



Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos	3	4 ED	9
1.1 Mochila	3	4.1 Trie	9
1.2 Kadane-DP	3	4.2 Range-query-bigger-than-k-BIT	10
1.3 Iterative-BS	3	4.3 Iterative-SegTree	10
2 Grafos	3	4.4 Recursive-SegTree	11
2.1 BFS	3	4.5 Delta-Encoding	11
2.2 Find-bridges	3	4.6 Seg-Tree-Farao	11
2.3 Dijkstra	3	4.7 BIT-2D	12
2.4 LCA	4	4.8 BIT	12
2.5 Floyd-Warshall	4	4.9 Sparse-Table	13
2.6 Kruskal	4	4.10 Union-Find	13
2.7 DFS	5	4.11 Mo	13
2.8 Topological-sort	5	5 Math	14
2.9 Kosaraju	5	5.1 Totient	14
2.10 Centroid	5	5.2 Sqrt-BigInt	14
2.11 Prim	5	5.3 Linear-Diophantine-Equation	14
3 Geometria	6	5.4 Sum-n2	14
3.1 Convex-polygon-intersection	6	5.5 Factorization-sqrt	14
3.2 Inside-polygon	6	5.6 Modular-Exponentiation	14
3.3 Angulo-interno-poligono-regular	6	5.7 Miller-Habin	14
3.4 Pick's-theorem	7	5.8 Inverso-Mult	15
3.5 linesweep	7	5.9 Pollard-Rho	15
3.6 Center-polygon	7	5.10 Verif-primo	16
3.7 Intersect-polygon	7	5.11 Crivo	16
3.8 Sort-by-Angle	7	5.12 Simpson's-formula	16
3.9 Cross-properties	7	5.13 FFT	16
3.10 Inter-Retangulos	7	5.14 Next-Permutation	17
3.11 Heron	7	5.15 Fast-Exponentiation	17
3.12 3D	7	5.16 Recursao-linear	17
3.13 Dot-properties	8	5.17 Raiz-primitiva	18
3.14 Uniao-segmentos	8	5.18 Kamenetsky	18
3.15 Minkowski-Sum	8	6 Misc	18
3.16 2D	8	6.1 2SAT	18
		6.2 LIS	19
		6.3 Bitwise	19
		6.4 Template	20

7	Strings	20
7.1	Trie	20
7.2	KMP	20
7.3	LCS	20
7.4	Pal-int	20
7.5	Z-Func	21
7.6	Hash	21
7.7	Manacher	21

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 Kadane-DP

```
1 // Largest Sum Contiguous Subarray
2
3 int maxSubArraySum(vector<int> a)
4 {
5     int size = a.size();
6     int max_so_far = a[0];
7     int curr_max = a[0];
8
9     for (int i=1;i<size;i++)
10     {
11         curr_max = max(a[i], curr_max+a[i]);
12         max_so_far = max(max_so_far, curr_max);
13     }
14     return max_so_far;
15 }
```

1.3 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 d[s]=0;
11
12 q.push(s);
13 used[s] = true;
14 while (!q.empty()) {
15     int v = q.front();
16     q.pop();
17     for (int u : adj[v]) {
18         if (d[u] > d[v] + 1) {
19             q.push(u);
20             d[u] = d[v] + 1;
21         }
22     }
23 }
```

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     visited[v] = true;
10     t[v] = low[v] = timer++;
11     for(int i=0;i<(int)grafo[v].size();i++){
12         int vert = grafo[v][i];
13         if(vert == p)
14             continue;
15         if(visited[vert])
16             low[v] = min(low[v], t[vert]);
17         else{
18             find_bridges(vert, v);
19             low[v] = min(low[v], low[vert]);
20             if(low[vert] > t[v])
21                 IS_BRIDGE(v, vert);
22         }
23     }
24 }
25
26 int main()
27 {
28     timer = 0;
29     visited.assign(N+1, false);
30     t.assign(N+1, 0);
31     low.assign(N+1, 0);
32
33     for(int i=0;i<N;i++)
34         if(!visited[i])
35             find_bridges(i);
36
37     return 0;
38 }
```

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     int dist, vert, aux;
14     distancia[k]=0;
15
16     fila.push(mp(k, 0));
17
18     while(!fila.empty()){
19         aux=fila.top().f;
20         fila.pop();
21
22         for(auto v: grafo[aux]){
23             vert=v.f;
24             dist=v.s;
25             if(distancia[vert]>distancia[aux]+dist){
26                 distancia[vert]=distancia[aux]+dist;
27                 fila.push(mp(vert, distancia[vert]));
28             }
29         }
30     }
31 }
32
33 int main()
34 {
35     for(int i=0; i<M; i++){
36         cin >> a >> b >> p;
37         grafo[a].pb(mp(b, p));
38         grafo[b].pb(mp(a, p));
39     }
40 }

```

2.4 LCA

```

1 const int K = 100;
2 int logv[MAX+1];
3 int st[MAX][K];
4 vector<vi> grafo(200010, vi());
5
6 void make(){
7     logv[1] = 0; // pre-computar tabela de log
8     for (int i = 2; i <= MAX; i++)
9         logv[i] = logv[i/2] + 1;
10 }
11
12 void precompute(int N, int array[]) { //
13     for (int i = 0; i < N; i++)
14         st[i][0] = array[i];
15
16     int k = logv[N];
17     for (int j = 1; j <= k; j++)
18         for (int i = 0; i + (1 << j) <= N; i++)
19             st[i][j] = min(st[i][j-1], st[i + (1 << (j
20 - 1))][j - 1]);
21 }
22
23 int query(int L, int R) {
24     int j = logv[R - L + 1];
25     int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);
26
27     return minimum;
28 }
29
30 int start[MAX+1], dfs_time;
31 int tour[2*MAX+1], id[2*MAX+1];
32 void dfs(int u, int pai=-1){

```

```

33     start[u] = dfs_time;
34     id[dfs_time] = u;
35     tour[dfs_time++] = start[u];
36     for(int v : grafo[u]){
37         if(v==pai)
38             continue;
39         dfs(v, u);
40         id[dfs_time] = u;
41         tour[dfs_time++] = start[u];
42     }
43 }
44
45 int LCA(int u, int v)
46 {
47     if(start[u] > start[v])
48         swap(u, v);
49     return id[query(start[u], start[v])];
50 }
51
52 int main()
53 {
54     int N, k, a, b;
55     cin >> N;
56
57     for(int i=0;i<N-1;i++){
58         {
59             cin >> a >> b;
60             grafo[a].pb(b);
61             grafo[b].pb(a);
62         }
63     }
64     dfs(1);
65
66     make();
67     precompute(2*N, tour);
68
69     cin >> k;
70     for(int i=0;i<k;i++){
71         {
72             cin >> a >> b;
73             cout << LCA(a, b) << endl;
74         }
75     }
76     return 0;
77 }

```

2.5 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 }

