



Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos	2	7 Misc	9
1.1 Recursive-BS	2	7.1 Bitwise	9
2 Grafos	2	7.2 Complexity	10
2.1 BFS	2	7.3 Aprox	10
2.2 Dijkstra	2	8 Strings	10
2.3 Floyd-Warshall	2	8.1 KMP	10
2.4 Kruskal	2		
2.5 DFS	3		
2.6 Represent	3		
3 Geometria	3		
3.1 Inter-Retas	3		
4 ED	3		
4.1 Iterative-SegTree	3		
4.2 Recursive-SegTree	4		
4.3 Delta-Encoding	4		
4.4 Seg-Tree-Farao	5		
4.5 BIT-2D	5		
4.6 BIT	6		
4.7 Union-Find	6		
5 STL	6		
5.1 Pair	6		
5.2 Set	6		
5.3 Stack	7		
5.4 Queue	7		
5.5 Priority-Queue	7		
5.6 Map	7		
5.7 Vector	7		
6 Math	8		
6.1 Modular-Exponentiation	8		
6.2 Verif-primo	8		
6.3 Crivo	8		
6.4 FFT-golfbot	8		
6.5 Kamenetsky	9		

1 Algoritmos

1.1 Recursive-BS

```
1 // Recursive binary search
2
3 int bs(int x, int ini, int fim)
4 {
5     if(fim>=ini)
6     {
7         int meio = (ini+fim)/2;
8
9         if(vetor[mid]==x)
10            return x;
11
12        if(vetor[meio]<x)
13            return bs(x, ini, meio-1);
14        else
15            return bs(x, meio+1, fim);
16    }
17
18    return -1;
19 }
```

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 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 }
```

2.3 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][
11 k] + dist[k][j]);
12 }
```

2.4 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 paix = find(Y);
24     pai[paix]=paix;
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].
39         dist;
40
41     sort(grafo+1, grafo+M+1, comp);
42
43     for(int i=1;i<=M;i++)
44     {
45         if(find(grafo[i].A)!=find(grafo[i].B))
46         {
47             join(grafo[i].A, grafo[i].B);
48             soma+=grafo[i].dist;
49         }
50
51     }
52
53     cout << soma << endl;
54 }

```

2.5 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 }

```

2.6 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);

```

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         return ans;
26     }
27
28     void update(int p, int value)
29     {
30         for(t[p+=n]=value; p>1; p>>=1)
31             t[p>>1]= max(t[p], t[p^1]);
32     }
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)
37             atualiza(esq, i, meio, pos, novo_valor);
38         else
39             atualiza(dir, meio+1, j, pos, novo_valor)
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);
78     x = consulta(1, 1, N, inicio, fim);
79 }

```

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
7                                 dos nos para que nao sejam calculados novamente
8                                 nas querys
9 int v[MAX];
10 no arvore[MAX];
11
12 pii somar(pii a, pii b) // une pairs
13 {
14     return mp(a.f+b.f, a.s+b.s);
15 }
16
17 no une(no l, no r)
18 {
19     if(l.total.s==0)
20         return r;
21     if(r.total.s==0)
22         return l;
23
24     no m;
25
26     m.prefix = max(l.prefix, somar(l.total, r.prefix));
27     m.sufix = max(r.sufix, somar(r.total, l.sufix));
28     m.total = somar(l.total, r.total); //Soma de
29     m.maximo = max(max(l.maximo, r.maximo), somar(l.sufix, r.prefix)); //Resultado para cada
30     m.sufix = max(r.sufix, somar(r.total, l.sufix));
31     m.total = somar(l.total, r.total);
32     return m;
33 }
34
35 no makenozero()
36 {
37     no m;
38     m.prefix=m.sufix=m.total=m.maximo=mp(0,0);
39     return m;
40 }
41
42 no makeno(int k)
43 {
44     no m;
45     m.prefix=m.sufix=m.total=m.maximo=mp(k,1);
46     return m;
47 }
48
49 void monta(int n)
50 {
51     if(noleft[n]==noright[n])
52     {
53         arvore[n]=makeno(v[noleft[n]]);
54         return;
55     }
56
57     int mid = (noleft[n]+noright[n])/2;
58     noleft[2*n]=noleft[n]; noright[2*n]=mid;
59     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;

```

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);

```

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 STL

5.1 Pair

```

1 pair<string, int> P;
2
3 cin>>P.first>>P.second;
4
5 // Pair of pair
6
7     pair<string, pair<double, double>> P;
8
9     P.first = "Joao";
10    P.second.first = 8.2;
11    P.second.second = 10;
12
13 // Vector of pair
14
15    vector<pair<int, string> > V;
16    sort(V.begin(), V.end());
17
18 //make.pair()
19
20    P = make_pair("Joao", 10);
21
22    for(int i=1;i<10;i++)
23    {
24        cin>>a>>b;
25        V.push_back(make_pair(a,b));
26    }

