



# Notebook - Maratona de Programação

Tiago de Souza Fernandes

## Sumário

<b>1 Algoritmos</b>	<b>2</b>	5.7 Verif-primo . . . . .	11
1.1 Mochila . . . . .	2	5.8 Crivo . . . . .	11
1.2 Iterative-BS . . . . .	2	5.9 Formulas . . . . .	12
<b>2 Grafos</b>	<b>2</b>	5.10 FFT-golfbot . . . . .	12
2.1 BFS . . . . .	2	5.11 Modular-Factorial . . . . .	12
2.2 Find-bridges . . . . .	2	5.12 Kamenetsky . . . . .	13
2.3 Dijkstra . . . . .	2	<b>6 Misc</b>	<b>13</b>
2.4 Floyd-Warshall . . . . .	3	6.1 LIS . . . . .	13
2.5 Kruskal . . . . .	3	6.2 Bitwise . . . . .	13
2.6 DFS . . . . .	3	<b>7 Strings</b>	<b>14</b>
2.7 Represent . . . . .	3	7.1 KMP . . . . .	14
2.8 Prim . . . . .	3	7.2 Pal-int . . . . .	14
<b>3 Geometria</b>	<b>4</b>	7.3 Z-Func . . . . .	14
3.1 Inter-Retas . . . . .	4		
3.2 Inter-Retangulos . . . . .	4		
3.3 Analytic-Geometry . . . . .	5		
<b>4 ED</b>	<b>5</b>		
4.1 Range-query-bigger-than-k-BIT . . . . .	5		
4.2 Iterative-SegTree . . . . .	6		
4.3 Recursive-SegTree . . . . .	6		
4.4 Delta-Encoding . . . . .	7		
4.5 Seg-Tree-Farao . . . . .	7		
4.6 BIT-2D . . . . .	8		
4.7 BIT . . . . .	8		
4.8 Sparse-Table . . . . .	8		
4.9 Union-Find . . . . .	9		
<b>5 Math</b>	<b>9</b>		
5.1 Linear-Diophantine-Equation . . . . .	9		
5.2 Factorization-sqrt . . . . .	9		
5.3 Modular-Exponentiation . . . . .	9		
5.4 Miller-Habin . . . . .	9		
5.5 Inverso-Mult . . . . .	11		
5.6 Pollard-Rho . . . . .	11		

# 1 Algoritmos

## 1.1 Mochila

```
1 int val[MAXN], peso[MAXN], dp[MAXN][MAXS]
2
3 int knapsack(int N, int M) // Objetos | Peso max
4 {
5     for(i=0; i<=N; i++)
6     {
7         for(j=0; j<=M; j++)
8         {
9             if (i==0 || j==0)
10                dp[i][j] = 0;
11             else if (peso[i-1] <= j)
12                dp[i][j] = max(val[i-1]+dp[i-1][j-
13                    peso[i-1]], dp[i-1][j]);
14             else
15                dp[i][j] = dp[i-1][j];
16         }
17     }
18     return dp[N][M];
19 }
```

## 1.2 Iterative-BS

```
1 int main()
2 {
3     int l=1, r=N;
4     int res=-1;
5
6     while(l <= r)
7     {
8         int m = (l + r)/2;
9         if(!ver(m))
10         {
11             l = m+1;
12         }
13         else
14         {
15             res = m;
16             r = m-1;
17         }
18     }
19     cout << res << endl;
20
21     return 0;
22 }
```

# 2 Grafos

## 2.1 BFS

```
1 //BFS (Breadth First Search) O(V+A)
2
3 vector<vector<int>> adj; // adjacency list
4 representation
5 int n; // number of nodes
6 int s; // source vertex
7
8 queue<int> q;
9 vector<int> d(n, INF);
10
11 q.push(s);
12 used[s] = true;
13 while (!q.empty()) {
14     int v = q.front();
15     q.pop();
16     for (int u : adj[v]) {
17         if (d[u] > d[v] + 1) {
```

```
17         q.push(u);
18         d[u] = d[v] + 1;
19     }
20 }
21 }
```

## 2.2 Find-bridges

```
1 #define vi vector<int>
2
3 vector< vector<int> > grafo;
4 vector<bool> visited;
5 vi t, low;
6 int timer=0;
7
8 void find_bridges(int v, int p=-1)
9 {
10     visited[v] = true;
11     t[v] = low[v] = timer++;
12     for(int i=0; i<(int)grafo[v].size(); i++)
13     {
14         int vert = grafo[v][i];
15         if(vert == p)
16             continue;
17         if(visited[vert])
18             low[v] = min(low[v], t[vert]);
19         else
20         {
21             find_bridges(vert, v);
22             low[v] = min(low[v], low[vert]);
23             if(low[vert] > t[v])
24                 IS_BRIDGE(v, vert);
25         }
26     }
27 }
28
29 int main()
30 {
31     timer = 0;
32     visited.assign(N+1, false);
33     t.assign(N+1, 0);
34     low.assign(N+1, 0);
35
36     for(int i=0; i<N; i++)
37         if(!visited[i])
38             find_bridges(i);
39
40     return 0;
41 }
```

