



# Notebook - Maratona de Programação

Tiago de Souza Fernandes

## Sumário

<b>1</b>	<b>Algoritmos</b>	<b>2</b>			
1.1	Mochila	2			
1.2	Kadane-DP	2			
1.3	Iterative-BS	2			
<b>2</b>	<b>Grafos</b>	<b>2</b>			
2.1	BFS	2			
2.2	Find-bridges	2			
2.3	Dijkstra	2			
2.4	LCA	3			
2.5	Floyd-Warshall	3			
2.6	Kruskal	3			
2.7	DFS	4			
2.8	Kosaraju	4			
2.9	Represent	4			
2.10	Centroid	4			
2.11	Prim	4			
<b>3</b>	<b>Geometria</b>	<b>5</b>			
3.1	Angle-adjacent-vertices-regular-polygon	5			
3.2	Inter-Retas	5			
3.3	Pick's-theorem	5			
3.4	Cross-properties	5			
3.5	Inter-Retangulos	6			
3.6	3D	6			
3.7	Dot-properties	6			
3.8	2D	6			
<b>4</b>	<b>ED</b>	<b>7</b>			
4.1	Range-query-bigger-than-k-BIT	7			
4.2	Iterative-SegTree	8			
4.3	Recursive-SegTree	8			
4.4	Delta-Encoding	9			
4.5	Seg-Tree-Farao	9			
4.6	BIT-2D	10			
4.7	BIT	10			
4.8	Sparse-Table	10			
			4.9	Union-Find	10
			4.10	Mo	11
<b>5</b>	<b>Math</b>	<b>11</b>			
5.1	Totient	11			
5.2	Sqrt-BigInt	11			
5.3	Linear-Diophantine-Equation	11			
5.4	Sum-n2	12			
5.5	Factorization-sqrt	12			
5.6	Modular-Exponentiation	12			
5.7	Miller-Habin	12			
5.8	Inverso-Mult	13			
5.9	Pollard-Rho	13			
5.10	Verif-primo	13			
5.11	Crivo	13			
5.12	Simpson's-formula	13			
5.13	FFT	14			
5.14	Modular-Factorial	14			
5.15	Recursao-linear	15			
5.16	Kamenetsky	15			
<b>6</b>	<b>Misc</b>	<b>16</b>			
6.1	LIS	16			
6.2	Bitwise	16			
6.3	Template	16			
<b>7</b>	<b>Strings</b>	<b>17</b>			
7.1	KMP	17			
7.2	LCS	17			
7.3	Pal-int	17			
7.4	Z-Func	17			
7.5	Hash	17			

# 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 {
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[tol] > t[v])
24                 IS_BRIDGE(v, vert);
25         }
26     }
27 }
28
29 int main()
30 {
31     timer = 0;
32     visited.assign(N+1, false);
33     t.assign(N+1, 0);
34     low.assign(N+1, 0);
35
36     for(int i=0; i<N; i++)
37         if(!visited[i])
38             find_bridges(i);
39
40     return 0;
41 }
```

## 2.3 Dijkstra

```
1 // Dijkstra - Shortest Path
2
```

```

3 #define pii pair<int, int>
4 #define vi vector<int>
5 #define vii vector< pair<int,int> >
6 #define INF 0x3f3f3f3f
7
8 vector<vii> grafo(N+1, vii());
9 vi distancia(N+1, INF);
10 priority_queue< pii, vii, greater<pii> > fila;
11
12 void dijkstra(int k)
13 {
14     int dist, vert, aux;
15     distancia[k]=0;
16
17     fila.push(mp(k, 0));
18
19     while(!fila.empty())
20     {
21         aux=fila.top().f;
22         fila.pop();
23
24         for(auto v: grafo[aux])
25         {
26             vert=v.f;
27             dist=v.s;
28             if(distancia[vert]>distancia[aux]+dist)
29             {
30                 distancia[vert]=distancia[aux]+dist;
31                 fila.push(mp(vert, distancia[vert]));
32             }
33         }
34     }
35 }
36
37 int main()
38 {
39     for(int i=0; i<M; i++)
40     {
41         cin >> a >> b >> p;
42         grafo[a].pb(mp(b, p));
43         grafo[b].pb(mp(a, p));
44     }
45 }

```

## 2.4 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]);

