



Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos	2	4 ED	9
1.1 Mochila	2	4.1 Trie	9
1.2 Unbounded-Knapsack	2	4.2 Range-query-bigger-than-k-BIT	9
1.3 Kadane-DP	2	4.3 Prefixsum2D	10
1.4 Iterative-BS	2	4.4 Iterative-SegTree	10
2 Grafos	2	4.5 Recursive-SegTree	11
2.1 BFS	2	4.6 Seg-Tree-MaxSubArray	11
2.2 Find-bridges	2	4.7 Delta-Encoding	12
2.3 Dijkstra	2	4.8 BIT-2D	12
2.4 LCA	3	4.9 BIT	12
2.5 Floyd-Warshall	3	4.10 BIT-kth	12
2.6 Kruskal	3	4.11 DQUERY-Seg-Persistente	12
2.7 DFS	4	4.12 Sparse-Table	13
2.8 Topological-sort	4	4.13 Union-Find	13
2.9 Kosaraju	4	4.14 CHT	14
2.10 Centroid	4	4.15 MKTHNUM-Seg-Persistente	14
2.11 Prim	4	4.16 Mo	15
3 Geometria	5	5 Math	15
3.1 Inside-polygon	5	5.1 Totient	15
3.2 MinDistPair	5	5.2 Sqrt-BigInt	15
3.3 Center-polygon	5	5.3 Linear-Diophantine-Equation	15
3.4 Intersect-polygon	5	5.4 Factorization-sqrt	16
3.5 Sort-by-Angle	5	5.5 FFT-simple	16
3.6 Convex-Hull	6	5.6 Modular-Exponentiation	16
3.7 Inter-Retangulos	6	5.7 Lagrange-interpolation	16
3.8 Tetrahedron-Distance3D	6	5.8 Miller-Habin	17
3.9 3D	7	5.9 Inverso-Mult	17
3.10 Heron	7	5.10 Pollard-Rho	17
3.11 Uniao-segmentos	7	5.11 Verif-primo	18
3.12 Minkowski-Sum	8	5.12 Crivo	18
3.13 Simetria-central	8	5.13 Bigmod	18
3.14 2D	8	5.14 Simpson's-formula	18
		5.15 FFT-tourist	18
		5.16 Next-Permutation	20
		5.17 Fast-Exponentiation	20
		5.18 Recursao-linear	20
		5.19 Raiz-primitiva	20

5.20	Kamenetsky	21
6	Misc	21
6.1	2SAT	21
6.2	LIS	21
6.3	Bitwise	21
6.4	All-Subsets	22
6.5	Template	22
6.6	Rand	22
6.7	Meet-in-the-middle	23
7	Strings	23
7.1	Trie	23
7.2	KMP	23
7.3	Suffix-array	23
7.4	LCS	23
7.5	Pal-int	24
7.6	Z-Func	24
7.7	Hash	24
7.8	Manacher	24

1 Algoritmos

1.1 Mochila

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

1.2 Unbounded-Knapsack

```
1 int w, n;
2 int c[MAX], v[MAX];
3
4 int unbounded_knapsack()
5 {
6
7     int dp[w+1];
8     memset(dp, 0, sizeof dp);
9
10    for(int i=0; i<=w; i++)
11        for(int j=0; j<n; j++)
12            if(c[j] <= i)
13                dp[i] = max(dp[i], dp[i-c[j]] + v[j]);
14
15    return dp[w];
16 }
17 }
```

1.3 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 }
```

1.4 Iterative-BS

```
1 int l=1, r=N;
2 int res=-1;
3
4 while(l<=r){
5     int m = (l+r)/2;
6     if(!ver(m)){
7         l = m+1;
8     }
9 }
```

```
9     else{
10         res = m;
11         r = m-1;
12     }
13 }
14 cout << res << endl;
```

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 int n;
2 vector<vi> adj(n+1, vi());
3
4 vector<bool> visited;
5 vi tin, low;
6 int timer;
7
8 void dfs(int v, int p=-1){
9     visited[v] = true;
10    tin[v] = low[v] = timer++;
11    for (int to: adj[v]){
12        if(to == p) continue;
13        if(visited[to])
14            low[v] = min(low[v], tin[to]);
15        else{
16            dfs(to, v);
17            low[v] = min(low[v], low[to]);
18            if(low[to] > tin[v])
19                IS_BRIDGE(v, to);
20        }
21    }
22 }
23
24 void find_bridges(){
25     timer = 0;
26     visited.assign(n, false);
27     tin.assign(n, -1);
28     low.assign(n, -1);
29     for(int i=0; i<n; i++)
30         if(!visited[i])
31             dfs(i);
32 }
```