```

2.6 Kruskal

```

1 // deve-se ter dsu codada com as funcoes make_set,
2 find_set e union_sets
3 struct Edge {
4     int u, v, weight;
5     bool operator<(Edge const& other) {
6         return weight < other.weight;
7     }
8 };

```

```

9 int n;
10 vector<Edge> edges;
11
12 int cost = 0;
13 vector<Edge> result;
14 for (int i = 0; i < n; i++)
15     make_set(i);
16
17 sort(edges.begin(), edges.end());
18
19 for (Edge e : edges) {
20     if (find_set(e.u) != find_set(e.v)) {
21         cost += e.weight;
22         result.push_back(e); // vector com as arestas
23         da MST
24         union_sets(e.u, e.v);
25     }
26 }

```

2.7 DFS

```

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

```

2.8 Topological-sort

```

1 vector<vi> grafo(MAX, vi());
2 int grau[MAX]; // Quantas arestas chegam no indice i
3
4 vi topological_sort(int N)
5 {
6     vi resp;
7     for(int i=0; i<N; i++){
8         if(!grau[i])
9             resp.push_back(i);
10
11     int k=0;
12     while(k < (int)resp.size()){
13         int u = resp[k];
14         k++;
15         for(auto v: grafo[u]){
16             grau[v]--;
17             if(!grau[v])
18                 resp.pb(v);
19         }
20     }
21     return resp;
22 }

```

2.9 Kosaraju

```

1 // KOSARAJU - O(V+E) - encontra componentes
2 // g -> grafo, gt -> grafo tempo
3 // vis -> visitado, cor -> componente fortemente
4 // conexo ordenado topologicamente
5 vector<int> g[N], gt[N], S; int vis[N], cor[N];
6 void dfs(int u){
7     vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
8     S.push_back(u);
9 }
10 void dfst(int u, int e){
11     cor[u] = e;

```

```

11     for(int v : gt[u]) if(!cor[v]) dfst(v, e);
12 }
13 void kosaraju(){
14     for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);
15     for(int i = 1; i <= n; i++) for(int j : g[i])
16         gt[j].push_back(i);
17     int e = 0; reverse(S.begin(), S.end());
18     for(int u : S) if(!cor[u]) dfst(u, ++e);
19 }

```

2.10 Centroid

```

1 vi g[MAX];
2 int size[MAX];
3 bool erased[MAX]; // vetor dos vertices apagados na
4 // decomp.
5
6 int sz(int u, int p) {
7     int s = 1;
8     for(auto prox : g[u]) {
9         if(prox != p and !erased[prox])
10             s += sz(prox, u);
11     }
12     return size[u] = s;
13 }
14
15 int centroid(int u, int p, int n) {
16     // chamar funcao sz antes, n = size[u]
17     for(auto prox : g[u]) {
18         if(prox != p and !erased[prox]) {
19             if(size[prox] > n/2) {
20                 return centroid(prox, u, n);
21             }
22         }
23     }
24     return u;
25 }

```

2.11 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] = INFINTO;
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         int davez = -1;
21
22         while(!fila.empty()){
23             int atual = fila.top().second;
24             fila.pop();
25
26             if(!processado[atual]){
27                 davez = atual;
28                 break;
29             }
30         }

```

```

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

```

3 Geometria

3.1 Inside-polygon

```

1 bool inside(vector<point> vet, point ext) //ccw
2 {
3     int l=2, r=(int)vet.size()-1;
4     int res=r;
5     while(l<r){
6         int mid = (l+r)/2;
7         if(esq(vet[0], vet[mid], ext)==-1){
8             l=mid+1;
9         }else
10         {
11             r=mid;
12             res=mid;
13         }
14     }
15     int a = esq(vet[0], vet[res-1], ext);
16     int b = esq(vet[res-1], vet[res], ext);
17     int c = esq(vet[res], vet[0], ext);
18
19     if((a==1 or b==1 or c==1) and (a==-1 or b==-1 or
c==-1)) return false;
20     else return true;
21 }

```

3.2 Angulo-interno-poligono-regular

$$a = \frac{N-2}{N} * 180$$

3.3 Pick's-theorem

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

3.4 linesweep

```

1 typedef pair<double, double> dd;
2
3 bool compare(dd a, dd b){
4     return a.st < b.st;
5 }
6
7 double closest(dd v[], int n){
8     sort(v, v+n, compare);
9     double best = FLT_MAX;
10    set<dd> box;
11    box.insert(v[0]);
12    int left = 0;
13    rep2(i, 1, n){
14        while(left < i && v[i].st-v[left].st > best){
15            box.erase(v[left++]);
16        }
17        for(set<dd>::iterator it = box.lower_bound(mp
(v[i].nd-best, v[i].st-best)); it!=box.end() && v[
i].nd + best >= it->nd; it++){
18            best = min(best, sqrt(pow(v[i].nd - it->
nd, 2.0) + pow(v[i].st - it->st, 2.0)));
19        }
20        box.insert(v[i]);
21    }
22    return best;
23 }

```

3.5 Center-polygon

```

1 point center(vector<point> A)
2 {
3     point centerA = point();
4     for(int i=0; i<(int)A.size(); i++){
5         centerA=centerA+A[i];
6     }
7     return centerA/((int)A.size());

```

3.6 Intersect-polygon

```

1 bool intersect(vector<point> A, vector<point> B) //
Ordered ccw
2 {
3     for(auto a: A)
4         if(inside(B, a))
5             return true;
6     for(auto b: B)
7         if(inside(A, b))
8             return true;
9
10    if(inside(B, center(A)))
11        return true;
12
13    return false;
14 }

```

3.7 Sort-by-Angle

```

1 int quarter(point a)
2 {
3     if(a.x>=0 and a.y>0) return 0;
4     if(a.x<0 and a.y>=0) return 1;
5     if(a.x<=0 and a.y<0) return 2;
6     return 3;
7 }
8
9 point c;
10 bool comp(point a, point b) //ccw
11 {
12     a=a-c;b=b-c;
13     int qa = quarter(a);
14     int qb = quarter(b);
15     if(qa==qb)
16         return cross(a,b)>0;
17     else
18         return qa<qb;
19 }
20
21 c = center(A);
22 sort(A.begin(), A.end(), comp);

```

3.8 Cross-properties

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

3.9 Inter-Retangulos

```

1 bool doOverlap(point l1, point r1, point l2, point r2)
2 {
3     if (l1.x>r2.x or l2.x>r1.x or l1.y<r2.y or l2.y<
4         r1.y)
5         return false;
6     return true;

```

3.10 Heron

$$A_{triangulo} = \sqrt{s(s-a)(s-b)(s-c)}$$

$$A_{quadrilatero} = \sqrt{(s-a)(s-b)(s-c)(s-d)}$$

3.11 3D