```

```

25
26     return minimum;
27 }
28
29 int start[MAX+1], dfs_time;
30 int tour[2*MAX+1], id[2*MAX+1];
31
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     dfs(1);
64
65     make();
66     precompute(2*N, tour);
67
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

```

```

2 struct Edge {
3     int u, v, weight;
4     bool operator<(Edge const& other) {
5         return weight < other.weight;
6     }
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 {
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 }

```

## 2.8 Kosaraju

```

1 // KOSARAJU - O(V+E) - encontra componentes
2 // fortemente conexos
3 // g -> grafo, gt -> grafo tempo
4 // vis -> visitado, cor -> componente fortemente
5 // conexo ordenado topologicamente
6 vector<int> g[N], gt[N], S; int vis[N], cor[N];
7 void dfs(int u){
8     vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
9     S.push_back(u);
10 }
11 void dfst(int u, int e){
12     cor[u] = e;
13     for(int v : gt[u]) if(!cor[v]) dfst(v, e);
14 }
15 void kosaraju(){
16     for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);
17     for(int i = 1; i <= n; i++) for(int j : g[i])
18         gt[j].push_back(i);
19     int e = 0; reverse(S.begin(), S.end());
20     for(int u : S) if(!cor[u]) dfst(u, ++e);
21 }

```

## 2.9 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);

```

## 2.10 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 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) );
18
19     while(1)
20     {
21         int davez = -1;
22
23         for(int i = 2; i <= n; i++)
24             if(distancia[i] < davez) davez = distancia[i];
25
26         if(davez == -1) break;
27
28         int u = davez;
29         processado[u] = 1;
30
31         for(int v : vizinhos[u])
32             if(!processado[v] & distancia[v] > distancia[u] + g[u][v])
33                 distancia[v] = distancia[u] + g[u][v];
34     }
35 }

```

```

22     while(!fila.empty())
23     {
24         int atual = fila.top().second;
25         fila.pop();
26
27         if(!processado[atual])
28         {
29             davez = atual;
30             break;
31         }
32     }
33
34     if(davez == -1)
35         break;
36
37     processado[davez] = true;
38
39     for(int i = 0; i < (int)vizinhos[davez].size()
40 ; i++)
41     {
42         int dist = vizinhos[davez][i].first;
43         int atual = vizinhos[davez][i].second;
44
45         if( distancia[atual] > dist && !
46 processado[atual])
47         {
48             distancia[atual] = dist;
49             fila.push( pii(distancia[atual],
50 atual) );
51         }
52     }
53
54     int custo_arvore = 0;
55     for(int i = 1; i <= n; i++)
56         custo_arvore += distancia[i];
57
58     return custo_arvore;
59 }
60
61 int main(){
62     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 Angle-adjacent-vertices-regular-polygon

$$a = 180/N$$

### 3.2 Inter-Retas

```

1 // Intersection between lines
2
3 typedef struct
4 {

```

```

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

```

### 3.3 Pick's-theorem

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

### 3.4 Cross-properties

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

### 3.5 Inter-Retangulos

