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) // Objetos | Peso max
{
    for(i=0;i<=N;i++)
    {
       for(j=0;j<=M;j++)
       {
            if (i==0 || 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 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;
}
```

1.3 Iterative-BS

```
int main()
{
    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;

    return 0;
}
```

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

```
#define vi vector<int>

vector< vector<int> > grafo;
vector<bool> visited;
vi t, low;
int timer=0;

void find_bridges(int v, int p=-1)
{
    visited[v] = true;
    t[v] = low[v] = timer++;
    for(int i=0;i<(int)grafo[v].size();i++)
    {
        int vert = grafo[v][i];
        if(vert == p)
            continue;
        if(visited[vert])
            low[v] = min(low[v], t[vert]);
        else
        {
            find_bridges(vert, v);
            low[v] = min(low[v], low[vert]);
            if(low[to] > t[v])
                IS_BRIDGE(v, vert);
        }
    }
}

int main()
{
    timer = 0;
    visited.assign(N+1, false);
    t.assign(N+1, 0);
    low.assign(N+1, 0);

    for(int i=0;i<N;i++)
        if(!visited[i])
            find_bridges(1);

    return 0;
}
```

2.3 Dijkstra

```
// 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

```
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

```
// 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

```
// 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 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);
}
```

2.9 Represent

```
// Grafos

// List of edges

    vector< pair<int, int> > arestas;
    arestas.push_back(make_pair(1, 2));
    arestas.push_back(make_pair(1, 3));

// Adjacency Matrix

    int grafo[10][10];

    grafo[1][2] = grafo[2][1] = 1;
    grafo[1][3] = grafo[3][1] = 2;

// Adjacency List

    vector<int> vizinhos[10];

    vizinhos[1].push_back(2);
    vizinhos[1].push_back(2);
```

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 Angle-adjacent-vertices-regular-polygon

$a = 180/N$

3.2 Inter-Retas

```
// Intersection between lines

typedef struct
{
    int x, y;
} pnt;

bool collinear(pnt p, pnt q, pnt r)
{
    if(q.x<=max(p.x,r.x) && q.x>=min(p.x,r.x) && q.y<=max(p.y,r.y) && q.y>=min(p.y,r.y))
        return true;

    return false;
}

int orientation(pnt p, pnt q, pnt r)
{
    int val=(q.y-p.y)*(r.x-q.x)-(q.x-p.x)*(r.y-q.y);

    if(val==0)
        return 0;
    else if(val>0)
        return 1;
    else
        return 2;
}

bool intersect(pnt p1, pnt q1, pnt p2, pnt q2)
{
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if(o1!=o2 and o3!=o4)
        return true;

    if(o1==0 && collinear(p1, p2, q1))
        return true;

    if(o2==0 && collinear(p1, q2, q1))
        return true;

    if(o3==0 && collinear(p2, p1, q2))
        return true;

    if(o4==0 && collinear(p2, q1, q2))
        return true;

    return false;

}
```

3.3 Pick's-theorem

- The area of a polygon with integer coordinates: $A = i + \frac{b}{2} - 1$
- i is the number os points inside the polygon;
- b is the number of points on the boundry;
- $2A$ is necessarily an integer value.

3.4 Cross-properties

- It equals zero if the vectors **a** and **b** are collinear (coplanar in triple product).
- It is positive if the rotation from the first to the second vector is clockwise and negative otherwise.

3.5 Inter-Retangulos

```
typedef struct
{
    int x, y;
} Point;

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;
}
```

3.6 3D

```
typedef ld cod;

bool eq(cod a, cod b){ return abs(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);
    }
};

// Produto Escalar
cod dot(point a, point b){
    return a.x*b.x + a.y*b.y + a.z*b.z;
}

// Produto Vetorial
point cross(point a, point b){
    return point(a.y*b.z - a.z*b.y,
                 a.z*b.x - a.x*b.z,
                 a.x*b.y - a.y*b.x);
}

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

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

3.7 Dot-properties

- Length of $\mathbf{a}$: $|\mathbf{a}| = \sqrt{\mathbf{a} \cdot \mathbf{a}}$.
- Projection of $\mathbf{a}$ onto $\mathbf{b}$: $\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{b}|}$.
- Angle between vectors: $\arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| \cdot |\mathbf{b}|}\right)$.
- The dot product is positive if the angle between them is acute, negative if it is obtuse and it equals zero if they are orthogonal, i.e. they form a right angle.

3.8 2D

```
typedef ld cod;

bool eq(cod a, cod b){ return abs(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};
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y);
    }

};

struct line
{
    point fp, sp;
    point(point fp=0, point sp=0): fp(fp), sp(sp){}

    //a=y1-y2;
    //b=x2-x1;
    //c=x2*y1-y2*x1;

};


// Produto Escalar
cod dot(point a, point b){
    return a.x*b.x + a.y*b.y;
}
// Produto Vetorial
cod cross(point a, point b){
    return a.x*b.y - a.y*b.x;
}