## 2.3 Dijkstra

```
1 // Dijkstra - Shortest Path
2
3 #define pii pair<int, int>
4 #define vi vector<int>
5 #define vii vector< pair<int,int> >
6 #define INF 0x3f3f3f3f
7
8 vector<vii> grafo(N+1, vii());
9 vi distancia(N+1, INF);
10 priority_queue< pii, vii, greater<pii> > fila;
11
12 void dijkstra(int k)
13 {
14     int dist, vert, aux;
15     distancia[k]=0;
16
17     fila.push(mp(k, 0));
18
19     while(!fila.empty())
20     {
```

```

21     aux=fila.top().f;
22     fila.pop();
23
24     for(auto v: grafo[aux])
25     {
26         vert=v.f;
27         dist=v.s;
28         if(distancia[vert]>distancia[aux]+dist)
29         {
30             distancia[vert]=distancia[aux]+dist;
31             fila.push(mp(vert, distancia[vert]));
32         }
33     }
34 }
35 }
36
37 int main()
38 {
39     for(int i=0; i<M; i++)
40     {
41         cin >> a >> b >> p;
42         grafo[a].pb(mp(b, p));
43         grafo[b].pb(mp(a, p));
44     }
45 }

```

## 2.4 Floyd-Warshall

```

1 // Floyd Warshall
2
3 int dist[MAX][MAX];
4
5 void Floydwarshall()
6 {
7     for(int k = 1; k <= n; k++)
8         for(int i = 1; i <= n; i++)
9             for(int j = 1; j <= n; j++)
10                 dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
11 }

```

## 2.5 Kruskal

```

1 // Kruskal - Minimum Spanning Tree
2
3 typedef struct
4 {
5     int A, B;
6     int dist;
7 } vertice;
8
9 vertice grafo[MAX];
10 int pai[MAX];
11
12 int find(int X) // Union-Find
13 {
14     if(pai[X]==X)
15         return X;
16     else
17         return pai[X]=find(pai[X]);
18 }
19
20 void join(int X, int Y)
21 {
22     int paix = find(X);
23     int paiz = find(Y);
24     pai[paix]=paiz;
25 }
26
27 bool comp(vertice A, vertice B)
28 {

```

```

29     return A.dist < B.dist;
30 }
31
32 void kruskal()
33 {
34     for(int i=1; i<=N; i++)
35         pai[i]=i;
36
37     for(int i=1; i<=M; i++)
38         cin >> grafo[i].A >> grafo[i].B >> grafo[i].dist;
39
40     sort(grafo+1, grafo+M+1, comp);
41
42     for(int i=1; i<M; i++)
43     {
44         if(find(grafo[i].A)!=find(grafo[i].B))
45         {
46             join(grafo[i].A, grafo[i].B);
47             soma+=grafo[i].dist;
48         }
49     }
50
51     cout << soma << endl;
52 }

```

## 2.6 DFS

```

1 //DFS (Depth First Search) O(V+A)
2
3 void DFS(int x)
4 {
5     for(int i=0; i<(int)vizinhos[x].size(); i++)
6     {
7         int v = vizinhos[x][i];
8         if(componente[v] == -1)
9         {
10             componente[v] = componente[x];
11             DFS(v);
12         }
13     }
14 }

```

## 2.7 Represent

```

1 // Grafos
2
3 // List of edges
4
5 vector< pair<int, int> > arestas;
6 arestas.push_back(make_pair(1, 2));
7 arestas.push_back(make_pair(1, 3));
8
9 // Adjacency Matrix
10
11 int grafo[10][10];
12
13 grafo[1][2] = grafo[2][1] = 1;
14 grafo[1][3] = grafo[3][1] = 2;
15
16 // Adjacency List
17
18 vector<int> vizinhos[10];
19
20 vizinhos[1].push_back(2);
21 vizinhos[1].push_back(3);

```

## 2.8 Prim

```

1 // Prim Algorithm
2 #define MAXN 10100
3 #define INFINTO 999999999

```

```

4
5 int n, m;
6 int distancia[MAXN];
7 int processado[MAXN];
8 vector<pii> vizinhos[MAXN];
9
10 int Prim()
11 {
12     for(int i = 2; i <= n; i++) distancia[i] = INFINITO;
13     distancia[1] = 0;
14
15     priority_queue< pii, vector<pii>, greater<pii> >
16     fila;
17     fila.push( pii(distancia[1], 1) );
18
19     while(1)
20     {
21         int davez = -1;
22
23         while(!fila.empty())
24         {
25             int atual = fila.top().second;
26             fila.pop();
27
28             if(!processado[atual])
29             {
30                 davez = atual;
31                 break;
32             }
33
34             if(davez == -1)
35                 break;
36
37             processado[davez] = true;
38
39             for(int i = 0; i < (int)vizinhos[davez].size(); i++)
40             {
41
42                 int dist = vizinhos[davez][i].first;
43                 int atual = vizinhos[davez][i].second;
44
45                 if( distancia[atual] > dist && !
46                 processado[atual])
47                 {
48                     distancia[atual] = dist;
49                     fila.push( pii(distancia[atual],
50                     atual) );
51                 }
52             }
53
54             int custo_arvore = 0;
55             for(int i = 1; i <= n; i++)
56                 custo_arvore += distancia[i];
57
58             return custo_arvore;
59         }
60
61     int main(){
62
63         cin >> n >> m;
64
65         for(int i = 1; i <= m; i++){
66
67             int x, y, tempo;
68             cin >> x >> y >> tempo;
69
70             vizinhos[x].pb( pii(tempo, y) );
71             vizinhos[y].pb( pii(tempo, x) );
72
73         }