```
1 typedef struct
2 {
3     int x, y;
4 } Point;
5
6 bool doOverlap(Point l1, Point r1, Point l2, Point r2
7 )
8 {
9     if (l1.x>r2.x or l2.x>r1.x or l1.y<r2.y or l2.y<
10    r1.y)
11         return false;
12     return true;
13 }
```

### 3.6 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
11     point operator+(const point &o) const{
12         return {x+o.x, y+o.y, z+o.z};
13     }
14     point operator-(const point &o) const{
15         return {x-o.x, y-o.y, z-o.z};
16     }
17     point operator*(cod t) const{
18         return {x*t, y*t, z*t};
19     }
20     point operator/(cod t) const{
21         return {x/t, y/t, z/t};
22     }
23     bool operator==(const point &o) const{
24         return eq(x, o.x) and eq(y, o.y) and eq(z, o.
25         z);
26     }
27 };
28
29 // Produto Escalar
30 cod dot(point a, point b){
31     return a.x*b.x + a.y*b.y + a.z*b.z;
32 }
33
34 // Produto Vetorial
35 point cross(point a, point b){
36     return point(a.y*b.z - a.z*b.y,
37                 a.z*b.x - a.x*b.z,
38                 a.x*b.y - a.y*b.x);
39 }
40
41 ld abs(point a){ // Modulo
42     return sqrt(dot(a, a));
43 }
44
45 ld proj(point a, point b){ // a sobre b
46     return dot(a, b)/abs(b);
47 }
48
49 ld angle(point a, point b){ // em radianos
50     return acos(dot(a, b) / abs(a) / abs(b));
51 }
52
53 cod triple(point a, point b, point c){
54     return dot(a, cross(b, c)); // Area do
55     paralelepipedo
56 }
57 }
```

### 3.7 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.

### 3.8 2D

```
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;
8     int id;
9     point(cod x=0, cod y=0): x(x), y(y){}
10
11     point operator+(const point &o) const{
12         return {x+o.x, y+o.y};
13     }
14     point operator-(const point &o) const{
15         return {x-o.x, y-o.y};
16     }
17     point operator*(cod t) const{
18         return {x*t, y*t};
19     }
20     point operator/(cod t) const{
21         return {x/t, y/t};
22     }
23     bool operator==(const point &o) const{
24         return eq(x, o.x) and eq(y, o.y);
25     }
26 };
27
28 struct line
29 {
30     point fp, sp;
31     point(point fp=0, point sp=0): fp(fp), sp(sp){}
32
33     //a=y1-y2;
34     //b=x2-x1;
35     //c=x2*y1-y2*x1;
36 };
37
38 // Produto Escalar
39 cod dot(point a, point b){
40     return a.x*b.x + a.y*b.y;
41 }
42
43 // Produto Vetorial
44 cod cross(point a, point b){
45     return a.x*b.y - a.y*b.x;
46 }
47
48 ld norm(point a){ // Modulo
49     return sqrt(dot(a, a));
50 }
51
52 ld proj(point a, point b){ // a sobre b
53     return dot(a, b)/norm(b);
54 }
55
56 ld angle(point a, point b){ // em radianos
```

```

58     return acos(dot(a, b) / norm(a) / norm(b));
59 }
60 cod dist(point a, point b){
61     return norm(a-b); // Dist euclidiana
62 }
63 int ccw(point a, point b){ // -1=cw; 0=parallel; 1=
64     ccw;
65     cod tmp = cross(a, b); // from a to b.
66     return (tmp > EPS) - (tmp < -EPS);
67 }
68 bool collinear(point a, point b, point c){
69     return eq(cross(a-b, b-c), 0);
70 }
71
72 point rot90cw(point a) { return {a.y, -a.x} };
73 point rot90ccw(point a) { return {-a.y, a.x} };
74
75
76 // Dist entre ponto e reta
77 cod distr(point a, line b){
78     cod crs = cross(point(a - b.fp), point(b.sp - b.
79     fp));
80     return norm(crs/dist(b.fp, b.sp));
81 }
82
83
84 int esq(point a, point b, point e)
85 { // From a to b: Esquerda = 1; Direita = -1;
86     Collinear = 0;
87     ll v = a.x*b.y + b.x*e.y + e.x*a.y - (a.y*b.x + b
88     .y*e.x + e.y*a.x);
89     if(v>0) return 1;
90     if(v==0) return 0;
91     return -1;
92 }
93
94 // Area de um poligono (pontos ordenados por
95     adjacencia)
96 cod area(vector <point> p){
97     cod ret = 0;
98     for(int i=2;i<(int)p.size();i++)
99         ret += cross(p[i] - p[0], p[i-1] - p[0])/2;
100     return norm(ret);
101 }
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

each query
94     int cnt = query(BIT, a[i].r) - query(BIT,
95     a[i].l - 1);
96
97     // This will ensure that answer of each
98     query
99     // are stored in order it was initially
100     asked.
101     ans[a[i].pos] = cnt;
102 }
103 else {
104     // a[i].r contains the position of the
105     // element in the original array.
106     update(BIT, n, a[i].r);
107 }
108 }
109 // Output the answer array
110 for (int i = 1; i <= q; ++i) {
111     cout << ans[i] << endl;
112 }
113 }
114 // Driver Code
115 int main()
116 {
117     int arr[] = { 7, 3, 9, 13, 5, 4 };
118     int n = sizeof(arr) / sizeof(arr[0]);
119
120     // 1-based indexing
121     int QueryL[] = { 1, 2 };
122     int QueryR[] = { 4, 6 };
123
124     // k for each query
125     int QueryK[] = { 6, 8 };
126
127     // number of queries
128     int q = sizeof(QueryL) / sizeof(QueryL[0]);
129
130     // Function call to get
131     solveQuery(arr, n, QueryL, QueryR, QueryK, q);
132
133     return 0;
134 }