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

```
    return acos(dot(a, b) / norm(a) / norm(b));
}
cod dist(point a, point b){
    return norm(a-b); // Dist euclidiana
}
int ccw(point a, point b){ // -1=cw; 0=parallel; 1=ccw;
    cod tmp = cross(a, b); // from a to b.
    return (tmp > EPS) - (tmp < -EPS);
}
bool collinear(point a, point b, point c){
    return eq(cross(a-c, b-c), 0);
}


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




// Dist entre ponto e reta
cod distr(point a, line b){
    cod crs = cross(point(a - b.fp), point(b.sp - b.fp));
    return norm(crs/dist(b.fp, b.sp));
}

int esq(point a, point b, point e)
{ // From a to b: Esquerda = 1; Direita = -1; Collinear = 0;
    ll v = a.x*b.y + b.x*e.y + e.x*a.y - (a.y*b.x + b.y*e.x + e.y*a.x);
    if(v>0) return 1;
    if(v==0) return 0;
    return -1;
}

// Area de um poligono (pontos ordenados por adjacencia)
cod area(vector <point> p){
  cod ret = 0;
  for(int i=2;i<(int)p.size();i++)
    ret += cross(p[i] - p[0], p[i-1] - p[0])/2;
  return norm(ret);
}
```

4 ED

4.1 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;
}
```

4.2 Iterative-SegTree

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

#define N 100010

struct Segtree
{
    int t[2*N]={0};

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

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

        return ans;
    }

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

};

int main()
{
    Segtree st;

    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);

}
```

4.3 Recursive-SegTree

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

vector<int> val(MAX, 0);
vector<int> vet(N);

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);

}
```

4.4 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;
}
```

4.5 Seg-Tree-Farao

```
typedef struct
{
    pii prefix, sufix, total, maximo;
} no;

int noleft[MAX], noright[MAX]; //Guarda os valores dos nos para que nao sejam calculados novamente nas querys
int v[MAX];
no arvore[MAX];

pii somar(pii a, pii b) // une pairs
{
    return mp(a.f+b.f, a.s+b.s);
}

no une(no l, no r)
{
    if(l.total.s==0)
        return r;
    if(r.total.s==0)
        return l;

    no m;

    m.prefix = max(l.prefix, somar(l.total, r.prefix)); //prefixo
    m.sufix = max(r.sufix, somar(r.total, l.sufix)); //sufixo
    m.total = somar(l.total, r.total); //Soma de todos os elementos da subarvore
    m.maximo = max(max(l.maximo, r.maximo), somar(l.sufix, r.prefix)); //Resultado para cada subarvore

    return m;
}

no makenozero()
{
    no m;
    m.prefix=m.sufix=m.total=m.maximo=mp(0,0);
    return m;
}

no makeno(int k)
{
    no m;
    m.prefix=m.sufix=m.total=m.maximo=mp(k,1);
    return m;
}

void monta(int n)
{
    if(noleft[n]==noright[n])
    {
        arvore[n]=makeno(v[noleft[n]]);
        return;
    }

    int mid = (noleft[n]+noright[n])/2;
    noleft[2*n]=noleft[n]; noright[2*n]=mid;
    noleft[2*n+1]=mid+1; noright[2*n+1]=noright[n];

    monta(2*n);
    monta(2*n+1);

    arvore[n]=une(arvore[2*n], arvore[2*n+1]);
}

no busca(int n, int esq, int dir)
{
    if(noleft[n]>=esq and noright[n]<=dir)
        return arvore[n];
    if(noright[n]<esq or noleft[n]>dir)
        return makenozero();

    return une(busca(2*n, esq, dir),busca(2*n+1, esq, dir));
}

int main()
{
    int T, N, Q, A, B;
    no aux;

    scanf("%d", &T);

    while(T--)
    {
        scanf("%d", &N);
        for(int i=1;i<=N;i++)
            scanf("%d", &v[i]); //Elementos da arvore

        noleft[1]=1;noright[1]=N;
        monta(1);

        cin >> Q;
        while(Q--)
        {
            scanf("%d%d", &A, &B); //Intervalo da query
            aux = busca(1, A, B);
            printf("%d %d\n", aux.maximo.f, aux.maximo.s);
        }
    }


    return 0;
}
```

4.6 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);
}
```

4.7 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;
    }
};
```

4.8 Sparse-Table

```
logv[1] = 0; // pre-computar tabela de log
for (int i = 2; i <= MAXN; i++)
    logv[i] = logv[i/2] + 1;

int logv[MAXN+1];
int st[MAXN][K];

// operacao da sparse table deve ser idempotente -> op(x, x) = x
void precompute(int N) { //
  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] = max(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);
}

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

    return maximum;
}
```

4.9 Union-Find

```
// Union-Find Functions

int pai[MAX], peso[MAX];

int find(int aux)
{
    if(pai[aux]==aux)
        return aux;
    else
        return pai[aux]=find(pai[aux], pai);
}

void join(int x, int y)
{
    x = find(x);
    y = find(y);

    if(pesos[x]<pesos[y])
        pai[x] = y;
    else if(pesos[x]>pesos[y])
        pai[y] = x;
    else if(pesos[x]==pesos[y])
    {
        pai[x] = y;
        pesos[y]++;
    }
}

int main()
```

```
{
    for(int i=1;i<=N;i++)
        pai[i]=i;
}
```