```

```

72
73         cout << Prim() << endl;
74
75         return 0;
76     }

```

## 3 Geometria

### 3.1 Inter-Retas

```

1 // Intersection between lines
2
3 typedef struct
4 {
5     int x, y;
6 } pnt;
7
8 bool collinear(pnt p, pnt q, pnt r)
9 {
10     if(q.x<=max(p.x,r.x) && q.x>=min(p.x,r.x) && q.y
11     <=max(p.y,r.y) && q.y>=min(p.y,r.y))
12         return true;
13
14     return false;
15 }
16
17 int orientation(pnt p, pnt q, pnt r)
18 {
19     int val=(q.y-p.y)*(r.x-q.x)-(q.x-p.x)*(r.y-q.y);
20
21     if(val==0)
22         return 0;
23     else if(val>0)
24         return 1;
25     else
26         return 2;
27 }
28
29 bool intersect(pnt p1, pnt q1, pnt p2, pnt q2)
30 {
31     int o1 = orientation(p1, q1, p2);
32     int o2 = orientation(p1, q1, q2);
33     int o3 = orientation(p2, q2, p1);
34     int o4 = orientation(p2, q2, q1);
35
36     if(o1!=o2 and o3!=o4)
37         return true;
38
39     if(o1==0 && collinear(p1, p2, q1))
40         return true;
41
42     if(o2==0 && collinear(p1, q2, q1))
43         return true;
44
45     if(o3==0 && collinear(p2, p1, q2))
46         return true;
47
48     if(o4==0 && collinear(p2, q1, q2))
49         return true;
50
51     return false;
52 }

```

### 3.2 Inter-Retangulos

```

1 typedef struct
2 {
3     int x, y;
4 } Point;
5

```

```

6 bool doOverlap(Point l1, Point r1, Point l2, Point r2)
7 {
8     if (l1.x>r2.x or l2.x>r1.x or l1.y<r2.y or l2.y<
9         r1.y)
10         return false;
11     return true;

```

### 3.3 Analytic-Geometry

```

1 struct point
2 {
3     double x, y;
4     point(double _x=0, double _y=0){
5         x=_x;y=_y;
6     }
7
8     void show(){
9         cout << "x = " << x << endl;
10        cout << "y = " << y << endl;
11    }
12
13    point operator+(const point &o) const{
14        return {x + o.x, y + o.y};
15    }
16    point operator-(const point &o) const{
17        return {x - o.x, y - o.y};
18    }
19    bool operator==(const point &o) const{
20        return (x == o.x and y == o.y);
21    }
22
23 };
24
25 struct line
26 {
27     point fp, sp;
28     line(point _fp=0, point _sp=0){
29         fp=_fp;sp=_sp;
30     }
31
32 };
33
34 // Produto Escalar
35 double dot(point a, point b){
36     return a.x*b.x + a.y*b.y;
37 }
38
39 // Produto Vetorial
40 double cross(point a, point b){
41     return a.x*b.y - a.y*b.x;
42 }
43
44 // Dist entre dois pontos
45 double dist(point a, point b){
46     point c = a - b;
47     return sqrt(c.x*c.x + c.y*c.y);
48 }
49
50 // Colinearidade entre 3 pontos
51 bool collinear(point a, point b, point c){
52     return ((c.y - b.y)*(b.x - a.x)==(b.y - a.y)*(c.x
53         -b.x));
54 }
55
56 // Dist entre ponto e reta
57 double distr(point a, line b){
58     double crs = cross(point(a - b.fp), point(b.sp -
59         b.fp));
60     return abs(crs/dist(b.fp, b.sp));

```

```

61 // Area de um poligono (pontos ordenados por
62     adjacencia)
63 double area(vector <point> p){
64     double ret = 0;
65     for(int i=2;i<(int)p.size();i++)
66         ret += cross(p[i] - p[0], p[i-1] - p[0])/2;
67 }
68 // Concavo ou Convexo
69 double ccw(point a, point b, point c){
70     double ret = cross(b - a, c - b);
71     return ret < 0;
72 }

```

## 4 ED

### 4.1 Range-query-bigger-than-k-BIT

```

1 // C++ program to print the number of elements
2 // greater than k in a subarray of range L-R.
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 // Structure which will store both
7 // array elements and queries.
8 struct node {
9     int pos;
10    int l;
11    int r;
12    int val;
13 };
14
15 // Boolean comparator that will be used
16 // for sorting the structural array.
17 bool comp(node a, node b)
18 {
19     // If 2 values are equal the query will
20     // occur first then array element
21     if (a.val == b.val)
22         return a.l > b.l;
23
24     // Otherwise sorted in descending order.
25     return a.val > b.val;
26 }
27
28 // Updates the node of BIT array by adding
29 // 1 to it and its ancestors.
30 void update(int* BIT, int n, int idx)
31 {
32     while (idx <= n) {
33         BIT[idx]++;
34         idx += idx & (-idx);
35     }
36 }
37
38 // Returns the count of numbers of elements
39 // present from starting till idx.
40 int query(int* BIT, int idx)
41 {
42     int ans = 0;
43     while (idx) {
44         ans += BIT[idx];
45         idx -= idx & (-idx);
46     }
47     return ans;
48 }
49
50 // Function to solve the queries offline
51 void solveQuery(int arr[], int n, int QueryL[],
52     int QueryR[], int QueryK[], int q)
53 {
54     // create node to store the elements