```

## 4.2 Iterative-SegTree

```

1 // Segment Tree Iterativa - Range maximum query
2
3 #define N 100010
4
5 struct Segtree
6 {
7     int t[2*N]={0};
8
9     void build()
10    {
11        for(int i=N-1; i>0; i--)
12            t[i]=max(t[i<<1], t[1<<1|1]);
13    }
14
15    int query(int l, int r)
16    {
17        int ans=0;
18        for(i+=N, r+=N; l<r; l>>=1, r>>=1)
19        {
20            if(l&1)
21                ans=max(ans, t[l++]);
22            if(r&1)
23                ans=max(ans, t[--r]);
24        }
25
26        return ans;
27    }

```

```

28
29 void update(int p, int value)
30 {
31     for(t[p+=n]=value; p>1; p>>=1)
32         t[p>>1]= max(t[p], t[p^1]);
33 }
34
35 };
36
37 int main()
38 {
39     Segtree st;
40
41     for(int i=0;i<n;i++)
42     {
43         cin >> aux;
44         st.t[N+i]=aux; //Leaves are stored in
45         continuous nodes with indices starting with N
46     }
47
48     st.build();
49     x = st.query(inicio, fim);
50     st.update(ind, value);
51 }

```

## 4.3 Recursive-SegTree

```

1 // Segment Tree Recursiva - Range maximum query
2
3 vector<int> val(MAX, 0);
4 vector<int> vet(N);
5
6 void monta(int i, int j, int no)
7 {
8     if(i==j)
9     {
10         val[no]=vet[i];
11         return;
12     }
13
14     int esq = 2*no;
15     int dir = 2*no+1;
16     int meio = (i+j)/2;
17
18     monta(i, meio, esq);
19     monta(meio+1, j, dir);
20
21     val[no]=max(val[esq], val[dir]);
22 }
23
24 void atualiza(int no, int i, int j, int pos, int
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
43         if(val[esq]>val[dir])
44             val[no]=val[esq];
45         else
46             val[no]=val[dir];

```



```

44     }
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 }

```

## 4.4 Delta-Encoding

```

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

```

## 4.5 Seg-Tree-Farao

```

1 typedef struct
2 {
3     pii prefix, sufix, total, maximo;
4 } no;
5
6 int noleft[MAX], noright[MAX]; //Guarda os valores
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 }