4.10 Mo

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

struct Query {
    int l, r, 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<pair<int,ll>> mo() {
    vector<pair<int,ll>> res;
    sort(queries.begin(), queries.end());

    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.pb(mp(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)))
vector<ll> phi_to_n(ll n) {
    vector<bool> isprime(n+1, true);
    vector<ll> 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 Sum-n2

Soma dos n primeiros números ao quadrado $= \frac{(2N^3+3N^2+N)}{6}$

5.5 Factorization-sqrt

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

int main()
{
    ll N;
    vector<int> div;

    cin >> N;

    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 0;
}
```

5.6 Modular-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;
}
```

5.7 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;

}
```

5.8 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);
}
```

5.9 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;

}
```

5.10 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;
}
```

5.11 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;
```

5.12 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);
}
```

5.13 FFT

```
struct num{
    double x, y;
    num() { x = y = 0; }
    num(double x, double 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}};
vector<int> rev = {0, 1};

const double PI = acosl(-1.0);

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){
        double angle = 2*PI / (1 << (base + 1));
        for(int i = 1 << (base - 1); i < (1 << base); i++){
            roots[i << 1] = roots[i];
            double 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;
vector<int> multiply(vector<int> &a, vector<int> &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);
    vector<int> res(need);
    for(int i = 0; i < need; i++)
        res[i] = fa[i].x + 0.5;

    return res;
}

int main()
{sws;

    //FFT
    vector<int> fx{1, 2, 3}; // 1+2x+3x^2
    vector<int> gx{4, 5}; // 4+5x
    vector<int> res;

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

    return 0;

}
```

5.14 Modular-Factorial

```
// C++ program to comput n! % p using Wilson’s Theorem
#include <bits/stdc++.h>
using namespace std;

int power(int x, unsigned int y, int p)
{
    int res = 1;
    x = x % p;

    while(y > 0)
    {
        if(y & 1)
            res = (res * x) % p;

        y = y >> 1;
        x = (x * x) % p;
    }
    return res;
}

int modInverse(int a, int p)
{
    return power(a, p-2, p);
}

int modFact(int n, int p)
{
    if (p <= n)
        return 0;

    int res = (p - 1);

    for(int i = n + 1; i < p; i++)
        res = (res * modInverse(i, p)) % p;
    return res;
}

int main()
{
    int n = 25, p = 29;
    cout << modFact(n, p);
    return 0;
}
```

5.15 Recursao-linear

```
vector<vl> mult(vector<vl> a, vector<vl> b, int n) {
    vector<vl> res;
    for(int i = 0; i < n; i++) {
        vl tmp;
        for(int j = 0; j < n; j++) {
            tmp.pb(0);
        }
        res.pb(tmp);
    }

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

    return res;
}

vector<vl> fexp(vector<vl> b, ll e, int n) {
    if(e == 0) {
        vector<vl> id;
        for(int i = 0; i < n; i++) {
            vl tmp;
            for(int j = 0; j < n; j++) {
                if(i == j)
                    tmp.pb(1);
                else
                    tmp.pb(0);
            }
            id.pb(tmp);
        }

        return id;
    }

    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;
}
```

5.16 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 LIS

```cpp
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();
```

6.2 Bitwise

```cpp
// Bitwise

    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); }
```

6.3 Template

```cpp
#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 sws ios_base::sync_with_stdio(false);cin.tie(NULL)
#define endl '\n'
#define teto(a, b) (a+b-1)/(b)

const int MAX = 400010;
const int MOD = 1e9+7;
const int INF = 0x3f3f3f3f;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;
const ld EPS = 1e-7;

using namespace std;
```

7 Strings

7.1 KMP

```
vector<int> preffix_function(const string &s){
    int n = s.size(); vector<int> b(n+1);
    b[0] = -1; int i = 0, j = -1;
    while(i < n){
        while(j >= 0 && s[i] != s[j]) j = b[j];
        b[++i] = ++j;
    }
    return b;
}
void kmp(const string &t, const string &p){
    vector<int> b = preffix_function(p);
    int n = t.size(), m = p.size();
    int j = 0;
    for(int i = 0; i < n; i++){
        while(j >= 0 && t[i] != p[j]) j = b[j];
        j++;
        if(j == m){

            j = b[j];
        }
    }
}
```

7.2 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);
}
```

7.3 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;
}
```

7.4 Z-Func

```
vector<int> z_algo(const string &s)
{
    int n = s.size();
    int L = 0, R = 0;
    vector<int> 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 && s[ z[i]+i ] == s[ z[i] ])
            z[i]++;
        if(i+z[i]-1 > R)
        {
            L = i;
            R = i + z[i] - 1;
        }
    }
    return z;
}
```

7.5 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;
}
```