```

```

55 // and the queries
56 node a[n + q + 1];
57 // 1-based indexing.
58
59 // traverse for all array numbers
60 for (int i = 1; i <= n; ++i) {
61     a[i].val = arr[i - 1];
62     a[i].pos = 0;
63     a[i].l = 0;
64     a[i].r = i;
65 }
66
67 // iterate for all queries
68 for (int i = n + 1; i <= n + q; ++i) {
69     a[i].pos = i - n;
70     a[i].val = QueryK[i - n - 1];
71     a[i].l = QueryL[i - n - 1];
72     a[i].r = QueryR[i - n - 1];
73 }
74
75 // In-built sort function used to
76 // sort node array using comp function.
77 sort(a + 1, a + n + q + 1, comp);
78
79 // Binary Indexed tree with
80 // initially 0 at all places.
81 int BIT[n + 1];
82
83 // initially 0
84 memset(BIT, 0, sizeof(BIT));
85
86 // For storing answers for each query( 1-based
87 // indexing ).
88 int ans[q + 1];
89
90 // traverse for numbers and query
91 for (int i = 1; i <= n + q; ++i) {
92     if (a[i].pos != 0) {
93         // call function to returns answer for
94         // each query
95         int cnt = query(BIT, a[i].r) - query(BIT,
96             a[i].l - 1);
97
98         // This will ensure that answer of each
99         // query
100         // are stored in order it was initially
101         // asked.
102         ans[a[i].pos] = cnt;
103     }
104     else {
105         // a[i].r contains the position of the
106         // element in the original array.
107         update(BIT, n, a[i].r);
108     }
109 }
110 }
111
112 // Driver Code
113 int main()
114 {
115     int arr[] = { 7, 3, 9, 13, 5, 4 };
116     int n = sizeof(arr) / sizeof(arr[0]);
117
118     // 1-based indexing
119     int QueryL[] = { 1, 2 };
120     int QueryR[] = { 4, 6 };
121
122     // k for each query

```

```

123     int QueryK[] = { 6, 8 };
124
125     // number of queries
126     int q = sizeof(QueryL) / sizeof(QueryL[0]);
127
128     // Function call to get
129     solveQuery(arr, n, QueryL, QueryR, QueryK, q);
130
131     return 0;
132 }

```

## 4.2 Iterative-SegTree

```

1 // Segment Tree Iterativa - Range maximum query
2
3 #define N 100010
4
5 struct Segtree
6 {
7     int t[2*N]={0};
8
9     void build()
10    {
11        for(int i=N-1; i>0; i--)
12            t[i]=max(t[i<<1], t[1<<1|1]);
13    }
14
15    int query(int l, int r)
16    {
17        int ans=0;
18        for(i+=N, r+=N; l<r; l>>=1, r>>=1)
19        {
20            if(l&1)
21                ans=max(ans, t[l++]);
22            if(r&1)
23                ans=max(ans, t[--r]);
24        }
25
26        return ans;
27    }
28
29    void update(int p, int value)
30    {
31        for(t[p+=N]=value; p>1; p>>=1)
32            t[p>>1]= max(t[p], t[p^1]);
33    }
34
35 };
36
37 int main()
38 {
39     Segtree st;
40
41     for(int i=0;i<n;i++)
42     {
43         cin >> aux;
44         st.t[N+i]=aux; //Leaves are stored in
45         // continuous nodes with indices starting with N
46     }
47
48     st.build();
49     x = st.query(inicio, fim);
50     st.update(ind, value);
51 }

```

## 4.3 Recursive-SegTree

```

1 // Segment Tree Recursiva - Range maximum query
2
3 vector<int> val(MAX, 0);
4 vector<int> vet(N);

```

```

5
6 void monta(int i, int j, int no)
7 {
8     if(i==j)
9     {
10         val[no]=vet[i];
11         return;
12     }
13
14     int esq = 2*no;
15     int dir = 2*no+1;
16     int meio = (i+j)/2;
17
18     monta(i, meio, esq);
19     monta(meio+1, j, dir);
20
21     val[no]=max(val[esq], val[dir]);
22 }
23
24 void atualiza(int no, int i, int j, int pos, int
    novo_valor)
25 {
26     if(i==j)
27     {
28         val[no]=novo_valor;
29     }else
30     {
31         int esq = 2*no;
32         int dir = 2*no+1;
33         int meio = (i+j)/2;
34
35         if(pos<=meio)
36             atualiza(esq, i, meio, pos, novo_valor);
37         else
38             atualiza(dir, meio+1, j, pos, novo_valor)
39
40         ;
41
42         if(val[esq]>val[dir])
43             val[no]=val[esq];
44         else
45             val[no]=val[dir];
46     }
47 }
48
49 int consulta(int no, int i, int j, int A, int B)
50 {
51     if(i>B || j<A)
52         return -1;
53     if(i>=A and j<=B)
54         return val[no];
55
56     int esq = 2*no;
57     int dir = 2*no+1;
58     int meio = (i+j)/2;
59
60     int resp_esq = consulta(esq, i, meio, A, B);
61     int resp_dir = consulta(dir, meio+1, j, A, B);
62
63     if(resp_dir!=-1)
64         return resp_esq;
65     if(resp_esq!=-1)
66         return resp_dir;
67
68     if(resp_esq>resp_dir)
69         return resp_esq;
70     else
71         return resp_dir;
72 }
73
74 int main()
75 {
76     monta(1, N, 1);
77     atualiza(1, 1, N, pos, valor);

```

```

76     x = consulta(1, 1, N, inicio, fim);
77
78 }

```

## 4.4 Delta-Encoding

```

1 // Delta encoding
2
3 for(int i=0;i<q;i++)
4 {
5     int l,r,x;
6     cin >> l >> r >> x;
7     delta[l] += x;
8     delta[r+1] -= x;
9 }
10
11 int atual = 0;
12
13 for(int i=0;i<n;i++)
14 {
15     atual += delta[i];
16     v[i] += atual;
17 }

```

## 4.5 Seg-Tree-Farao

```

1 typedef struct
2 {
3     pii prefix, sufix, total, maximo;
4 } no;
5
6 int noleft[MAX], noright[MAX]; //Guarda os valores
    dos nos para que nao sejam calculados novamente
    nas queries
7 int v[MAX];
8 no arvore[MAX];
9
10 pii somar(pii a, pii b) // une pairs
11 {
12     return mp(a.f+b.f, a.s+b.s);
13 }
14
15 no une(no l, no r)
16 {
17     if(l.total.s==0)
18         return r;
19     if(r.total.s==0)
20         return l;
21
22     no m;
23
24     m.prefix = max(l.prefix, somar(l.total, r.prefix)
    ); //prefixo
25     m.sufix = max(r.sufix, somar(r.total, l.sufix));
    //sufixo
26     m.total = somar(l.total, r.total); //Soma de
    todos os elementos da subarvore
27     m.maximo = max(max(l.maximo, r.maximo), somar(l.
    sufix, r.prefix)); //Resultado para cada
    subarvore
28
29     return m;
30 }
31
32 no makenozero()
33 {
34     no m;
35     m.prefix=m.sufix=m.total=m.maximo=mp(0,0);
36     return m;
37 }
38
39 no makeno(int k)