```

1 typedef ld cod;
2
3 bool eq(cod a, cod b){ return abs(a - b) <= EPS; }
4
5 struct point
6 {
7     cod x, y, z;
8     point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z) {}
9
10    point operator+(const point &o) const{
11        return {x+o.x, y+o.y, z+o.z};
12    }
13    point operator-(const point &o) const{
14        return {x-o.x, y-o.y, z-o.z};
15    }
16    point operator*(cod t) const{
17        return {x*t, y*t, z*t};

```

```

18    }
19    point operator/(cod t) const{
20        return {x/t, y/t, z/t};
21    }
22    bool operator==(const point &o) const{
23        return eq(x, o.x) and eq(y, o.y) and eq(z, o.
24        z);
25    }
26 };
27 // Produto Escalar
28 cod dot(point a, point b){
29     return a.x*b.x + a.y*b.y + a.z*b.z;
30 }
31 // Produto Vetorial
32 point cross(point a, point b){
33     return point(a.y*b.z - a.z*b.y,
34                 a.z*b.x - a.x*b.z,
35                 a.x*b.y - a.y*b.x);
36 }
37
38 ld abs(point a){ // Modulo
39     return sqrt(dot(a, a));
40 }
41
42 ld proj(point a, point b){ // a sobre b
43     return dot(a, b)/abs(b);
44 }
45 ld angle(point a, point b){ // em radianos
46     return acos(dot(a, b) / abs(a) / abs(b));
47 }
48
49 cod triple(point a, point b, point c){
50     return dot(a, cross(b, c)); // Area do
51     paralelepipedo

```

3.12 Dot-properties

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

3.13 Uniao-segmentos

```

1 int length_union(const vector<pii> &a){
2     int n = a.size();
3     vector<pair<int, bool>> x(n*2);
4     for(int i = 0; i < n; i++){
5         x[i*2] = {a[i].ff, false};
6         x[i*2+1] = {a[i].ss, true};
7     }
8
9     sort(x.begin(), x.end());
10
11    int result=0;
12    int c=0;
13    for(int i=0;i<2*n;i++){
14        if(i and c and x[i].ff>x[i-1].ff)
15            result += x[i].ff-x[i-1].ff;
16
17        if(x[i].ss) c--;
18        else c++;
19    }

```

```

20     return result;
21 }

```

3.14 Minkowski-Sum

```

1 vector<point> mk(const vector<point> &a, const vector<
    point> &b){
2     int i = 0, j = 0;
3     for(int k = 0; k < (int)a.size(); k++){ if(a[k]<a[i
    ])
4         i = k;
5     for(int k = 0; k < (int)b.size(); k++){ if(b[k]<b[j
    ])
6         j = k;
7
8     vector<point> c;
9     c.reserve(a.size() + b.size());
10    for(int k = 0; k < int(a.size()+b.size()); k++){
11        point pt{a[i] + b[j]};
12        if((int)c.size() >= 2 and !ccw(c[c.size()-2],
    c.back(), pt))
13            c.pop_back();
14        c.pb(pt);
15        int q = i+1, w = j+1;
16        if(q == int(a.size())) q = 0;
17        if(w == int(b.size())) w = 0;
18        if(ccw(c.back(), a[i]+b[w], a[q]+b[j]) < 0) i
    = q;
19        else j = w;
20    }
21
22    if(!ccw(c[0], c[(int)c.size()-1], c[(int)c.size()
    -2]))
23        c.pop_back();
24    if(!ccw(c.back(), c[0], c[1])){
25        c[0]=c.back();
26        c.pop_back();
27    }
28    c.shrink_to_fit();
29
30    return c;
31 }

```

3.15 2D

```

1 ld max(ld a, ld b){ return(a>b ? a:b);}
2 ld min(ld a, ld b){ return(a<b ? a:b);}
3
4 typedef ld cod;
5 bool eq(cod a, cod b){ return fabs1(a - b) <= EPS; }
6
7 // typedef int cod;
8 // bool eq(cod a, cod b){ return (a==b); }
9
10 struct point
11 {
12     cod x, y;
13     int id;
14     point(cod x=0, cod y=0): x(x), y(y){}
15
16
17     point operator+(const point &o) const{
18         return {x+o.x, y+o.y};
19     }
20     point operator-(const point &o) const{
21         return {x-o.x, y-o.y};
22     }
23     point operator*(cod t) const{
24         return {x*t, y*t};
25     }
26     point operator/(cod t) const{
27         return {x/t, y/t};
28     }
29 }

```

```

28 }
29 bool operator<(const point &o) const{
30     if(!eq(x, o.x)) return x < o.x;
31     return y < o.y;
32 }
33 bool operator==(const point &o) const{
34     return eq(x, o.x) and eq(y, o.y);
35 }
36
37 };
38
39 struct line
40 {
41     point fp, sp;
42     point(point fp=0, point sp=0): fp(fp), sp(sp){}
43
44     //a=y1-y2;
45     //b=x2-x1;
46     //c=x2*y1-y2*x1;
47 };
48
49
50
51 // Produto Escalar
52 cod dot(point a, point b){
53     return a.x*b.x + a.y*b.y;
54 }
55 // Produto Vetorial
56 cod cross(point a, point b){
57     return a.x*b.y - a.y*b.x;
58 }
59
60 ld norm(point a){ // Modulo
61     return sqrt(dot(a, a));
62 }
63 ld proj(point a, point b){ // a sobre b
64     return dot(a, b)/norm(b);
65 }
66 ld max(ld a, ld b){ return (a>b ? a:b); }
67 ld min(ld a, ld b){ return (a<b ? a:b); }
68 ld angle(point a, point b){ // em radianos
69     ld ang = dot(a, b) / norm(a) / norm(b);
70     return acos(max(min(ang, 1), -1));
71 }
72 int ccw(point a, point b, point e) //-1=dir; 0=
    collinear; 1=esq;
73 {
74     cod tmp = cross(b-a, e-a); // from a to b
75     return (tmp > EPS) - (tmp < -EPS);
76 }
77 ld order_angle(point a, point b) // from a to b ccw (
    a in front of b)
78 {
79     ld aux = angle(a,b)*180/PI;
80     return (cross(a,b)<=0 ? aux:360-aux);
81 }
82
83 bool collinear(point a, point b, point c){
84     return eq(cross(a-c, b-c), 0);
85 }
86
87 point rotccw(point p, ld a) // em radianos
88 {
89     //a = a*acos(0.0)/90; // graus
90     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
    +p.x*sin(a)));
91 }
92
93 point rot90cw(point a) { return {a.y, -a.x} };
94 point rot90ccw(point a) { return {-a.y, a.x} };
95
96 // Area de um poligono (pontos ordenados por
    adjacencia)

```



