



Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos	2	6 Misc	11
1.1 Iterative-BS	2	6.1 Bitwise	11
2 Grafos	2	7 Strings	11
2.1 BFS	2	7.1 KMP	11
2.2 Find-bridges	2		
2.3 Dijkstra	2		
2.4 Floyd-Warshall	3		
2.5 Kruskal	3		
2.6 DFS	3		
2.7 Represent	3		
2.8 Prim	3		
3 Geometria	4		
3.1 Inter-Retas	4		
4 ED	4		
4.1 Iterative-SegTree	4		
4.2 Recursive-SegTree	5		
4.3 Delta-Encoding	5		
4.4 Seg-Tree-Farao	5		
4.5 BIT-2D	6		
4.6 BIT	6		
4.7 Union-Find	7		
5 Math	7		
5.1 Linear-Diophantine-Equation	7		
5.2 Factorization-sqrt	7		
5.3 Modular-Exponentiation	7		
5.4 Miller-Habin	8		
5.5 Pollard-Rho	9		
5.6 Verif-primo	9		
5.7 Crivo	9		
5.8 formulas	9		
5.9 FFT-golfbot	10		
5.10 Modular-Factorial	10		
5.11 Kamenetsky	11		

1 Algoritmos

1.1 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 void BFS(int x)
4 {
5     int atual, v, u;
6     queue<int> fila;
7     fila.push(x);
8
9     componente[x] = valor;
10    atual = 0;
11    while(!fila.empty())
12    {
13        v = fila.front();
14        fila.pop();
15
16        for(int i = 0; i < (int)vizinhos[v].size(); i
17        ++)
18        {
19            u = vizinhos[v][i];
20            if(componente[u] == -1)
21            {
22                componente[u] = componente[v];
23                fila.push(u);
24            }
25        }
26    }
```

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(1);
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;
9 vi distancia;
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(int i=0; i<grafo[aux].size(); i++)
25         {
26             vert=grafo[aux][i].f;
27             dist=grafo[aux][i].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     dist.assign(N+1, INF);
40     grafo.assign(N+1, vii());
41
42     for(int i=0; i<M; i++)
43     {
44         cin >> a >> b >> p;
45         grafo[a].pb(mp(b, p));
46         grafo[b].pb(mp(a, p));
47     }
48 }

```

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 paiy = find(Y);
24     pai[paix]=paiy;
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 INFINITO 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) );

```

```

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

```

4 ED

4.1 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 }

```

4.2 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
25 novo_valor)
26 {
27     if(i==j)
28     {
29         val[no]=novo_valor;
30     }else
31     {
32         int esq = 2*no;
33         int dir = 2*no+1;
34         int meio = (i+j)/2;
35
36         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     if(val[esq]>val[dir])
42         val[no]=val[esq];
43     else
44         val[no]=val[dir];
45 }
46
47 int consulta(int no, int i, int j, int A, int B)
48 {
49     if(i>B || j<A)
50         return -1;
51     if(i>=A and j<=B)
52         return val[no];
53
54     int esq = 2*no;
55     int dir = 2*no+1;
56     int meio = (i+j)/2;
57
58     int resp_esq = consulta(esq, i, meio, A, B);
59     int resp_dir = consulta(dir, meio+1, j, A, B);
60
61     if(resp_dir==-1)
62         return resp_esq;
63     if(resp_esq==-1)
64         return resp_dir;
65
66     if(resp_esq>resp_dir)
67         return resp_esq;
68     else
69         return resp_dir;
70 }
71
72 int main()
73 {
74     monta(1, N, 1);
75     atualiza(1, 1, N, pos, valor);
76     x = consulta(1, 1, N, inicio, fim);
77 }
78 }

```

4.3 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 }

```

4.4 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)
25 ); //prefixo
26     m.sufix = max(r.sufix, somar(r.total, l.sufix));
27     //sufixo
28     m.total = somar(l.total, r.total); //Soma de
29     todos os elementos da subarvore
30     m.maximo = max(max(l.maximo, r.maximo), somar(l.
31 sufix, r.prefix)); //Resultado para cada
32 subarvore
33
34     return m;
35 }
36
37 no makenozero()
38 {
39     no m;
40     m.prefix=m.sufix=m.total=m.maximo=mp(0,0);
41     return m;
42 }
43
44 no makeno(int k)
45 {
46     no m;
47     m.prefix=m.sufix=m.total=m.maximo=mp(k,1);
48     return m;
49 }
50
51 void monta(int n)
52 {
53     if(noleft[n]==noright[n])
54     {
55         arvore[n]=makeno(v[noleft[n]]);
56         return;
57     }
58
59     int mid = (noleft[n]+noright[n])/2;
60     noleft[2*n]=noleft[n]; noright[2*n]=mid;
61     noleft[2*n+1]=mid+1; noright[2*n+1]=noright[n];
62
63     monta(2*n);
64     monta(2*n+1);
65
66     arvore[n]=une(arvore[2*n], arvore[2*n+1]);
67 }
68
69 no busca(int n, int esq, int dir)
70 {
71     if(noleft[n]>=esq and noright[n]<=dir)
72         return arvore[n];
73     if(noright[n]<esq or noleft[n]>dir)
74         return makenozero();
75
76     return une(busca(2*n, esq, dir), busca(2*n+1, esq,

```