```

```

40 {
41     no m;
42     m.prefix=m.sufix=m.total=m.maximo=mp(k,1);
43     return m;
44 }
45
46 void monta(int n)
47 {
48     if(noleft[n]==noright[n])
49     {
50         arvore[n]=makeno(v[noleft[n]]);
51         return;
52     }
53
54     int mid = (noleft[n]+noright[n])/2;
55     noleft[2*n]=noleft[n]; noright[2*n]=mid;
56     noleft[2*n+1]=mid+1; noright[2*n+1]=noright[n];
57
58     monta(2*n);
59     monta(2*n+1);
60
61     arvore[n]=une(arvore[2*n], arvore[2*n+1]);
62 }
63
64 no busca(int n, int esq, int dir)
65 {
66     if(noleft[n]>=esq and noright[n]<=dir)
67         return arvore[n];
68     if(noright[n]<esq or noleft[n]>dir)
69         return makenozero();
70
71     return une(busca(2*n, esq, dir), busca(2*n+1, esq,
72         dir));
73 }
74
75 int main()
76 {
77     int T, N, Q, A, B;
78     no aux;
79
80     scanf("%d", &T);
81
82     while(T--)
83     {
84         scanf("%d", &N);
85         for(int i=1; i<=N; i++)
86             scanf("%d", &v[i]); //Elementos da arvore
87
88         noleft[1]=1; noright[1]=N;
89         monta(1);
90
91         cin >> Q;
92         while(Q--)
93         {
94             scanf("%d%d", &A, &B); //Intervalo da
95             query
96             aux = busca(1, A, B);
97             printf("%d %d\n", aux.maximo.f, aux.
98                 maximo.s);
99         }
100
101     return 0;
102 }

```

## 4.6 BIT-2D

```

1 // BIT 2D
2
3 int bit[MAX][MAX];
4
5 int sum(int x, int y)

```

```

6 {
7     int resp=0;
8
9     for(int i=x; i>0; i-=i&-i)
10         for(int j=y; j>0; j-=j&-j)
11             resp+=bit[i][j];
12
13     return resp;
14 }
15
16 void update(int x, int y, int delta)
17 {
18     for(int i=x; i<MAX; i+=i&-i)
19         for(int j=y; j<MAX; j+=j&-j)
20             bit[i][j]+=delta;
21 }
22
23 int query(int x1, y1, x2, y2)
24 {
25     return sum(x2,y2) - sum(x2,y1) - sum(x1,y2) + sum
26         (x1,y1);
27 }

```

## 4.7 BIT

```

1 // (BIT) Fenwick Tree
2
3 int N, bit[MAX];
4
5 int soma(int x)
6 {
7     int resp=0;
8
9     // for(int i=x; i>0; i-=i&-i)
10     //     resp+=bit[i];
11
12     while(x > 0)
13     {
14         resp += bit[x];
15         x -= (x & -x);
16     }
17
18     return resp;
19 }
20
21 int query(int L, int R)
22 {
23     return soma(R)-soma(L-1);
24 }
25
26 void update(int x, int v) // add v in x
27 {
28     // for(; x<=n; x+=x&-x)
29     //     bit[x] += v;
30
31     while(x <= N)
32     {
33         bit[x] += v;
34         x += (x & -x);
35     }
36 }

```

## 4.8 Sparse-Table

```

1 // Precompute log2
2 int logv[MAXN+1];
3 logv[1] = 0;
4 for (int i = 2; i <= MAXN; i++)
5     logv[i] = logv[i/2] + 1;
6
7 int st[MAXN][K];
8

```



```

9 void precompute(int N)
10 {
11     for (int i = 0; i < N; i++)
12         st[i][0] = array[i];
13
14     int k = logv[N];
15     for (int j = 1; j <= k; j++)
16         for (int i = 0; i + (1 << j) <= N; i++)
17             st[i][j] = max(st[i][j-1], st[i + (1 << (
18 j - 1))][j - 1]);
19
20 int query(int L, int R)
21 {
22     int j = logv[R - L + 1];
23     int minimum = min(st[L][j], st[R - (1 << j) + 1][
24 j]);
25 }

```

## 4.9 Union-Find

```

1 // Union-Find Functions
2
3 int pai[MAX], peso[MAX];
4
5 int find(int aux)
6 {
7     if(pai[aux]==aux)
8         return aux;
9     else
10         return pai[aux]=find(pai[aux], pai);
11 }
12
13 void join(int x, int y)
14 {
15     x = find(x);
16     y = find(y);
17
18     if(pesos[x]<pesos[y])
19         pai[x] = y;
20     else if(pesos[x]>pesos[y])
21         pai[y] = x;
22     else if(pesos[x]==pesos[y])
23     {
24         pai[x] = y;
25         pesos[y]++;
26     }
27 }
28
29 int main()
30 {
31     for(int i=1;i<=N;i++)
32         pai[i]=i;
33 }