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, 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-comutar 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][
26 j]);
27     return minimum;
28 }

```

```

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

```

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 }

```

```

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     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){

```

```

6     vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
7     S.push_back(u);
8 }
9 void dfst(int u, int e){
10     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 int sz(int u, int p) {
6     int s = 1;
7     for(auto prox : g[u]) {
8         if(prox != p and !erased[prox])
9             s += sz(prox, u);
10    }
11    return size[u] = s;
12 }
13
14 int centroid(int u, int p, int n) {
15     // chamar funcao sz antes, n = size[u]
16     for(auto prox : g[u]) {
17         if(prox != p and !erased[prox]) {
18             if(size[prox] > n/2) {
19                 return centroid(prox, u, n);
20             }
21         }
22     }
23     return u;
24 }

```

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]){

```

```

26         davez = atual;
27         break;
28     }
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(ccw(vet[0], vet[mid], ext)==1)
8             l=mid+1;
9         else{
10             r=mid;
11             res=mid;
12         }
13     }
14     int a = ccw(vet[0], vet[res-1], ext);
15     int b = ccw(vet[res-1], vet[res], ext);
16     int c = ccw(vet[res], vet[0], ext);
17

```

```

18     if((a==1 or b==1 or c==1) and (a==-1 or b==-1 or
c==-1)) return false;
19     else return true;
20 }

```

3.2 MinDistPair

```

1 ll MinDistPair(vp &vet){
2     int n = vet.size();
3     sort(vet.begin(), vet.end());
4     set<point> s;
5
6     ll best_dist = LLINF;
7     int j=0;
8     for(int i=0; i<n; i++){
9         int d = ceil(sqrt(best_dist));
10        while(j<n and vet[i].x-vet[j].x >= d){
11            s.erase(point(vet[j].y, vet[j].x));
12            j++;
13        }
14
15        auto it1 = s.lower_bound({vet[i].y - d, vet[i
].x});
16        auto it2 = s.upper_bound({vet[i].y + d, vet[i
].x});
17
18        for(auto it=it1; it!=it2; it++){
19            ll dx = vet[i].x - it->y;
20            ll dy = vet[i].y - it->x;
21            if(best_dist > dx*dx + dy*dy){
22                best_dist = dx*dx + dy*dy;
23                // vet[i] e inv(it)
24            }
25        }
26
27        s.insert(point(vet[i].y, vet[i].x));
28    }
29    return best_dist;
30 }

```

3.3 Center-polygon

```

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

```

3.4 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 }

```

3.5 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);

```

3.6 Convex-Hull

```

1 vp convex_hull(const vp points)
2 {
3     vp P(points);
4     sort(P.begin(), P.end());
5     vp lower, upper;
6     for(const auto& p: P){
7         int n = int(lower.size());
8         while(n>=2 and ccw(lower[n-2], lower[n-1], p)
9             !=-1){
10             lower.pop_back();
11             n = int(lower.size());
12         }
13         lower.push_back(p);
14     }
15     reverse(P.begin(), P.end());
16     for(const auto& p: P){
17         int n = int(upper.size());
18         while(n>=2 and ccw(upper[n-2], upper[n-1], p)
19             !=-1){
20             upper.pop_back();
21             n = int(upper.size());
22         }
23         upper.push_back(p);
24     }
25     lower.pop_back();
26     lower.insert(lower.end(), upper.begin(), upper.
27         end()-1);
28     return lower;
29 }

```

3.7 Inter-Retangulos

```

1 bool doOverlap(point l1, point r1, point l2, point r2
2 )
3 {
4     if (l1.x>r2.x or l2.x>r1.x or l1.y<r2.y or l2.y<
5         r1.y)
6         return false;
7     return true;
8 }

```

3.8 Tetrahedron-Distance3D

```

1 bool nulo(point a){
2     return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0))
3 ;
4 }
5 ld misto(point p1, point p2, point p3){

```