```

    dir));
72 }
73
74 int main()
75 {
76     int T, N, Q, A, B;
77     no aux;
78
79     scanf("%d", &T);
80
81     while(T--)
82     {
83         scanf("%d", &N);
84         for(int i=1; i<=N; i++)
85             scanf("%d", &v[i]); //Elementos da arvore
86
87         noleft[1]=1; noright[1]=N;
88         monta(1);
89
90         cin >> Q;
91         while(Q--)
92         {
93             scanf("%d%d", &A, &B); //Intervalo da
94             query
95             aux = busca(1, A, B);
96             printf("%d %d\n", aux.maximo.f, aux.
97                 maximo.s);
98         }
99
100     return 0;
101 }

```

4.5 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 }

```

4.6 BIT

```

1 // (BIT) Fenwick Tree
2
3 int 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, R)
22 {
23     return soma(R)-soma(L);
24 }
25
26 void update(int x, int v)
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 }

```

4.7 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(
    NULL)
15 #define forn(i, n) for(int i=0; i<(int)(n); i++)
16 #define mdc(a, b) (__gcd((a), (b)))
17 #define mmc(a, b) (((a)/__gcd(a, b)) * b)
18 #define endl '\n'
19 #define teto(a, b) (a+b-1)/b
20
21 using namespace std;
22
23 ll llrand()
24 {
25     ll tmp = rand();
26     return (tmp << 31) | rand();
27 }
28
29 ll add(ll a, ll b, ll c)
30 {
31     return (a + b)%c;
32 }
33
34 ll mul(ll a, ll b, ll c)
35 {
36     ll ans = 0;
37     while(b)
38     {
39         if(b & 1)
40             ans = add(ans, a, c);
41         a = add(a, a, c);
42         b /= 2;
43     }
44     return ans;
45 }
46
47 ll rho(ll n)
48 {
49     ll x, c, y, d, k;
50     int i;
51     do{
52         i = 1;
53         x = llrand()%n;
54         c = llrand()%n;
55         y = x, k = 4;
56         do{
57             if(++i == k)
58             {
59                 y = x;
60                 k *= 2;
61             }
62             x = add(mul(x, x, n), c, n);
63             d = __gcd(abs(x - y), n);
64         }

```

```

65         while(d == 1);
66     }
67     while(d == n);
68
69     return d;
70 }
71
72 ll fexp(ll a, ll b, ll c)
73 {
74     ll ans = 1;
75     while(b)
76     {
77         if(b & 1)
78             ans = mul(ans, a, c);
79         a = mul(a, a, c);
80         b /= 2;
81     }
82     return ans;
83 }
84
85 bool rabin(ll n)
86 {
87     if(n <= 1)
88         return 1;
89     if(n <= 3)
90         return 1;
91
92     ll s=0, d=n-1;
93     while(d%2==0)
94     {
95         d/=2;
96         s++;
97     }
98
99     for(int k = 0; k < 64*4; k++)
100     {
101         ll a = (llrand()%(n - 3)) + 2;
102         ll x = fexp(a, d, n);
103         if(x != 1 and x != n-1)
104         {
105             for(int r = 1; r < s; r++)
106             {
107                 x = mul(x, x, n);
108                 if(x == 1)
109                     return 0;
110                 if(x == n-1)
111                     break;
112             }
113             if(x != n-1)
114                 return 0;
115         }
116     }
117     return 1;
118 }
119
120
121
122 int main()
123 {
124     //sws;
125     //freopen("input.txt", "r", stdin);
126     //freopen("output.txt", "w", stdout);
127
128     ll N, resp;
129     vector<ll> div;
130
131     cin >> N;
132     resp = N;
133
134     while(N>1 and !rabin(N))
135     {
136         ll d = rho(N);
137         if(!rabin(d))

```