```

97 ld area(vector <point> p){
98     ld ret = 0;
99     for(int i=2;i<(int)p.size();i++)
100         ret += cross(p[i] - p[0], p[i-1] - p[0]);
101     return fabs1(ret/2);
102 }
103
104 // Dist entre ponto e reta
105 cod distr(point a, line b){
106     cod crs = cross(point(a - b.fp), point(b.sp - b.
107         fp));
108     return norm(crs/dist(b.fp, b.sp));
109 }

```

4 ED

4.1 Trie

```

1  class Trie {
2  private:
3      struct Node {
4          map<char, Node*> children;
5          int qt = 0;
6          ll size = 0;
7      };
8
9      Node* root;
10
11     void dfs(Node* cur) {
12         ll sz = 1;
13
14         for(auto prox : cur->children) {
15             dfs(prox.second);
16             sz += (prox.second->size);
17         }
18
19         cur->size = sz;
20     }
21
22     void del(Node* cur, int dep, string &s) {
23         if(dep >= 32)
24             return;
25
26         Node* prox = cur->children[s[dep]];
27         prox->qt--;
28         del(prox, dep+1, s);
29
30         if(prox->qt == 0)
31             cur->children.erase(s[dep]);
32     }
33
34 public:
35     Trie() {
36         root = new Node();
37         root->qt = 1;
38     }
39
40     void add(string s) {
41         Node* cur = root;
42
43         for(auto c : s) {
44             if(cur->children.count(c) == 0) {
45                 cur->children[c] = new Node();
46             }
47             cur->children[c]->qt++;
48             cur = cur->children[c];
49         }
50     }
51
52     void del(string &s) {
53         Node* cur = root;
54         del(cur, 0, s);

```

```

55     }
56
57     void size() {
58         this->dfs(root);
59     }
60 };

```

4.2 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     // If 2 values are equal the query will
19     // occur first then array element
20     if (a.val == b.val)
21         return a.l > b.l;
22
23     // Otherwise sorted in descending order.
24     return a.val > b.val;
25 }
26
27 // Updates the node of BIT array by adding
28 // 1 to it and its ancestors.
29 void update(int* BIT, int n, int idx){
30     while (idx <= n){
31         BIT[idx]++;
32         idx += idx & (-idx);
33     }
34 }
35
36 // Returns the count of numbers of elements
37 // present from starting till idx.
38 int query(int* BIT, int idx){
39     int ans = 0;
40     while (idx){
41         ans += BIT[idx];
42         idx -= idx & (-idx);
43     }
44     return ans;
45 }
46
47 // Function to solve the queries offline
48 void solveQuery(int arr[], int n, int QueryL[],
49     int QueryR[], int QueryK[], int q){
50     // create node to store the elements
51     // and the queries
52     node a[n + q + 1];
53     // 1-based indexing.
54
55     // traverse for all array numbers
56     for(int i = 1; i <= n; ++i){
57         a[i].val = arr[i - 1];
58         a[i].pos = 0;
59         a[i].l = 0;
60         a[i].r = i;
61     }
62
63     // iterate for all queries
64     for(int i = n + 1; i <= n + q; ++i){

```

```

65     a[i].pos = i - n;
66     a[i].val = QueryK[i - n - 1];
67     a[i].l = QueryL[i - n - 1];
68     a[i].r = QueryR[i - n - 1];
69 }
70
71 // In-built sort function used to
72 // sort node array using comp function.
73 sort(a + 1, a + n + q + 1, comp);
74
75 // Binary Indexed tree with
76 // initially 0 at all places.
77 int BIT[n + 1];
78
79 // initially 0
80 memset(BIT, 0, sizeof(BIT));
81
82 // For storing answers for each query( 1-based
83 // indexing ).
84 int ans[q + 1];
85
86 // traverse for numbers and query
87 for (int i = 1; i <= n + q; ++i){
88     if (a[i].pos != 0) {
89         // call function to returns answer for
90         // each query
91         int cnt = query(BIT, a[i].r) - query(BIT,
92             a[i].l - 1);
93
94         // This will ensure that answer of each
95         // query
96         // are stored in order it was initially
97         // asked.
98         ans[a[i].pos] = cnt;
99     }
100     else{
101         // a[i].r contains the position of the
102         // element in the original array.
103         update(BIT, n, a[i].r);
104     }
105 }
106 }
107
108 // Driver Code
109 int main()
110 {
111     int arr[] = { 7, 3, 9, 13, 5, 4 };
112     int n = sizeof(arr) / sizeof(arr[0]);
113
114     // 1-based indexing
115     int QueryL[] = { 1, 2 };
116     int QueryR[] = { 4, 6 };
117
118     // k for each query
119     int QueryK[] = { 6, 8 };
120
121     // number of queries
122     int q = sizeof(QueryL) / sizeof(QueryL[0]);
123
124     // Function call to get
125     solveQuery(arr, n, QueryL, QueryR, QueryK, q);
126
127     return 0;
128 }

```

4.3 Iterative-SegTree

```

1 // Segment Tree Iterativa - Range maximum query

```

```

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

```

4.4 Recursive-SegTree

```

1 // Segment Tree Recursiva - Range maximum query
2
3 int val[4*MAX];
4 int vet[MAX];
5
6 void monta(int i, int j, int no){
7     if(i==j){
8         val[no]=vet[i];
9         return;
10    }
11
12    int esq = 2*no;
13    int dir = 2*no+1;
14    int meio = (i+j)/2;
15
16    monta(i, meio, esq);
17    monta(meio+1, j, dir);
18
19    val[no]=max(val[esq], val[dir]);
20 }
21
22 void atualiza(int no, int i, int j, int pos, int
23     novo_valor){
24     if(i==j){

```

```

24     val[no]=novo_valor;
25 }else{
26     int esq = 2*no;
27     int dir = 2*no+1;
28     int meio = (i+j)/2;
29
30     if(pos<=meio)
31         atualiza(esq, i, meio, pos, novo_valor);
32     else
33         atualiza(dir, meio+1, j, pos, novo_valor);
34
35     if(val[esq]>val[dir])
36         val[no]=val[esq];
37     else
38         val[no]=val[dir];
39 }
40 }
41
42 int consulta(int no, int i, int j, int A, int B){
43     if(i>B || j<A)
44         return -1;
45     if(i>=A and j<=B)
46         return val[no];
47
48     int esq = 2*no;
49     int dir = 2*no+1;
50     int meio = (i+j)/2;
51
52     int resp_esq = consulta(esq, i, meio, A, B);
53     int resp_dir = consulta(dir, meio+1, j, A, B);
54
55     if(resp_dir!=-1)
56         return resp_esq;
57     if(resp_esq!=-1)
58         return resp_dir;
59
60     if(resp_esq>resp_dir)
61         return resp_esq;
62     else
63         return resp_dir;
64 }
65
66 int main()
67 {
68     monta(1, N, 1);
69     atualiza(1, 1, N, pos, valor);
70     x = consulta(1, 1, N, inicio, fim);
71 }