```

6     return (p1^p2)*p3;
7 }
8
9 ld dist_pt_face(point p, vp v){
10     assert(v.size()==3);
11
12     point v1 = v[1]-v[0];
13     point v2 = v[2]-v[0];
14     point n = (v1^v2);
15
16     for(int i=0;i<3;i++){
17         point va = p-v[i];
18         point vb = v[(i+1)%3]-v[i];
19         point ve = vb^n;
20         ld d = ve*v[i];
21         //se ponto coplanar com um dos lados do
22         //prisma (va^vb eh nulo),
23         //ele esta dentro do prisma (poderia
24         //desconsiderar pois distancia
25         //vai ser a msm da distancia do ponto ao
26         //segmento)
27         if(!nulo(va^vb) and (v[(i+2)%3]*ve>d) ^ (p*ve
28             >d)) return LLINF;
29     }
30
31     //se ponto for coplanar ao triangulo (e dentro do
32     //triangulo)
33     //vai retornar zero corretamente
34     return fabs(misto(p-v[0],v1,v2)/norm(n));
35 }
36
37 ld dist_pt_seg(point p, vp li){
38     return norm((li[1]-li[0])^(p-li[0]))/norm(li[1]-
39         li[0]);
40 }
41
42 ld dist_line(vp l1, vp l2){
43     point n = (l1[1]-l1[0])^(l2[1]-l2[0]);
44     if(nulo(n)) //retas paralelas - dist ponto a reta
45         return dist_pt_seg(l2[0],l1);
46
47     point o1o2 = l2[0]-l1[0];
48     return fabs((o1o2*n)/norm(n));
49 }
50
51 // retas paralelas e intersecao nao nula
52 ld dist_seg(vp l1, vp l2){
53     assert(l2.size()==2);
54     assert(l1.size()==2);
55
56     //pontos extremos do segmento
57     ld ans = LLINF;
58     for(int i=0;i<2;i++){
59         for(int j=0;j<2;j++){
60             ans = min(ans, norm(l1[i]-l2[j]));
61         }
62     }
63
64     //verificando distancia de ponto extremo com
65     //ponto interno dos segs
66     for(int t=0;t<2;t++){
67         for(int i=0;i<2;i++){
68             for(int k=0;k<2;k++){
69                 point va = l1[i]-l2[k];
70                 point vb = l2[!k]-l2[k];
71                 ld ang = atan2(norm((vb^va)), vb*va);
72                 if(ang>PI/2) c = false;
73             }
74             if(c)
75                 ans = min(ans,dist_pt_seg(l1[i],l2));
76         }
77     }
78     swap(l1,l2);
79 }

```

```

72 //ponto interno com ponto interno dos segmentos
73 point v1 = l1[1]-l1[0], v2 = l2[1]-l2[0];
74 point n = v1^v2;
75 if(!nulo(n)){
76     bool ok = true;
77     for(int t=0;t<2;t++){
78         point n2 = v2^v1;
79         point o1o2 = l2[0]-l1[0];
80         ld escalar = (o1o2*n2)/(v1*n2);
81         if(escalar<0 or escalar>1) ok = false;
82         swap(l1,l2);
83         swap(v1,v2);
84     }
85     if(ok) ans = min(ans,dist_line(l1,l2));
86 }
87
88 return ans;
89 }
90
91 ld ver(vector<vp> &vet){
92     ld ans = LLINF;
93     // vertice - face
94     for(int k=0;k<2;k++){
95         for(int pt=0;pt<4;pt++){
96             for(int i=0;i<4;i++){
97                 vp v;
98                 for(int j=0;j<4;j++){
99                     if(i!=j) v.pb(vet[!k][j]);
100                 }
101                 ans = min(ans, dist_pt_face(vet[k][pt], v));
102             }
103         }
104     }
105     // edge - edge
106     for(int i1=0;i1<4;i1++){
107         for(int j1=0;j1<i1;j1++){
108             for(int i2=0;i2<4;i2++){
109                 for(int j2=0;j2<i2;j2++){
110                     ans = min(ans, dist_seg({vet[0][i1], vet[0][j1]}, {vet[1][i2], vet[1][j2]}));
111                 }
112             }
113         }
114     }
115     return ans;
116 }

```

3.9 3D