```

5.2 Set

```

1 // Set - Red-Black Trees - O(logn)
2
3 set<int> S;
4
5 //S.insert()
6
7     S.insert(10); // O(logN)
8
9 //S.find()
10
11    if(S.find(3) != S.end())// O(logN)
12
13 //S.erase
14
15    S.erase(10);
16
17 //Outros
18    S.clear();
19    S.size();
20    S.begin();
21    S.end();
22
23    p = S.lower_bound(n); // Retorna um ponteiro para
24    o primeiro elemento maior ou igual a n (not less
25    than n)
26
27    p = S.upper_bound(n); // Retorna um ponteiro para
28    o primeiro elemento maior que n (greater than n)

```

```

27 // (set<int>::iterator)
28
29 for(set<int>::iterator it=S.begin(); it!=S.end();
    it++)
30 {
31     cout << *it << " ";
32 }

```

5.3 Stack

```

1 // Stack
2
3 stack<int> pilha;
4
5 //pilha.push()
6
7     pilha.push(N);
8
9 //pilha.empty()
10
11     if(pilha.empty()==true/false)
12
13 //pilha.pop()
14
15     pilha.pop();
16
17 //pilha.front()
18
19     p = pilha.top();

```

5.4 Queue

```

1 // Queue
2
3 queue<int> fila;
4
5 //fila.push()
6
7     fila.push(N);
8
9 //fila.empty()
10
11     if(fila.empty()==true/false)
12
13 //fila.pop()
14
15     fila.pop();
16
17 //fila.front()
18
19     p = fila.front();
20

```

5.5 Priority-Queue

```

1 // Priority Queue - O(logn)
2
3 priority_queue<int> plista;
4
5 //plista.push()
6
7     plista.push(N);
8
9 //plista.empty()
10
11     if(plista.empty()==true/false)
12
13 //plista.pop()
14
15     plista.pop();
16
17 //plista.front()

```

```

18
19     p = plista.top();

```

5.6 Map

```

1 // Map - Red-Black Trees
2
3 map<string, int> M;
4
5 //S.insert()
6
7     M.insert(make_pair("Tiago", 18));
8     //ou
9     M["Tiago"]=18; // O(logN)
10
11 //S.find()
12
13     if(M.find("Tiago") != M.end()) // O(logN)
14
15         cout << M["Tiago"] << endl;
16
17 //S.erase
18
19     M.erase("Tiago"); // O(logN)
20
21
22 //S.count()
23
24     if(S.count(N))
25
26 //Other
27
28     M.clear();
29     M.size();
30     M.begin();
31     M.end();
32
33 // (map<int>::iterator)
34
35     for(map<string,int>::iterator it=M.begin(); it!=M
        .end(); it++)
36     {
37         cout << "(" << it->first << ", " << it->
            second << ") ";
38     }

```

5.7 Vector

```

1 // Vector - Vetor
2
3 vector<int> V;
4 vector<tipo> nome;
5 vector<tipo> V(n, value);
6
7 //push_back()
8
9     V.push_back(2);
10    V.push_front(2);
11
12 // front() back()
13
14    cout << V.front() << endl;
15    cout << V.back() << endl;
16
17 //size()
18
19    tamanho = V.size();
20
21 //resize()
22
23    V.resize(10);
24    V.resize(n, k);

```

```

25
26 //pop_back()
27
28     V.pop_back();
29
30 //clear()
31
32     V.clear();
33     sort(V.begin(), V.end());
34
35 //upper_bound() e lower_bound()
36
37     vector<int>::iterator low,up;
38     low=lower_bound(v.begin(), v.end(), 20);
39     up=upper_bound(v.begin(), v.end(), 20);
40     cout << "lower_bound at position " << (low- v.
41     begin()) << '\n';
42     cout << "upper_bound at position " << (up - v.
43     begin()) << '\n';
44
45 //binary_search()
46
47     if(binary_search(vet.begin(), vet.end(), 15))
48
49 //accumulate()
50
51     cout << accumulate(first, last, sum, func) <<
52     endl;
53     //first - pointer to the first element
54     //last - last element
55     //sum - inicial value
56     //func
57
58     int func(int x, int y)
59     {
60         //return x*y;
61         return x+y;
62     }
63
64 //partial_sum()
65
66     partial_sum(first, last, vet, func);
67
68     int func(int x, int y)
69     {
70         //return x*y;
71         return x+y;
72     }
73
74 //assign()
75     //Diferente do resize() por mudar o valor de
76     todos os elementos do vector
77
78     vector<int> vet;
79     vet.assign(N, x);
80
81     vector< vector<int> > vet;
82     vet.assign(N, vector<int>());
83
84 //sort()
85
86     sort(vet, vet+N, func);
87
88     bool func(Aluno a, Aluno b)
89     {
90         return a.nota < b.nota; // True caso a venha
91         antes de b, False caso contrario
92     }
93
94 //fill()
95
96     vector<int> vet(5); // 0 0 0 0 0
97