```

```

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     m.sufix = max(r.sufix, somar(r.total, l.sufix));
26     m.total = somar(l.total, r.total); //Soma de
27     m.maximo = max(max(l.maximo, r.maximo), somar(l.sufix, r.prefix)); //Resultado para cada
28     return m;
29 }
30
31 no makenozero()
32 {
33     no m;
34     m.prefix=m.sufix=m.total=m.maximo=mp(0,0);
35     return m;
36 }
37
38 no makeno(int k)
39 {
40     no m;
41     m.prefix=m.sufix=m.total=m.maximo=mp(k,1);
42     return m;
43 }
44
45 void monta(int n)
46 {
47     if(noleft[n]==noright[n])
48     {
49         arvore[n]=makeno(v[noleft[n]]);
50         return;
51     }
52
53     int mid = (noleft[n]+noright[n])/2;
54     noleft[2*n]=noleft[n]; noright[2*n]=mid;
55     noleft[2*n+1]=mid+1; noright[2*n+1]=noright[n];
56
57     monta(2*n);
58     monta(2*n+1);
59
60     arvore[n]=une(arvore[2*n], arvore[2*n+1]);
61 }
62
63 no busca(int n, int esq, int dir)
64 {
65     if(noleft[n]>=esq and noright[n]<=dir)
66         return arvore[n];
67     if(noright[n]<esq or noleft[n]>dir)
68         return makenozero();
69
70     return une(busca(2*n, esq, dir), busca(2*n+1, esq, dir));
71 }
72
73 int main()
74 {
75     int T, N, Q, A, B;
76     no aux;
77
78     scanf("%d", &T);
79 }

```

```

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
query
94         aux = busca(1, A, B);
95         printf("%d %d\n", aux.maximo.f, aux.
maximo.s);
96     }
97 }
98
99
100 return 0;
101 }

```

## 4.6 BIT-2D

```

1 // BIT 2D
2
3 int bit[MAX][MAX];
4
5 int sum(int x, int y)
6 {
7     int resp=0;
8
9     for(int i=x;i>0;i-=i&-i)
10         for(int j=y;j>0;j-=j&-j)
11             resp+=bit[i][j];
12
13     return resp;
14 }
15
16 void update(int x, int y, int delta)
17 {
18     for(int i=x;i<MAX;i+=i&-i)
19         for(int j=y;j<MAX;j+=j&-j)
20             bit[i][j]+=delta;
21 }
22
23 int query(int x1, y1, x2, y2)
24 {
25     return sum(x2,y2) - sum(x2,y1) - sum(x1,y2) + sum
(x1,y1);
26 }

```

## 4.7 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 };

```

## 4.8 Sparse-Table

```

1 logv[1] = 0; // pre-computar tabela de log
2 for (int i = 2; i <= MAXN; i++)
3     logv[i] = logv[i/2] + 1;
4
5 int logv[MAXN+1];
6 int st[MAXN][K];
7
8 // operacao da sparse table deve ser idempotente ->
9 op(x, x) = x
10 void precompute(int N) { //
11     for (int i = 0; i < N; i++)
12         st[i][0] = array[i];
13
14     int k = logv[N];
15     for (int j = 1; j <= k; j++)
16         for (int i = 0; i + (1 << j) <= N; i++)
17             st[i][j] = max(st[i][j-1], st[i + (1 << (j
- 1))][j - 1]);
18 }
19
20 int query(int L, int R) {
21     int j = logv[R - L + 1];
22     int maximum = max(st[L][j], st[R - (1 << j) + 1][
j]);
23
24     return maximum;
25 }

```

## 4.9 Union-Find

```

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

```

```

30 {
31     for(int i=1; i<=N; i++)
32         pai[i]=i;
33 }

```

## 4.10 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      bool operator<(Query other) const
6      {
7          return make_pair(l / BLK, r) <
8                 make_pair(other.l / BLK, other.r);
9      }
10 };
11
12 void add(); void remove() // implementar operacoes de
   acordo com o problema, cuidado com TLE ao
   utilizar MAP
13
14 vector<pair<int, ll>> mo() {
15     vector<pair<int, ll>> res;
16     sort(queries.begin(), queries.end());
17
18     int l = 0, r = -1;
19     for(Query q : queries) {
20         while(l > q.l) {
21             l--;
22             add(l);
23         }
24         while(r < q.r) {
25             r++;
26             add(r);
27         }
28         while(l < q.l) {
29             remove(l);
30             l++;
31         }
32         while(r > q.r) {
33             remove(r);
34             r--;
35         }
36         res.pb(mp(q.idx, RESPOSTA)); // adicionar
   resposta de acordo com o problema
37     }
38     return res; // ordenar o vetor pelo indice e
   responder queries na ordem
39 }