```

1 // typedef int cod;
2 // bool eq(cod a, cod b){ return (a==b); }
3
4 #define vp vector<point>
5 typedef ld cod;
6 bool eq(cod a, cod b){ return fabs(a - b) <= EPS; }
7
8 struct point
9 {
10     cod x, y, z;
11     point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z) {}
12
13     point operator+(const point &o) const{
14         return {x+o.x, y+o.y, z+o.z};
15     }
16     point operator-(const point &o) const{
17         return {x-o.x, y-o.y, z-o.z};
18     }
19     point operator*(cod t) const{
20         return {x*t, y*t, z*t};
21     }
22     point operator/(cod t) const{
23         return {x/t, y/t, z/t};
24     }

```

```

25     bool operator==(const point &o) const{
26         return eq(x, o.x) and eq(y, o.y) and eq(z, o.z);
27     }
28     cod operator*(const point &o) const{ // dot
29         return x*o.x + y*o.y + z*o.z;
30     }
31     point operator^(const point &o) const{ // cross
32         return point(y*o.z - z*o.y,
33                     z*o.x - x*o.z,
34                     x*o.y - y*o.x);
35     }
36 };
37
38 ld dist(point a, point b){
39     return sqrt((a-b)*(a-b));
40 }
41 bool nulo(point a){
42     return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0));
43 }
44
45 ld norm(point a){ // Modulo
46     return sqrt(a*a);
47 }
48 ld proj(point a, point b){ // a sobre b
49     return (a*b)/norm(b);
50 }
51 ld angle(point a, point b){ // em radianos
52     return acos((a*b) / norm(a) / norm(b));
53 }
54
55 cod triple(point a, point b, point c){
56     return dot(a, b^c); // Area do paralelepipedo
57 }
58
59 struct plane{
60     point p1, p2, p3;
61     plane(point p1=0, point p2=0, point p3=0): p1(p1), p2(p2), p3(p3){}
62
63     point aux = (p1-p3)^(p2-p3);
64     cod a = aux.x, b = aux.y, c = aux.z;
65     cod d = -a*p1.x - b*p1.y - c*p1.z;
66     // ax+by+cz+d = 0;
67 };
68
69 cod dist(plane pl, point p){
70     return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
71 }
72 }

```

3.10 Heron

```

1 ld heron(int a, int b, int c){
2     ld s = (a+b+c)/2.0;
3     return sqrtl(s*(s-a)*(s-b)*(s-c));
4 }
5
6 ld heron(int a, int b, int c, int d){
7     ld s = (a+b+c+d)/2.0;
8     return sqrtl((s-a)*(s-b)*(s-c)*(s-d));
9 }

```

3.11 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 }

```

3.12 Minkowski-Sum

```

1 vp mk(const vp &a,const vp &b){
2     int i = 0, j = 0;
3     for(int k = 0; k < (int)a.size(); k++){if(a[k]<a[i
4     ])
5         i = k;
6     for(int k = 0; k < (int)b.size(); k++){if(b[k]<b[j
7     ])
8         j = k;
9     vp c;
10    c.reserve(a.size() + b.size());
11    for(int k = 0; k < int(a.size()+b.size()); k++){
12        point pt{a[i] + b[j]};
13        if((int)c.size() >= 2 and !ccw(c[c.size()-2],
14        c.back(), pt))
15            c.pop_back();
16        c.pb(pt);
17        int q = i+1, w = j+1;
18        if(q == int(a.size())) q = 0;
19        if(w == int(b.size())) w = 0;
20        if(ccw(c.back(), a[i]+b[w], a[q]+b[j]) < 0) i
21        = q;
22        else j = w;
23    }
24    if(!ccw(c[0], c[(int)c.size()-1], c[(int)c.size()
25    -2]))
26        c.pop_back();
27    if(!ccw(c.back(), c[0], c[1])){
28        c[0]=c.back();
29        c.pop_back();
30    }
31    c.shrink_to_fit();
32    return c;
33 }

```

3.13 Simetria-central

```

1 bool simetric(vector<point> &a){ //ordered
2     int n = a.size();
3     c = center(a);
4     if(n&1) return false;
5     for(int i=0;i<n/2;i++)
6         if(!collinear(a[i], a[i+n/2], c))
7             return false;
8     return true;
9 }