```

```

93     fill(vet.begin(), vet.begin()+2, 8); // 8 8 0 0 0
94

```

6 Math

6.1 Modular-Exponentiation

```

1 // Modular exponentiaion - (x^y)%p in O(log y)
2 int power(int x, unsigned int y, int p)
3 {
4     int res = 1;
5     x%=p;
6
7     while(y>0)
8     {
9         if(y&1)
10             res=(res*x)%p;
11
12         y=y>>1;
13         x=(x*x)%p;
14     }
15 }

```

6.2 Verif-primo

```

1 // prime verrification sqrt(N)
2
3 long long eh_primo(long long N)
4 {
5     if(N==2)
6     {
7         return true;
8     }
9     else if(N==1 or N%2==0)
10    {
11        return false;
12    }
13    for(long long i=3;i*i<=N;i+=2)
14        if(N%i==0)
15            return false;
16
17    return true;
18 }

```

6.3 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;

```

6.4 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)*
odd;
71     }
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 }

```

6.5 Kamenetsky

```

1 // Number of digits in n! O(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 }

```

7 Misc

7.1 Bitwise

```

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

```

```

9 Left shift - b<<1 // The result is 00010010
10 (18)
11 Right shift - b>>1 // The result is 00000100
12 (4)
13 // Exchange two int variables
14 a^=b;
15 b^=a;
16 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 }

```

7.2 Complexity

```

1 // Complexity
2
3 If n <= 12, the time complexity can be O(n!).
4 If n <= 25, the time complexity can be O(2^n).
5 If n <= 100, the time complexity can be O(n^4).
6 If n <= 500, the time complexity can be O(n^3).

```

```

7 If n <= 10^4, the time complexity can be O(n^2).
8 If n <= 10^6, the time complexity can be O(n log
9 n).
10 If n <= 10^8, the time complexity can be O(n).
11 If n > 10^8, the time complexity can be O(log n)
12 or O(1).

```

7.3 Aprox

```

1 // Approximation
2
3 value- round() floor() ceil() trunc()
4 -----
5 +2.3 +2.0 +2.0 +3.0 +2.0
6 +3.8 +4.0 +3.0 +4.0 +3.0
7 +5.5 +6.0 +5.0 +6.0 +5.0
8 -2.3 -2.0 -3.0 -2.0 -2.0
9 -3.8 -4.0 -4.0 -3.0 -3.0
10 -5.5 -6.0 -6.0 -5.0 -5.0

```

8 Strings

8.1 KMP

```

1 //KMP Algorithm
2
3 #include <bits/stdc++.h>
4
5 // Fills lps[] for given patttern 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 }
42
43 // Prints occurrences of txt[] in pat[]
44 void KMPSearch(char* pat, char* txt)
45 {
46     int M = strlen(pat);
47     int N = strlen(txt);

```

```

47 // create lps[] that will hold the longest prefix suffix
48 // values for pattern
49 int lps[M];
50
51 // Preprocess the pattern (calculate lps[] array)
52 computeLPSArray(pat, M, lps);
53
54 int i = 0; // index for txt[]
55 int j = 0; // index for pat[]
56 while (i < N) {
57     if (pat[j] == txt[i]) {
58         j++;
59         i++;
60     }
61
62     if (j == M) {
63         printf("Found pattern at index %d ", i -
64 j);
65         j = lps[j - 1];
66     }
67
68     // mismatch after j matches
69     else if (i < N and pat[j] != txt[i]) {
70         // Do not match lps[0..lps[j-1]]
71         // characters,
72         // they will match anyway
73         if (j != 0)
74             j = lps[j - 1];
75         else
76             i = i + 1;
77     }
78 }
79
80 // Driver program to test above function
81 int main()
82 {
83     char txt[] = "ABABDABACDABABCABAB";
84     char pat[] = "ABABCABAB";
85     KMPSearch(pat, txt);
86     return 0;
87 }

```