```

## 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,
   int &y0, int &g)
17 {
18     g = gcd(abs(a), abs(b), x0, y0);
19     if (c % g)
20         return false;
21
22     x0 *= c / g;
23     y0 *= c / g;
24     if (a < 0) x0 = -x0;

```

```

25     if (b < 0) y0 = -y0;
26     return true;
27 }
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 // 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 }

```

## 5.7 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
78     ll N;
79     cin >> N;
80
81     cout << rabin(N) << endl;
82
83     return 0;
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         } while(d == 1);
50     } while(d == n);
51     while(d == n);
52     return d;
53 }
```

```
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 }
```

```

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

```

## 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 +
    b.x, a.y + b.y); }
8 inline num operator-(num a, num b) { return num(a.x -
    b.x, a.y - b.y); }
9 inline num operator*(num a, num b) { return num(a.x *
    b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
10 inline num conj(num a) { return num(a.x, -a.y); }
11
12 int base = 1;
13 vector<num> roots = {{0, 0}, {1, 0}};
14 vector<int> rev = {0, 1};
15
16 const double PI = acos(-1.0);
17
18 void ensure_base(int nbase){
19     if(nbase <= base)
20         return;
21
22     rev.resize(1 << nbase);
23     for(int i = 0; i < (1 << nbase); i++){
24         rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (
    nbase - 1));
25
26     roots.resize(1 << nbase);
27
28     while(base < nbase){
29         double angle = 2*PI / (1 << (base + 1));
30         for(int i = 1 << (base - 1); i < (1 << base);
    i++){
31             roots[i << 1] = roots[i];
32             double angle_i = angle * (2 * i + 1 - (1
    << base));
33             roots[(i << 1) + 1] = num(cos(angle_i),
    sin(angle_i));
34         }
35         base++;
36     }
37 }
38
39 void fft(vector<num> &a, int n = -1){
40     if(n == -1)
41         n = a.size();
42
43     assert((n & (n-1)) == 0);
44     int zeros = __builtin_ctz(n);
45     ensure_base(zeros);

```

```

46     int shift = base - zeros;
47     for(int i = 0; i < n; i++){
48         if(i < (rev[i] >> shift))
49             swap(a[i], a[rev[i] >> shift]);
50
51     for(int k = 1; k < n; k <= 1)
52         for(int i = 0; i < n; i += 2 * k)
53             for(int j = 0; j < k; j++){
54                 num z = a[i+j+k] * roots[j+k];
55                 a[i+j+k] = a[i+j] - z;
56                 a[i+j] = a[i+j] + z;
57             }
58 }
59
60 vector<num> fa, fb;
61 vector<int> multiply(vector<int> &a, vector<int> &b){
62     int need = a.size() + b.size() - 1;
63     int nbase = 0;
64     while((1 << nbase) < need) nbase++;
65     ensure_base(nbase);
66     int sz = 1 << nbase;
67     if(sz > (int) fa.size())
68         fa.resize(sz);
69
70     for(int i = 0; i < sz; i++){
71         int x = (i < (int) a.size() ? a[i] : 0);
72         int y = (i < (int) b.size() ? b[i] : 0);
73         fa[i] = num(x, y);
74     }
75     fft(fa, sz);
76     num r(0, -0.25 / sz);
77     for(int i = 0; i <= (sz >> 1); i++){
78         int j = (sz - i) & (sz - 1);
79         num z = (fa[j] * fa[j] - conj(fa[i] * fa[i]))
    * r;
80         if(i != j) {
81             fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[
    j])) * r;
82         }
83         fa[i] = z;
84     }
85     fft(fa, sz);
86     vector<int> res(need);
87     for(int i = 0; i < need; i++)
88         res[i] = fa[i].x + 0.5;
89
90     return res;
91 }
92
93 int main()
94 {sws;
95
96     //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 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 }

```

## 5.15 Recursao-linear