```

## 5 Math

### 5.1 Linear-Diophantine-Equation

```

1 // Linear Diophantine Equation
2 int gcd(int a, int b, int &x, int &y)
3 {
4     if (a == 0)
5     {
6         x = 0; y = 1;
7         return b;
8     }
9     int x1, y1;
10    int d = gcd(b%a, a, x1, y1);
11    x = y1 - (b / a) * x1;
12    y = x1;

```

```

13    return d;
14 }
15
16 bool find_any_solution(int a, int b, int c, int &x0,
17 int &y0, int &g)
18 {
19     g = gcd(abs(a), abs(b), x0, y0);
20     if (c % g)
21         return false;
22
23     x0 *= c / g;
24     y0 *= c / g;
25     if (a < 0) x0 = -x0;
26     if (b < 0) y0 = -y0;
27     return true;
28
29 // All solutions
30 // x = x0 + k*b/g
31 // y = y0 - k*a/g

```

### 5.2 Factorization-sqrt

```

1 // Factorization of a number in sqrt(n)
2
3 int main()
4 {
5     ll N;
6     vector<int> div;
7
8     cin >> N;
9
10    for(ll i=2;i*i<=N;i++)
11    {
12        if(N%i==0)
13        {
14            vet.pb(i);
15            while(N%i==0)
16                N/=i;
17        }
18    }
19    if(N!=1)
20        vet.pb(N);
21
22    return 0;
23 }

```

### 5.3 Modular-Exponentiation

```

1 // Modular exponentiaion - (x^y)%mod in O(log y)
2 ll power(ll x, ll y, ll mod)
3 {
4     ll res = 1;
5     x%=mod;
6
7     while(y)
8     {
9         if(y&1)
10             res=(res*x)%mod;
11
12         y=y>>1;
13         x=(x*x)%mod;
14     }
15     return res;
16 }

```

### 5.4 Miller-Habin

```

1 #include <bits/stdc++.h>
2 #define mod 1000000007
3 #define Pi 3.14159265358979311599796346854
4 #define INF 0x3f3f3f3f

```

```

5  #define MAX 1000010
6  #define f first
7  #define s second
8  #define ll long long
9  #define pb push_back
10 #define mp make_pair
11 #define pii pair<int, int>
12 #define vi vector<int>
13 #define vii vector<pii>
14 #define sws ios_base::sync_with_stdio(false);cin.tie(
15     NULL)
16 #define forn(i, n) for(int i=0; i<(int)(n); i++)
17 #define mdc(a, b) (__gcd((a), (b)))
18 #define mmc(a, b) (((a)/__gcd(a, b)) * b)
19 #define endl '\n'
20 #define teto(a, b) (a+b-1)/b
21
22 using namespace std;
23
24 ll llrand()
25 {
26     ll tmp = rand();
27     return (tmp << 31) | rand();
28 }
29
30 ll add(ll a, ll b, ll c)
31 {
32     return (a + b)%c;
33 }
34
35 ll mul(ll a, ll b, ll c)
36 {
37     ll ans = 0;
38     while(b)
39     {
40         if(b & 1)
41             ans = add(ans, a, c);
42         a = add(a, a, c);
43         b /= 2;
44     }
45     return ans;
46 }
47
48 ll rho(ll n)
49 {
50     ll x, c, y, d, k;
51     int i;
52     do{
53         i = 1;
54         x = llrand()%n;
55         c = llrand()%n;
56         y = x, k = 4;
57         do{
58             if(++i == k)
59             {
60                 y = x;
61                 k *= 2;
62             }
63             x = add(mul(x, x, n), c, n);
64             d = __gcd(abs(x - y), n);
65         } while(d == 1);
66     } while(d == n);
67     return d;
68 }
69
70 ll fexp(ll a, ll b, ll c)
71 {
72     ll ans = 1;
73     while(b)
74     {
75         if(b & 1)
76             ans = mul(ans, a, c);
77         a = mul(a, a, c);
78         b /= 2;
79     }
80     return ans;
81 }
82
83 bool rabin(ll n)
84 {
85     if(n <= 1)
86         return 1;
87     if(n <= 3)
88         return 1;
89     ll s=0, d=n-1;
90     while(d%2==0)
91     {
92         d/=2;
93         s++;
94     }
95     for(int k = 0; k < 64*4; k++)
96     {
97         ll a = (llrand()%(n - 3)) + 2;
98         ll x = fexp(a, d, n);
99         if(x != 1 and x != n-1)
100         {
101             for(int r = 1; r < s; r++)
102             {
103                 x = mul(x, x, n);
104                 if(x == 1)
105                     return 0;
106                 if(x == n-1)
107                     break;
108             }
109             if(x != n-1)
110                 return 0;
111         }
112     }
113     return 1;
114 }
115
116 int main()
117 {
118     //sws;
119     //freopen("input.txt", "r", stdin);
120     //freopen("output.txt", "w", stdout);
121
122     ll N, resp;
123     vector<ll> div;
124
125     cin >> N;
126     resp = N;
127
128     while(N>1 and !rabin(N))
129     {
130         ll d = rho(N);
131         if(!rabin(d))
132             continue;
133         div.pb(d);
134         while(N%d==0)
135             N/=d;
136     }
137     if(N!=resp and N!=1)
138         div.pb(N);
139
140     if(div.empty())
141         cout << resp << endl;
142     else

```

```

150     {
151         for(int i=0;i<(int)div.size();i++)
152             resp = __gcd(resp, div[i]);
153     }
154     cout << resp << endl;
155 }
156
157 return 0;
158
159 }

```

## 5.5 Inverso-Mult

```

1 // ax + my = 1, e gcd(a, m) = 1 para existir solucao
2 // outra forma de escrever: a*x = 1 (mod m)
3 int x, y;
4 int g = gcd(a, m, x, y);
5 if (g != 1)
6     cout << "No solution!";
7 else
8 {
9     x = (x%m + m) % m;
10    cout << x << endl;
11 }