```

4.5 Delta-Encoding

```

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

```

4.6 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 queries
9
10 int v[MAX];
11 no arvore[MAX];
12
13 pii somar(pii a, pii b) // une pairs
14 {
15     return mp(a.f+b.f, a.s+b.s);
16 }
17
18 no une(no l, no r)
19 {
20     if(l.total.s==0)
21         return r;
22     if(r.total.s==0)
23         return l;
24
25     no m;
26
27     m.prefix = max(l.prefix, somar(l.total, r.prefix));
28     //prefixo
29     m.sufix = max(r.sufix, somar(r.total, l.sufix));
30     //sufixo
31     m.total = somar(l.total, r.total); //Soma de
32     // todos os elementos da subarvore
33     m.maximo = max(max(l.maximo, r.maximo), somar(l.
34     sufix, r.prefix)); //Resultado para cada
35     // subarvore
36
37     return m;
38 }
39
40 no makenozero()
41 {
42     no m;
43     m.prefix=m.sufix=m.total=m.maximo=mp(0,0);
44     return m;
45 }
46
47 no makeno(int k)
48 {
49     no m;
50     m.prefix=m.sufix=m.total=m.maximo=mp(k,1);
51     return m;
52 }
53
54 void monta(int n)
55 {
56     if(noleft[n]==noright[n])
57     {
58         arvore[n]=makeno(v[noleft[n]]);
59         return;
60     }
61
62     int mid = (noleft[n]+noright[n])/2;
63     noleft[2*n]=noleft[n]; noright[2*n]=mid;
64     noleft[2*n+1]=mid+1; noright[2*n+1]=noright[n];
65
66     monta(2*n);
67     monta(2*n+1);
68
69     arvore[n]=une(arvore[2*n], arvore[2*n+1]);
70 }
71
72 no busca(int n, int esq, int dir)
73 {
74     if(noleft[n]>=esq and noright[n]<=dir)
75         return arvore[n];
76     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 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 }

```

4.7 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 }

```

4.8 BIT

```

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

```

```

4
5     FT(int n) {
6         this->n = n + 1;
7         bit.assign(n + 1, 0);
8     }
9
10    int sum(int idx) {
11        int ret = 0;
12        for (++idx; idx > 0; idx -= idx & -idx)
13            ret += bit[idx];
14        return ret;
15    }
16
17    int sum(int l, int r) {
18        return sum(r) - sum(l - 1);
19    }
20
21    void add(int idx, int delta) {
22        for (++idx; idx <= n; idx += idx & -idx)
23            bit[idx] += delta;
24    }
25 };

```

4.9 Sparse-Table

```

1 int logv[MAX+1];
2 void make_log() {
3     logv[1] = 0; // pre-computar tabela de log
4     for (int i = 2; i <= MAX; i++)
5         logv[i] = logv[i/2] + 1;
6 }
7
8 struct Sparse {
9     int n;
10    vector<vector<int>> st;
11
12    Sparse(int n, vi array) {
13        this->n = n;
14        int k = logv[n];
15        st.assign(n+1, vector<int>(k+1, 0));
16
17        for (int i = 0; i < n; i++)
18            st[i][0] = array[i];
19
20        for (int j = 1; j <= k; j++)
21            for (int i = 0; i + (1 << j) <= n; i++)
22                st[i][j] = f(st[i][j-1], st[i + (1 <<
23                (j - 1))][j - 1]);
24    }
25
26    int f(int a, int b) {
27        return min(a, b);
28    }
29
30    int query(int L, int R) {
31        int j = logv[R - L + 1];
32        int res = f(st[L][j], st[R - (1 << j) + 1][j
33        ]);
34        return res;
35    }
36 };

```

4.10 Union-Find

```

1 struct DSU {
2     int n;
3     vector<int> parent, size;
4
5     DSU(int n) {
6         this->n = n;
7         parent.assign(n+1, 0);
8         size.assign(n+1, 1);

```

```

9
10     for(int i = 0; i <= n; i++)
11         parent[i] = i;
12 }
13
14 int find(int v) {
15     if(v == parent[v])
16         return v;
17     return find(parent[v]);
18 }
19
20 void join(int a, int b) {
21     a = find(a);
22     b = find(b);
23
24     if(a != b) {
25         if(size[a] < size[b])
26             swap(a, b);
27
28         parent[b] = a;
29         size[a] += size[b];
30     }
31 }
32 };

```

4.11 Mo

```

1  const int BLK = 500; // tamanho do bloco, algo entre
   300 e 500 e nice
2
3  struct Query {
4      int l, r, idx;
5      Query(int l, int r, int idx) {
6          this->l = l;
7          this->r = r;
8          this->idx = idx;
9      }
10     bool operator<(Query other) const {
11         return make_pair(l / BLK, r) <
12             make_pair(other.l / BLK, other.r);
13     }
14 };
15
16 void add() void remove() // implementar operacoes de
   acordo com o problema, cuidado com TLE ao
   utilizar MAP
17
18 vector<int> mo(vector<Query> queries) {
19     vector<int> res(queries.size());
20     sort(queries.begin(), queries.end());
21     resposta = 0;
22
23     int l = 0, r = -1;
24     for(Query q : queries) {
25         while(l > q.l) {
26             l--;
27             add(l);
28         }
29         while(r < q.r) {
30             r++;
31             add(r);
32         }
33         while(l < q.l) {
34             remove(l);
35             l++;
36         }
37         while(r > q.r) {
38             remove(r);
39             r--;
40         }
41         res[q.idx] = resposta; // adicionar resposta
   de acordo com o problema
42     }

```

```

43     return res; // ordenar o vetor pelo indice e
   responder queries na ordem
44 }

```

5 Math

5.1 Totient

```

1  // phi(p^k) = (p^(k-1))*(p-1) com p primo
2  // O(sqrt(m))
3  ll phi(ll m) {
4      ll res = m;
5      for(ll d = 2; d*d <= m; d++) {
6          if(m % d == 0) {
7              res = (res/d) * (d-1);
8              while(m % d == 0) {
9                  m /= d;
10             }
11         }
12     }
13     if(m > 1) {
14         res /= m;
15         res *= (m-1);
16     }
17
18     return res;
19 }
20
21 // modificacao do crivo, O(n*log(log(n)))
22 vector<ll> phi_to_n(ll n) {
23     vector<bool> isprime(n+1, true);
24     vector<ll> tot(n+1);
25     tot[0] = 0; tot[1] = 1;
26     for(ll i = 1; i <= n; i++) {
27         tot[i] = i;
28     }
29
30     for(ll p = 2; p <= n; p++) {
31         if(isprime[p]) {
32             tot[p] = p-1;
33             for(ll i = p+p; i <= n; i += p) {
34                 isprime[i] = false;
35                 tot[i] = (tot[i]/p)*(p-1);
36             }
37         }
38     }
39
40     return tot;
41 }