```

138         continue;
139         div.pb(d);
140         while(N%d==0)
141             N/=d;
142     }
143     if(N!=resp and N!=1)
144         div.pb(N);
145
146
147     if(div.empty())
148         cout << resp << endl;
149     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 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 }

```

5.6 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 }

```

5.7 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;

```

5.8 formulas

```

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

```

5.9 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 }

```

5.10 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 }

```

5.11 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 Bitwise

```

1 // Bitwise
2
3 unsigned char a = 5, b = 9; // a = (00000101), b
4                               = (00001001)
5
6 AND -          a&b    // The result is 00000001
7 (1)
8 OR -           a|b    // The result is 00001101
9 (13)
10 XOR -          a^b    // The result is 00001100
11 (12)
12 NOT -          ~a     // The result is 11111010
13 (250)
14 Left shift -    b<<1  // The result is 00010010
15 (18)
16 Right shift -   b>>1  // The result is 00000100
17 (4)
18
19 // Exchange two int variables
20
21     a^=b;
22     b^=a;
23     a^=b;

```

```

17
18 // Even or Odd
19
20     (x & 1)? printf("Odd"): printf("Even");
21
22 // Turn on the j-th bit
23
24     int S = 34; //(100010)
25     int j = 3;
26
27     S = S | (1<<j);
28
29 // Turn off the j-th bit
30
31     int S = 42; //(101010)
32     int j = 1;
33
34     S &= ~(1<<j)
35
36     S == 40 //(101000)
37
38 // Check the j-th element
39
40     int S = 42; //(101010)
41     int j = 3;
42
43     T = S & (1<<j); // T = 0
44
45 // Exchange o j-th element
46
47     S ^= (1<<j)
48
49 // Position of the first bit on
50
51     T = (S & (-S))
52     T -> 4 bit ligado //(1000)
53
54 // Most significant digit of N
55
56
57     double K = log10(N);
58     K = K - floor(K);
59     int X = pow(10, K);
60
61 // Number of digits in N
62
63     X =floor(log10(N)) + 1;
64
65 // Power of two
66
67     bool isPowerOfTwo(int x)
68     {
69         return x && (!(x&(x-1)));
70     }

```

7 Strings

7.1 KMP

```

1 //KMP Algorithm
2
3 #include <bits/stdc++.h>
4
5 // Fills lps[] for given pattern pat[0..M-1]
6 void computeLPSArray(char* pat, int M, int* lps)
7 {
8     // length of the previous longest prefix suffix
9     int len = 0;
10
11     lps[0] = 0; // lps[0] is always 0
12
13     // the loop calculates lps[i] for i = 1 to M-1

```

```

14     int i = 1;
15     while (i < M) {
16         if (pat[i] == pat[len]) {
17             len++;
18             lps[i] = len;
19             i++;
20         }
21         else // (pat[i] != pat[len])
22         {
23             // This is tricky. Consider the example.
24             // AAACAAAA and i = 7. The idea is
25             similar
26             // to search step.
27             if (len != 0) {
28                 len = lps[len - 1];
29
30                 // Also, note that we do not
31                 increment
32                 // i here
33             }
34             else // if (len == 0)
35             {
36                 lps[i] = 0;
37                 i++;
38             }
39         }
40     }
41     // Prints occurrences of txt[] in pat[]
42     void KMPSearch(char* pat, char* txt)
43     {
44         int M = strlen(pat);
45         int N = strlen(txt);
46
47         // create lps[] that will hold the longest prefix
48         // suffix
49         // values for pattern
50         int lps[M];
51
52         // Preprocess the pattern (calculate lps[] array)
53         computeLPSArray(pat, M, lps);
54
55         int i = 0; // index for txt[]
56         int j = 0; // index for pat[]
57         while (i < N) {
58             if (pat[j] == txt[i]) {
59                 j++;
60                 i++;
61             }
62
63             if (j == M) {
64                 printf("Found pattern at index %d ", i -
65                     j);
66                 j = lps[j - 1];
67             }
68
69             // mismatch after j matches
70             else if (i < N and pat[j] != txt[i]) {
71                 // Do not match lps[0..lps[j-1]]
72                 // characters,
73                 // they will match anyway
74                 if (j != 0)
75                     j = lps[j - 1];
76                 else
77                     i = i + 1;
78             }
79         }
80     }
81
82     // Driver program to test above function
83     int main()
84     {
85         char txt[] = "ABABDABACDABABCABAB";
86         char pat[] = "ABABCABAB";
87         KMPSearch(pat, txt);
88         return 0;
89     }

```