```

## 5.6 Pollard-Rho

```

1 // Pollard Rho Algorithm
2
3 #include <bits/stdc++.h>
4 #define ll long long
5
6 using namespace std;
7
8 ll llrand()
9 {
10     ll tmp = rand();
11     return (tmp << 31) | rand();
12 }
13
14 ll add(ll a, ll b, ll c)
15 {
16     return (a + b)%c;
17 }
18
19 ll mul(ll a, ll b, ll c)
20 {
21     ll ans = 0;
22     while(b)
23     {
24         if(b & 1)
25             ans = add(ans, a, c);
26         a = add(a, a, c);
27         b /= 2;
28     }
29     return ans;
30 }
31
32 ll rho(ll n)
33 {
34     ll x, c, y, d, k;
35     int i;
36     do{
37         i = 1;
38         x = llrand()%n;
39         c = llrand()%n;
40         y = x, k = 4;
41         do{
42             if(++i == k)
43             {
44                 y = x;
45                 k *= 2;

```

```

46             }
47             x = add(mul(x, x, n), c, n);
48             d = __gcd(abs(x - y), n);
49         }
50         while(d == 1);
51     }
52     while(d == n);
53
54     return d;
55 }
56
57 int main()
58 {
59     srand(time(0));
60
61     ll N;
62     cin >> N;
63
64     ll div = rho(N);
65     cout << div << " " << N/div << endl;
66
67     // Finding all divisors
68
69     vector<ll> div;
70
71     while(N>1 and !rabin(N))
72     {
73         ll d = rho(N);
74         if(!rabin(d))
75             continue;
76         div.pb(d);
77         while(N%d==0)
78             N/=d;
79     }
80     if(N!=resp and N!=1)
81         div.pb(N);
82
83     return 0;
84 }
85
86 }

```

## 5.7 Verif-primo

```

1 // prime verification sqrt(N)
2
3 bool eh_primo(long long N)
4 {
5     if(N==2)
6         return true;
7     else if(N==1 or N%2==0)
8         return false;
9     for(long long i=3;i*i<=N;i+=2)
10         if(N%i==0)
11             return false;
12     return true;
13 }

```

## 5.8 Crivo

```

1 // Sieve of Eratosthenes
2
3 int N;
4 vector<bool> primos(100010, true);
5 cin >> N;
6
7 primos[0]=false;
8 primos[1]=false;
9
10 for(int i=2;i<=N;i++)
11     if(primos[i])
12         for(int j=i+i; j<=N; j+=i)

```

```
13         primos[j]=false;
```

## 5.9 Formulas

```
1 int sum_x2(11 N)
2 {
3     return (2*N*N*N + 3*N*N + N)/6;
4 }
```

## 5.10 FFT-golfbot

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int N = (1<<19);
6 const double two_pi = 4 * acos(0);
7
8 struct cpx
9 {
10     cpx(){}
11     cpx(double aa): a(aa){}
12     cpx(double aa,double bb):a(aa),b(bb){}
13     double a;
14     double b;
15     double modsq(void) const
16     {
17         return a*a+b*b;
18     }
19     cpx bar(void) const
20     {
21         return cpx(a,-b);
22     }
23 };
24
25 cpx b[N+100];
26 cpx c[N+100];
27 cpx B[N+100];
28 cpx C[N+100];
29 int a[N+100];
30 int x[N+100];
31 double coss[N+100], sins[N+100];
32 int n,m,p;
33
34 cpx operator +(cpx a,cpx b)
35 {
36     return cpx(a.a+b.a,a.b+b.b);
37 }
38
39 cpx operator *(cpx a,cpx b)
40 {
41     return cpx(a.a*b.a-a.b*b.b,a.a*b.b+a.b*b.a);
42 }
43
44 cpx operator /(cpx a,cpx b)
45 {
46     cpx r = a*b.bar();
47     return cpx(r.a/b.modsq(),r.b/b.modsq());
48 }
49
50 cpx EXP(int i,int dir)
51 {
52     return cpx(coss[i],sins[i]*dir);
53 }
54
55 void FFT(cpx *in,cpx *out,int step,int size,int dir)
56 {
57     if(size<1) return;
58     if(size==1)
59     {
60         out[0]=in[0];
61         return;
```

```
62     }
63     FFT(in,out,step*2,size/2,dir);
64     FFT(in+step,out+size/2,step*2,size/2,dir);
65     for(int i=0;i<size/2;++i)
66     {
67         cpx even=out[i];
68         cpx odd=out[i+size/2];
69         out[i] = even+EXP(i*step,dir)*odd;
70         out[i+size/2]=even+EXP((i+size/2)*step,dir)*
71         odd;
72     }
73
74 int main()
75 {
76     for(int i=0;i<=N;++i)
77     {
78         coss[i]=cos(two_pi*i/N);
79         sins[i]=sin(two_pi*i/N);
80     }
81     while(cin >> n) // Numero de tacadas possiveis
82     {
83         fill(x,x+N+100,0);
84         fill(a,a+N+100,0);
85         for(int i=0;i<n;++i)
86         {
87             cin >> p; // Distancia das tacadas
88             x[p]=1;
89         }
90         for(int i=0;i<N+100;++i)
91         {
92             b[i]=cpx(x[i],0);
93         }
94         cin >> m; // Querys
95         for(int i=0;i<m;++i)
96         {
97             cin >> a[i]; // Distancia da query
98         }
99         FFT(b,B,1,N,1);
100         for(int i=0;i<N;++i)
101             C[i]=B[i]*B[i];
102         FFT(C,c,1,N,-1);
103         for(int i=0;i<N;++i)
104             c[i]=c[i]/N;
105         int cnt=0;
106         for(int i=0;i<m;++i)
107             if(c[a[i]].a>0.5 || x[a[i]])
108                 cnt++;
109         cout << cnt << endl;
110     }
111     return 0;
112 }
```

## 5.11 Modular-Factorial