```

5.2 Sqrt-BigInt

```

1  public static BigInteger isqrtNewton(BigInteger n) {
2      BigInteger a = BigInteger.ONE.shiftLeft(n.
   bitLength() / 2);
3      boolean p_dec = false;
4      for (;;) {
5          BigInteger b = n.divide(a).add(a).shiftRight
   (1);
6          if (a.compareTo(b) == 0 || a.compareTo(b) < 0
   && p_dec)
7              break;
8          p_dec = a.compareTo(b) > 0;
9          a = b;
10     }
11     return a;
12 }

```

5.3 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

```

5.4 Sum-n2

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

5.5 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 }

```

5.6 Modular-Exponentiation

```

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

```

```

7
8     return res%mod;
9 }

```

5.7 Miller-Habin

```

1 ll llrand()
2 {
3     ll tmp = rand();
4     return (tmp << 31) | rand();
5 }
6
7 ll add(ll a, ll b, ll c)
8 {
9     return (a + b)%c;
10 }
11
12 ll mul(ll a, ll b, ll c)
13 {
14     ll ans = 0;
15     while(b)
16     {
17         if(b & 1)
18             ans = add(ans, a, c);
19         a = add(a, a, c);
20         b /= 2;
21     }
22     return ans;
23 }
24
25 ll fexp(ll a, ll b, ll c)
26 {
27     ll ans = 1;
28     while(b)
29     {
30         if(b & 1)
31             ans = mul(ans, a, c);
32         a = mul(a, a, c);
33         b /= 2;
34     }
35     return ans;
36 }
37
38 bool rabin(ll n)
39 {
40     if(n <= 1)
41         return 1;
42     if(n <= 3)
43         return 1;
44
45     ll s=0, d=n-1;
46     while(d%2==0)
47     {
48         d/=2;
49         s++;
50     }
51
52     for(int k = 0; k < 64*4; k++)
53     {
54         ll a = (llrand()%(n - 3)) + 2;
55         ll x = fexp(a, d, n);
56         if(x != 1 and x != n-1)
57         {
58             for(int r = 1; r < s; r++)
59             {
60                 x = mul(x, x, n);
61                 if(x == 1)
62                     return 0;
63                 if(x == n-1)
64                     break;
65             }
66             if(x != n-1)
67                 return 0;

```

```

68     }
69 }
70
71 return 1;
72 }
73
74
75 int main()
76 {
77     ll N;
78     cin >> N;
79
80     cout << rabin(N) << endl;
81
82     return 0;
83 }
84
85 }

```

5.8 Inverso-Mult

```

1 // gcd(a, m) = 1 para existir solucao
2 // ax + my = 1, ou a*x = 1 (mod m)
3 ll inv(ll a, ll m) { // com gcd
4     ll x, y;
5     gcd(a, m, x, y);
6     return ((x % m) + m) % m;
7 }
8
9 ll inv(ll a, ll phim) { // com phi(m), se m for primo
10     entao phi(m) = p-1
11     ll e = phim-1;
12     return fexp(a, e);
13 }

```

5.9 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         div.pb(d);
75         while(N%d==0)
76             N/=d;
77     }
78     if(N!=1)
79         div.pb(N);
80
81     return 0;
82 }
83
84 }

```

5.10 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 }

```

5.11 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;

```

5.12 Simpson's-formula

```

1 inline ld simpson(ld fl, ld fr, ld fmid, ld l, ld r){
2     return (fl+fr+4*fmid)*(r-l)/6;
3 }
4
5 ld rsimpson(ld slr, ld fl, ld fr, ld fmid, ld l, ld r
6 )
7 {
8     ld mid = (l+r)/2;
9     ld fml = f((l+mid)/2), fmr = f((mid+r)/2);
10    ld slm = simpson(fl,fmid,fml,l,mid);
11    ld smr = simpson(fmid,fr,fmr,mid,r);
12    if(fabs(slr-slm-smr) < EPS) return slm+smr; //
13    approx. good enough
14    return rsimpson(slm,fl,fmid,fml,l,mid)+rsimpson(
15    smr,fmid,fr,fmr,mid,r);
16 }
17
18 ld integrate(ld l, ld r)
19 {
20     ld mid = (l+r)/2;
21     ld fl = f(l), fr = f(r);
22     ld fmid = f(mid);
23     return rsimpson(simpson(fl,fr,fmid,l,r),fl,fr,
24     fmid,l,r);
25 }

```

5.13 FFT

```

1 struct num{
2     double x, y;
3     num() { x = y = 0; }
4     num(double x, double y) : x(x), y(y) {}
5 };
6
7 inline num operator+(num a, num b) { return num(a.x +
8     b.x, a.y + b.y); }
9 inline num operator-(num a, num b) { return num(a.x -
10     b.x, a.y - b.y); }
11 inline num operator*(num a, num b) { return num(a.x *
12     b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
13 inline num conj(num a) { return num(a.x, -a.y); }
14
15 int base = 1;
16 vector<num> roots = {{0, 0}, {1, 0}};
17 vector<int> rev = {0, 1};
18
19 const double PI = acos(-1.0);
20
21 void ensure_base(int nbase){
22     if(nbase <= base)
23         return;
24
25     rev.resize(1 << nbase);
26     for(int i = 0; i < (1 << nbase); i++)
27         rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (
28         nbase - 1));
29
30     roots.resize(1 << nbase);
31
32     while(base < nbase){

```