```

3.14 2D

```

1 // typedef int cod;
2 // bool eq(cod a, cod b){ return (a==b); }

```

```

3
4 #define vp vector<point>
5 typedef ld cod;
6 bool eq(cod a, cod b){ return fabs1(a - b) <= EPS; }
7
8 struct point{
9     cod x, y;
10    int id;
11    point(cod x=0, cod y=0): x(x), y(y){}
12
13
14    point operator+(const point &o) const{
15        return {x+o.x, y+o.y};
16    }
17    point operator-(const point &o) const{
18        return {x-o.x, y-o.y};
19    }
20    point operator*(cod t) const{
21        return {x*t, y*t};
22    }
23    point operator/(cod t) const{
24        return {x/t, y/t};
25    }
26    cod operator*(const point &o) const{ // dot
27        return x * o.x + y * o.y;
28    }
29    cod operator^(const point &o) const{ // cross
30        return x * o.y - y * o.x;
31    }
32    bool operator<(const point &o) const{
33        if(!eq(x, o.x)) return x < o.x;
34        return y < o.y;
35    }
36    bool operator==(const point &o) const{
37        return eq(x, o.x) and eq(y, o.y);
38    }
39 };
40
41 bool nulo(point a){
42     return (eq(a.x, 0) and eq(a.y, 0));
43 }
44
45 ld norm(point a){ // Modulo
46     return sqrt(a*a);
47 }
48 ld proj(point a, point b){ // a sobre b
49     return a*b/norm(b);
50 }
51 ld max(ld a, ld b){ return (a>b ? a:b); }
52 ld min(ld a, ld b){ return (a<b ? a:b); }
53 ld angle(point a, point b){ // em radianos
54     ld ang = a*b / norm(a) / norm(b);
55     return acos(max(min(ang, 1), -1));
56 }
57 ld angle_vec(point v){
58     // return 180/PI*atan2(v.x, v.y);
59     return atan2(v.x, v.y);
60 }
61 int ccw(point &a, point &b, point &c){ //-1=dir; 0=
62     collinear; 1=esq;
63     cod tmp = (b-a)^(c-a); // from a to b
64     return (tmp > EPS) - (tmp < -EPS);
65 }
66 ld order_angle(point a, point b){ // from a to b ccw
67     (a in front of b)
68     ld aux = angle(a,b)*180/PI;
69     return ((a^b)<=0 ? aux:360-aux);
70 }
71
72 bool collinear(point a, point b, point c){
73     return eq((a-c)^(b-c), 0);
74 }

```

```

74 point rotccw(point p, ld a){
75     // a = PI*a/180; // graus
76     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
77     +p.x*sin(a)));
78 }
79 point rot90cw(point a) { return point(a.y, -a.x); };
80 point rot90ccw(point a) { return point(-a.y, a.x); };
81
82 // Area de um poligono (pontos ordenados por
83     adjacencia)
84 ld area(vp &p){
85     ld ret = 0;
86     for(int i=2;i<(int)p.size();i++)
87         ret += (p[i]-p[0])^(p[i-1]-p[0]);
88     return fabs1(ret/2);
89 }
90
91
92 struct line{
93     point p1, p2;
94     line(point p1=0, point p2=0): p1(p1), p2(p2){}
95
96     cod a = p1.y-p2.y;
97     cod b = p2.x-p1.x;
98     cod c = -(a*p1.x + b*p1.y);
99     // ax+by+c = 0;
100
101     bool inside(point p){
102         return eq(norm(p1-p)+norm(p2-p) - norm(p2-p1)
103         , 0);
104     }
105 };
106
107 point intersection(line l1, line l2){
108     ld det = l1.a*l2.b - l1.b*l2.a;
109     if(det==0) return point(INF, INF);
110     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
111     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
112     return point(x, y);
113 }
114
115 // Dist entre ponto e segmento de reta
116 cod distr(point p, point a, point b){
117     if(((b-a)^(p-b)) > 0)
118         return norm(p-b);
119     if(((a-b)^(p-a)) > 0)
120         return norm(p-a);
121     return fabs((b-a)^(p-a))/norm(a-b);
122 }
123
124 struct circle{
125     point c;
126     cod r;
127     circle(point c=0, cod r=0): c(c), r(r){}
128 };

```

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 // Returns the count of numbers of elements
36 // present from starting till idx.
37 int query(int* BIT, int idx){
38     int ans = 0;
39     while (idx){
40         ans += BIT[idx];
41
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
90             // call function to returns answer for
91             // each query
92             int cnt = query(BIT, a[i].r) - query(BIT, a[i].l - 1);

```