```
1 // C++ program to compute n! % p using Wilson's
   Theorem
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int power(int x, unsigned int y, int p)
6 {
7     int res = 1;
8     x = x % p;
9
10    while(y > 0)
11    {
12        if(y & 1)
13            res = (res * x) % p;
14
15        y = y >> 1;
16        x = (x * x) % p;
17    }
```

```

18     return res;
19 }
20
21 int modInverse(int a, int p)
22 {
23     return power(a, p-2, p);
24 }
25
26 int modFact(int n, int p)
27 {
28     if (p <= n)
29         return 0;
30
31     int res = (p - 1);
32
33     for(int i = n + 1; i < p; i++)
34         res = (res * modInverse(i, p)) % p;
35     return res;
36 }
37
38 int main()
39 {
40     int n = 25, p = 29;
41     cout << modFact(n, p);
42     return 0;
43 }

```

## 5.12 Kamenetsky

```

1 // Number of digits in n! 0(1)
2
3 #define Pi 3.14159265358979311599796346854
4 #define Eul 2.71828182845904509079559829842
5
6 long long findDigits(int n)
7 {
8     double x;
9
10    if (n < 0)
11        return 0;
12    if (n == 1)
13        return 1;
14
15    x = ((n * log10(n / euler) + log10(2 * Pi * n)
16         / 2.0));
17
18    return floor(x) + 1;
19 }

```

## 6 Misc

### 6.1 LIS

```

1 multiset<int> S;
2 for(int i = 0; i < n; i++){
3     auto it = S.upper_bound(vet[i]); // low for inc
4     if(it != S.end())
5         S.erase(it);
6     S.insert(vet[i]);
7 }
8 // size of the lis
9 int ans = S.size();

```

### 6.2 Bitwise

```

1 // Bitwise
2
3 unsigned char a = 5, b = 9; // a = (00000101), b
4                               = (00001001)

```

```

5 AND -          a&b    // The result is 00000001
6                               (1)
7 OR -           a|b    // The result is 00001101
8                               (13)
9 XOR -          a^b    // The result is 00001100
10                              (12)
11 NOT -          ~a     // The result is 11111010
12                              (250)
13 Left shift -   b<<1   // The result is 00010010
14                              (18)
15 Right shift -  b>>1   // The result is 00000100
16                              (4)
17
18 // Exchange two int variables
19
20     a^=b;
21     b^=a;
22     a^=b;
23
24 // Even or Odd
25
26     (x & 1)? printf("Odd"): printf("Even");
27
28 // Turn on the j-th bit
29
30     int S = 34; //(100010)
31     int j = 3;
32
33     S = S | (1<<j);
34
35 // Turn off the j-th bit
36
37     int S = 42; //(101010)
38     int j = 1;
39
40     S &= ~(1<<j)
41
42     S == 40 //(101000)
43
44 // Check the j-th element
45
46     int S = 42; //(101010)
47     int j = 3;
48
49     T = S & (1<<j); // T = 0
50
51 // Least significant bit (lsb)
52
53     int lsb(int x)
54     {
55         return x&-x;
56     }
57
58 // Exchange o j-th element
59
60     S ^= (1<<j)
61
62 // Position of the first bit on
63
64     T = (S & (-S))
65     T -> 4 bit ligado //(1000)
66
67 // Most significant digit of N
68
69     double K = log10(N);
70     K = K - floor(K);
71     int X = pow(10, K);

```

```

72 // Power of two
73
74 bool isPowerOfTwo(int x)
75 {
76     return x && (!(x&(x-1)));
77 }

```

## 7 Strings

### 7.1 KMP

```

1 vector<int> prefix_function(const string &s){
2     int n = s.size(); vector<int> b(n+1);
3     b[0] = -1; int i = 0, j = -1;
4     while(i < n){
5         while(j >= 0 && s[i] != s[j]) j = b[j];
6         b[++i] = ++j;
7     }
8     return b;
9 }
10 void kmp(const string &t, const string &p){
11     vector<int> b = prefix_function(p);
12     int n = t.size(), m = p.size();
13     int j = 0;
14     for(int i = 0; i < n; i++){
15         while(j >= 0 && t[i] != p[j]) j = b[j];
16         j++;
17         if(j == m){
18             j = b[j];
19         }
20     }
21 }
22 }

```

### 7.2 Pal-int

```

1 bool ehpalindromo(ll n)
2 {
3     if(n<0)
4         return false;
5

```

```

6     int divisor = 1;
7     while(n/divisor >= 10)
8         divisor *= 10;
9
10    while(n != 0)
11    {
12        int leading = n / divisor;
13        int trailing = n % 10;
14
15        if(leading != trailing)
16            return false;
17
18        n = (n % divisor)/10;
19
20        divisor = divisor/100;
21    }
22
23    return true;
24 }

```

### 7.3 Z-Func

```

1 vector<int> z_algo(const string &s)
2 {
3     int n = s.size();
4     int L = 0, R = 0;
5     vector<int> z(n, 0);
6     for(int i = 1; i < n; i++)
7     {
8         if(i <= R)
9             z[i] = min(z[i-L], R - i + 1);
10        while(z[i]+i < n && s[ z[i]+i ] == s[ z[i] ])
11            z[i]++;
12        if(i+z[i]-1 > R)
13        {
14            L = i;
15            R = i + z[i] - 1;
16        }
17    }
18    return z;
19 }

```