```

29         double angle = 2*PI / (1 << (base + 1));
30         for(int i = 1 << (base - 1); i < (1 << base);
31             i++){
32             roots[i << 1] = roots[i];
33             double angle_i = angle * (2 * i + 1 - (1
34             << base));
35             roots[(i << 1) + 1] = num(cos(angle_i),
36             sin(angle_i));
37         }
38         base++;
39     }
40 }
41
42 void fft(vector<num> &a, int n = -1){
43     if(n == -1)
44         n = a.size();
45
46     assert((n & (n-1)) == 0);
47     int zeros = __builtin_ctz(n);
48     ensure_base(zeros);
49     int shift = base - zeros;
50     for(int i = 0; i < n; i++){
51         if(i < (rev[i] >> shift))
52             swap(a[i], a[rev[i] >> shift]);
53     }
54
55     for(int k = 1; k < n; k <= 1)
56         for(int i = 0; i < n; i += 2 * k)
57             for(int j = 0; j < k; j++){
58                 num z = a[i+j+k] * roots[j+k];
59                 a[i+j+k] = a[i+j] - z;
60                 a[i+j] = a[i+j] + z;
61             }
62     }
63
64     vector<num> fa, fb;
65     vector<int> multiply(vector<int> &a, vector<int> &b){
66         int need = a.size() + b.size() - 1;
67         int nbase = 0;
68         while((1 << nbase) < need) nbase++;
69         ensure_base(nbase);
70         int sz = 1 << nbase;
71         if(sz > (int) fa.size())
72             fa.resize(sz);
73
74         for(int i = 0; i < sz; i++){
75             int x = (i < (int) a.size() ? a[i] : 0);
76             int y = (i < (int) b.size() ? b[i] : 0);
77             fa[i] = num(x, y);
78         }
79
80         fft(fa, sz);
81         num r(0, -0.25 / sz);
82         for(int i = 0; i <= (sz >> 1); i++){
83             int j = (sz - i) & (sz - 1);
84             num z = (fa[j] * fa[j] - conj(fa[i] * fa[i]))
85             * r;
86             if(i != j) {
87                 fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[
88                 j])) * r;
89             }
90             fa[i] = z;
91         }
92         fft(fa, sz);
93         vector<int> res(need);
94         for(int i = 0; i < need; i++)
95             res[i] = fa[i].x + 0.5;
96
97         return res;
98     }
99
100 int main()
101 {sws;
102
103     //FFT

```



```

97     vector<int> fx{1, 2, 3}; // 1+2x+3x^2
98     vector<int> gx{4, 5}; // 4+5x
99     vector<int> res;
100
101     res = multiply(fx,gx); //4 + 13x + 22x^2 + 15x^3
102
103     return 0;
104
105 }

```

5.14 Next-Permutation

```

1 vector<int> a = {1, 2, 3};
2 int n = a.size();
3 do{
4     display(a, n); // 1,2,3; 1,3,2; 2,1,3; 3,1,2;
5     // 2,3,1; 3,2,1;
6 }while(next_permutation(a.begin(), a.begin() + n));

```

5.15 Fast-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 }

```

5.16 Recursao-linear

```

1 vector<vl> id(int n) {
2     vector<vl> res(n, vl(n, 0));
3     for(int i = 0; i < n; i++) res[i][i] = 1;
4     return res;
5 }
6
7 vector<vl> mult(vector<vl> a, vector<vl> b, int n) {
8     vector<vl> res(n, vl(n, 0));
9
10    for(int row = 0; row < n; row++) {
11        for(int col = 0; col < n; col++) {
12            ll val = 0;
13            for(int k = 0; k < n; k++) {
14                ll delta = (a[row][k] * b[k][col]) %
15                MOD;
16                val = (val + delta) % MOD;
17            }
18            res[row][col] = val;
19        }
20    }
21    return res;
22 }
23
24 vector<vl> fexp(vector<vl> b, ll e, int n) {
25     if(e == 0) {
26         return id(n);
27     }
28
29     vector<vl> res = fexp(b, e/2, n);
30     res = mult(res, res, n);
31

```

```

32     if(e%2)
33         res = mult(res, b, n);
34
35     return res;
36 }
37
38 // k = tamanho da recorrência/matriz, n = n-esimo
39 // termo
40 // f(n) = c1*f(n-1) + c2*f(n-2) + ... + ck*f(n-k)
41 // base -> [f(k-1), f(k-2), ..., f(0)]
42 // coeficientes -> [c1, c2, ..., ck]
43 vl solve(int k, int n, vl base, vl coef) {
44     vector<vl> inicial;
45     inicial.pb(coef);
46     for(int row = 0; row < k-1; row++) {
47         vl tmp;
48         for(int col = 0; col < k; col++) {
49             if(col == row)
50                 tmp.pb(1);
51             else
52                 tmp.pb(0);
53         }
54         inicial.pb(tmp);
55     }
56     vector<vl> matexp = fexp(inicial, max(0, n-k+1),
57     k);
58     vl res(k);
59     for(int row = 0; row < k; row++) {
60         ll val = 0;
61         for(int aux = 0; aux < k; aux++) {
62             val += matexp[row][aux]*base[aux];
63         }
64         res[row] = val; // res = (f(n), f(n-1), ...,
65         f(n-k+1))
66     }
67     return res;
68 }

```

5.17 Raiz-primitiva

```

1 ll fexp(ll b, ll e, ll mod) {
2     if(e == 0) return 1LL;
3     ll res = fexp(b, e/2LL, mod);
4     res = (res*res)%mod;
5     if(e%2LL)
6         res = (res*b)%mod;
7
8     return res%mod;
9 }
10
11 vl fatorar(ll n) { // fatora em primos
12     vl fat;
13     for(int i = 2; i*i <= n; i++) {
14         if(n%i == 0) {
15             fat.pb(i);
16             while(n%i == 0)
17                 n /= i;
18         }
19     }
20     return fat;
21 }
22
23 // O(log(n) ^ 2)
24 bool raiz_prim(ll a, ll mod, ll phi, vl fat) {
25     if(__gcd(a, mod) != 1 or fexp(a, phi/2, mod) ==
26     1) // phi de euler sempre eh PAR
27         return false;
28     for(auto f : fat) {
29         if(fexp(a, phi/f, mod) == 1)

```

```

30         return false;
31     }
32
33     return true;
34 }
35
36 // mods com raizes primitivas: 2, 4, p^k, 2*p^k, p eh
37 // primo impar, k inteiro --- O(n log^2(n))
38 ll achar_raiz(ll mod, ll phi) {
39     if(mod == 2) return 1;
40     vl fat, elementos;
41     fat = fatorar(phi);
42
43     for(ll i = 2; i <= mod-1; i++) {
44         if(raiz_prim(i, mod, phi, fat))
45             return i;
46     }
47
48     return -1; // retorna -1 se nao existe
49 }
50 vl todas_raizes(ll mod, ll phi, ll raiz) {
51     vl raizes;
52     if(raiz == -1) return raizes;
53     ll r = raiz;
54     for(ll i = 1; i <= phi-1; i++) {
55         if(__gcd(i, phi) == 1) {
56             raizes.pb(r);
57         }
58         r = (r * raiz) % mod;
59     }
60
61     return raizes;
62 }

```

5.18 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 }

```

6 Misc

6.1 2SAT

```

1 vector<int> g[MAX], gt[MAX], S; int vis[MAX], cor[MAX]
2 ];
3 int val(int n, bool tvalue) {
4     if(tvalue) return 2*n;
5     return 2*n + 1;
6 }
7
8 void dfs(int u) {
9     vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
10

```