```

1 vector<vl> mult(vector<vl> a, vector<vl> b, int n) {
2     vector<vl> res;
3     for(int i = 0; i < n; i++) {
4         vl tmp;
5         for(int j = 0; j < n; j++) {
6             tmp.pb(0);
7         }
8         res.pb(tmp);
9     }
10
11     for(int row = 0; row < n; row++) {
12         for(int col = 0; col < n; col++) {
13             ll val = 0;
14             for(int k = 0; k < n; k++) {
15                 val += (a[row][k]*b[k][col]);
16             }
17             res[row][col] = val;
18         }
19     }
20
21     return res;
22 }
23
24 vector<vl> fexp(vector<vl> b, ll e, int n) {
25     if(e == 0) {
26         vector<vl> id;
27         for(int i = 0; i < n; i++) {
28             vl tmp;
29             for(int j = 0; j < n; j++) {
30                 if(i == j)
31                     tmp.pb(1);
32                 else
33                     tmp.pb(0);
34             }

```

```

35             id.pb(tmp);
36         }
37
38         return id;
39     }
40
41     vector<vl> res = fexp(b, e/2, n);
42     res = mult(res, res, n);
43
44     if(e%2)
45         res = mult(res, b, n);
46
47     return res;
48 }
49
50 // k = tamanho da recorrência/matriz, n = n-esimo
51 // termo
52 // f(n) = c1*f(n-1) + c2*f(n-2) + ... + ck*f(n-k)
53 // base -> [f(k-1), f(k-2), ..., f(0)]
54 // coeficientes -> [c1, c2, ..., ck]
55 vl solve(int k, int n, vl base, vl coef) {
56     vector<vl> inicial;
57     inicial.pb(coef);
58     for(int row = 0; row < k-1; row++) {
59         vl tmp;
60         for(int col = 0; col < k; col++) {
61             if(col == row)
62                 tmp.pb(1);
63             else
64                 tmp.pb(0);
65         }
66         inicial.pb(tmp);
67     }
68
69     vector<vl> matexp = fexp(inicial, max(0, n-k+1),
70                             k);
71     vl res(k);
72
73     for(int row = 0; row < k; row++) {
74         ll val = 0;
75         for(int aux = 0; aux < k; aux++) {
76             val += matexp[row][aux]*base[aux];
77         }
78         res[row] = val; // res = (f(n), f(n-1), ...,
79                             f(n-k+1))
80     }
81
82     return res;
83 }

```

## 5.16 Kamenetsky

```

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

```

## 6 Misc

### 6.1 LIS

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

### 6.2 Bitwise

```
1 // Bitwise
2
3 unsigned char a = 5, b = 9; // a = (00000101), b
  = (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)
13 Left shift -   b<<1   // The result is 00010010
14 (18)
15 Right shift -  b>>1   // The result is 00000100
16 (4)
17
18 // Exchange two int variables
19
20     a^=b;
21     b^=a;
22     a^=b;
23
24 // Even or Odd
25
26     (x & 1)? printf("Odd"): printf("Even");
27
28 // Turn on the j-th bit
29
30     int S = 34; //(100010)
31     int j = 3;
32
33     S = S | (1<<j);
34
35 // Turn off the j-th bit
36
37     int S = 42; //(101010)
38     int j = 1;
39
40     S &= ~(1<<j)
41
42     S == 40 //(101000)
43
44 // Check the j-th element
45
46     int S = 42; //(101010)
47     int j = 3;
48
49     T = S & (1<<j); // T = 0
50
51 // Least significant bit (lsb)
52
53     int lsb(int x){ return x&-x; }
54
55 // 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); }
```

### 6.3 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 KMP

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

### 7.2 LCS

```
1 string LCSUBSTR(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 }
```

### 7.3 Pal-int

```
1 bool ehpalindromo(ll n)
2 {
3     if(n<0)
4         return false;
5
6     int divisor = 1;
7     while(n/divisor >= 10)
8         divisor *= 10;
9
10    while(n != 0)
11    {
12        int leading = n / divisor;
13        int trailing = n % 10;
14
15        if(leading != trailing)
16            return false;
17
18        n = (n % divisor)/10;
19        divisor = divisor/100;
20    }
21
22    return true;
23 }
24 }
```

### 7.4 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 }
```

### 7.5 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 }
```