, y);
    cout << merge(x, y) << endl;

    return 0;
}
```

7 Strings

7.1 Trie

```
int trie[MAX][26];
bool finish[MAX];
int nxt = 1;

void Add(string &str){
    int node = 0;
    for(auto s: str){
        if(trie[node][s-'a'] == 0){
            node = trie[node][s-'a'] = nxt;
            nxt++;
        }else
            node = trie[node][s-'a'];
    }
    finish[node] = true;
}

bool Find(string &str){
    int idx = 0;
    for(auto s: str)
        if(trie[idx][s-'a'] == 0)
            return false;
        else
            idx = trie[idx][s-'a'];
    return finish[idx];
}
```

7.2 KMP

```
vi pi(const string &s){
    int n=s.size();
    vi p(n);
    for(int i=1, j=0; i<n; i++){
        while(j>0 and s[i]!=s[j]) j=p[j-1];
        if(s[j]==s[i]) j++;
        p[i]=j;
    }
    return p;
}

vi kmp(const string &t, const string &s){
    vi p = pi(s+'$'), match;
    int n=t.size(), m=s.size();
    for(int i=0, j=0; i<n; i++){
        while(j>0 and t[i]!=s[j]) j=p[j-1];
        if(t[i]==s[j]) j++;
        if(j==m) match.pb(i-j+1);
    }
    return match;
}
```

7.3 Suffix-array

```
vi suffix_array(string s){
    s.pb('$');
    int n = s.size();

    vi p(n), c(n);
    vector< pair<char, int> > a(n);
    for(int i=0;i<n;i++) a[i] = {s[i], i};
    sort(a.begin(), a.end());

    for(int i=0;i<n;i++) p[i] = a[i].ss;
    c[p[0]]=0;
    for(int i=1;i<n;i++)
        c[p[i]] = c[p[i-1]] + (a[i].ff!=a[i-1].ff);

    int k=0;
    while((1<<k) < n){
        vector< pair<pii, int> > a(n);
        for(int i=0;i<n;i++)
            a[i] = {{c[i], c[(i+(1<<k))%n]}, i};
        sort(a.begin(), a.end());

        for(int i=0;i<n;i++) p[i] = a[i].ss;
        c[p[0]]=0;
        for(int i=1;i<n;i++)
            c[p[i]] = c[p[i-1]] + (a[i].ff!=a[i-1].ff
    );
        k++;
    }
    return p;
}
```

7.4 LCS

```
string LCSubStr(string X, string Y)
{
    int m = X.size();
    int n = Y.size();

    int result = 0, end;
    int len[2][n];
    int currRow = 0;

    for(int i=0;i<=m;i++){
        for(int j=0;j<=n;j++){
            if(i==0 || j==0)
                len[currRow][j] = 0;
            else if(X[i-1] == Y[j-1]){
                len[currRow][j] = len[1-currRow][j-1]
     + 1;
                if(len[currRow][j] > result){
                    result = len[currRow][j];
                    end = i - 1;
                }
            }
            else
                len[currRow][j] = 0;
        }

        currRow = 1 - currRow;
    }

    if(result==0)
        return string();

    return X.substr(end - result + 1, result);
}
```

7.5 Pal-int

```
bool ehpalindromo(ll n) {
    if(n<0)
        return false;

    int divisor = 1;
    while(n/divisor >= 10)
        divisor *= 10;

    while(n != 0) {
        int leading = n / divisor;
        int trailing = n % 10;

        if(leading != trailing)
            return false;

        n = (n % divisor)/10;

        divisor = divisor/100;
    }

    return true;
}
```

7.6 Z-Func

```
vi z_algo(const string &s)
{   // returns vector for each idx where a prefix of
    size i starts.
    int n = s.size();
    int L = 0, R = 0;
    vi z(n, 0);
    for(int i = 1; i < n; i++){
        if(i <= R)
            z[i] = min(z[i-L], R - i + 1);
        while(z[i]+i < n and s[ z[i]+i ] == s[ z[i]
    ])
            z[i]++;
        if(i+z[i]-1 > R){
            L = i;
            R = i + z[i] - 1;
        }
    }
    return z;
}
```

7.7 Hash

```
ll compute_hash(string const& s) {
    const ll p = 31; // primo, melhor = perto da
    quantidade de caracteres
    const ll m = 1e9 + 9; // maior mod = menor
    probabilidade de colisao
    ll hash_value = 0;
    ll p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) *
    p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

7.8 Manacher

```
// O(n), d1 -> palindromo impar, d2 -> palindromo par
     (centro da direita)
void manacher(string &s, vi &d1, vi &d2) {
    int n = s.size();
    for(int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r -
    i + 1);
        while(0 <= i - k && i + k < n && s[i - k] ==
    s[i + k]) {
            k++;
        }
        d1[i] = k--;
        if(i + k > r) {
            l = i - k;
            r = i + k;
        }
    }

    for(int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1],
    r - i + 1);
        while(0 <= i - k - 1 && i + k < n && s[i - k
    - 1] == s[i + k]) {
            k++;
        }
        d2[i] = k--;
        if(i + k > r) {
            l = i - k - 1;
            r = i + k ;
        }
    }
}
```