```

10     S.push_back(u);
11 }
12
13 void dfst(int u, int e) {
14     cor[u] = e;
15     for(int v : gt[u]) if(!cor[v]) dfst(v, e);
16 }
17
18 void kosaraju(int n) {
19     for(int i = 0; i <= n; i++) if(!vis[i]) dfs(i);
20     for(int i = 0; i <= n; i++) for(int j : g[i])
21         gt[j].push_back(i);
22     int e = 0; reverse(S.begin(), S.end());
23     for(int u : S) if(!cor[u]) dfst(u, ++e);
24 }
25
26 // antes de chamar essa funcao, colocar as arestas do
27 // grafo
28 bool solve(int n, vi &res) {
29     kosaraju(2*n); // MAX > 2*N
30     vi r;
31
32     forn(i, n) {
33         int t = val(i, true), f = val(i, false);
34         if(cor[t] == cor[f]) {
35             return false;
36         }
37         else {
38             if(cor[t] > cor[f])
39                 r.pb(1);
40             else
41                 r.pb(0);
42         }
43     }
44     swap(r, res);
45     return true;
46 }

```

6.2 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();

```

6.3 Bitwise

```

1 // Bitwise
2 #pragma GCC target("popcnt")
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

```

```

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 // Least significant bit (lsb)
46
47     int lsb(int x){ return x&-x; }
48
49 // Exchange o j-th element
50
51     S ^= (1<<j)
52
53 // Position of the first bit on
54
55     T = (S & (-S))
56     T -> 4 bit ligado //(1000)
57
58 // Most significant digit of N
59
60     double K = log10(N);
61     K = K - floor(K);
62     int X = pow(10, K);
63
64 // Number of digits in N
65
66     X =floor(log10(N)) + 1;
67
68 // Power of two
69
70     bool isPowerOfTwo(int x){ return x && (!(x&(x
-1)))}; }
71
72 // Turn off the first bit 1
73     m = m & (m-1);
74
75 // Built-in functions
76
77 // Number of bits 1
78     __builtin_popcount()
79     __builtin_popcountll()
80
81 // Number of leading zeros
82     __builtin_clz()
83     __builtin_clzll()
84
85 // Number of trailing zeros

```

```

86     __builtin_ctz()
87     __builtin_ctzll()
88
89 // floor(log2(x))
90
91     int flog2(int x){ return 32-1-__builtin_clz(x
); }
92
93     int flog2ll(ll x){ return 64-1-
__builtin_clzll(x); }

```

6.4 Template

```

1 #include <bits/stdc++.h>
2 #define ff first
3 #define ss second
4 #define ll long long
5 #define ld long double
6 #define pb push_back
7 #define eb emplace_back
8 #define mp make_pair
9 #define mt make_tuple
10 #define pii pair<int, int>
11 #define vi vector<int>
12 #define sws ios_base::sync_with_stdio(false);cin.tie(
NULL)
13 #define endl '\n'
14 #define teto(a, b) (a+b-1)/(b)
15
16 const int MAX = 400010;
17 const int MOD = 1e9+7;
18 const int INF = 0x3f3f3f3f;
19 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
20 const ld EPS = 1e-7;
21
22 using namespace std;

```

7 Strings

7.1 Trie

```

1 int trie[MAX][26];
2 bool finish[MAX];
3 int nxt = 1;
4
5 void Add(string &str){
6     int node = 0;
7     for(auto s: str){
8         if(trie[node][s-'a'] == 0){
9             node = trie[node][s-'a'] = nxt;
10            nxt++;
11        }else
12            node = trie[node][s-'a'];
13    }
14    finish[node] = true;
15 }
16
17 bool Find(string &str){
18     int idx = 0;
19     for(auto s: str)
20         if(trie[idx][s-'a'] == 0)
21             return false;
22         else
23             idx = trie[idx][s-'a'];
24     return finish[idx];
25 }

```

7.2 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
11 void kmp(const string &t, const string &p){
12     vector<int> b = prefix_function(p);
13     int n = t.size(), m = p.size();
14     int j = 0;
15     for(int i = 0; i < n; i++){
16         while(j >= 0 && t[i] != p[j]) j = b[j];
17         j++;
18         if(j == m) {
19             j = b[j];
20         }
21     }
22 }

```

7.3 LCS

```

1 string LCSSubStr(string X, string Y)
2 {
3     int m = X.size();
4     int n = Y.size();
5
6     int result = 0, end;
7     int len[2][n];
8     int currRow = 0;
9
10    for(int i=0;i<=m;i++){
11        for(int j=0;j<=n;j++){
12            if(i==0 || j==0)
13                len[currRow][j] = 0;
14            else if(X[i-1] == Y[j-1]){
15                len[currRow][j] = len[1-currRow][j-1]
16                + 1;
17                if(len[currRow][j] > result){
18                    result = len[currRow][j];
19                    end = i - 1;
20                }
21            }
22            else
23                len[currRow][j] = 0;
24
25            currRow = 1 - currRow;
26        }
27
28        if(result==0)
29            return string();
30
31        return X.substr(end - result + 1, result);
32 }

```

7.4 Pal-int

```

1 bool ehpalindromo(ll n) {
2     if(n<0)
3         return false;
4
5     int divisor = 1;
6     while(n/divisor >= 10)
7         divisor *= 10;
8
9     while(n != 0) {
10        int leading = n / divisor;
11        int trailing = n % 10;
12
13        if(leading != trailing)

```

```

14            return false;
15
16            n = (n % divisor)/10;
17
18            divisor = divisor/100;
19        }
20
21        return true;
22 }

```

7.5 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 }

```

7.6 Hash

```

1 ll compute_hash(string const& s) {
2     const ll p = 31; // primo, melhor = perto da
3     // quantidade de caracteres
4     const ll m = 1e9 + 9; // maior mod = menor
5     // probabilidade de colisao
6     ll hash_value = 0;
7     ll p_pow = 1;
8     for (char c : s) {
9         hash_value = (hash_value + (c - 'a' + 1) *
10         p_pow) % m;
11         p_pow = (p_pow * p) % m;
12     }
13     return hash_value;
14 }

```

7.7 Manacher

```

1 // 0(n), d1 -> palindromo impar, d2 -> palindromo par
2 // (centro da direita)
3 void manacher(string &s, vi &d1, vi &d2) {
4     int n = s.size();
5     for(int i = 0, l = 0, r = -1; i < n; i++) {
6         int k = (i > r) ? 1 : min(d1[l + r - i], r -
7         i + 1);
8         while(0 <= i - k && i + k < n && s[i - k] ==
9         s[i + k]) {
10            k++;
11        }
12        d1[i] = k--;
13        if(i + k > r) {
14            l = i - k;
15            r = i + k;
16        }
17    }
18
19    for(int i = 0, l = 0, r = -1; i < n; i++) {
20        int k = (i > r) ? 0 : min(d2[l + r - i + 1],
21        r - i + 1);

```

```

18     while(0 <= i - k - 1 && i + k < n && s[i - k 23         l = i - k - 1;
- 1] == s[i + k]) { 24         r = i + k ;
19         k++; 25     }
20     } 26
21     d2[i] = k--; 27 }
22     if(i + k > r) {

```