```

91         a[i].l - 1);
92
93         // This will ensure that answer of each
94         // query
95         // are stored in order it was initially
96         // asked.
97         ans[a[i].pos] = cnt;
98     }
99     else{
100         // a[i].r contains the position of the
101         // element in the original array.
102         update(BIT, n, a[i].r);
103     }
104 }
105 // Output the answer array
106 for (int i = 1; i <= q; ++i){
107     cout << ans[i] << endl;
108 }
109
110 // Driver Code
111 int main()
112 {
113     int arr[] = { 7, 3, 9, 13, 5, 4 };
114     int n = sizeof(arr) / sizeof(arr[0]);
115
116     // 1-based indexing
117     int QueryL[] = { 1, 2 };
118     int QueryR[] = { 4, 6 };
119
120     // k for each query
121     int QueryK[] = { 6, 8 };
122
123     // number of queries
124     int q = sizeof(QueryL) / sizeof(QueryL[0]);
125
126     // Function call to get
127     solveQuery(arr, n, QueryL, QueryR, QueryK, q);
128
129     return 0;
130 }

```

4.3 Prefixsum2D

```

1 11 find_sum(vector<vi> &mat, int a, int b, int c, int
2     d){
3     // superior-esq(c,d) (a,b)inferior-dir
4     return mat[a][b]-mat[a][d-1]-mat[c-1][b]+mat[c-1][d-1];
5 }
6
7 int main(){
8     for(int i=1;i<=n;i++)
9         for(int j=1;j<=n;j++)
10             mat[i][j]+=mat[i-1][j]+mat[i][j-1]-mat[i-1][j-1];
11
12 }

```

4.4 Iterative-SegTree

```

1 // Segment Tree Iterativa - Max
2
3 struct Segtree{
4     vi t;
5     int n;
6
7     Segtree(int n){
8         this->n = n;
9         t.assign(2*n, 0);
10    }

```

```

11 void build(){
12     for(int i=n-1; i>0; i--){
13         t[i]=max(t[i<<1], t[i<<1|1]);
14     }
15 }
16
17 int query(int l, int r){ // idx 0
18     int ans=0;
19     for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
20         if(l&1)
21             ans=max(ans, t[l++]);
22         if(r&1)
23             ans=max(t[--r], ans);
24     }
25     return ans;
26 }
27
28 void update(int p, int value){
29     for(t[p+=n]=value; p >>= 1;){
30         t[p] = max(t[p<<1], t[p<<1|1]);
31     }
32 };
33
34 int main()
35 {
36     Segtree st(n);
37
38     for(int i=0; i<n; i++){
39         cin >> aux;
40         st.t[n+i]=aux; //Leaves are stored in
41         continuous nodes with indices starting with N
42     }
43
44     st.build();
45     x = st.query(inicio, fim);
46     st.update(ind, value);
47 }
48 }

```

4.5 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){
25         val[no]=novo_valor;
26     }else{
27         int esq = 2*no;
28         int dir = 2*no+1;
29         int meio = (i+j)/2;
30
31         if(pos<=meio)

```

```

31         atualiza(esq, i, meio, pos, novo_valor);
32     else
33         atualiza(dir, meio+1, j, pos, novo_valor)
34     };
35
36     if(val[esq]>val[dir])
37         val[no]=val[esq];
38     else
39         val[no]=val[dir];
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 }

```

4.6 Seg-Tree-MaxSubArray

```

1 // Subarray with maximum sum
2 struct no{
3     ll pref, suff, total, best;
4     no(ll x=0): pref(x), suff(x), total(x), best(x){}
5 };
6
7 no merge(no l, no r){
8
9     no ans;
10    ans.pref = max(0LL, max(l.pref, l.total+r.pref));
11    ans.suff = max(0LL, max(r.suff, l.suff+r.total));
12    ans.total = l.total+r.total;
13    ans.best = max(max(l.best, r.best), l.suff+r.pref);
14    return ans;
15 }
16
17 struct Segtree{
18     vector<no> t;
19     int n;
20
21     Segtree(int n){
22         this->n = n;
23         t.assign(2*n, no(0));
24     }
25
26     void build(){
27         for(int i=n-1; i>0; i--){

```

```

28         t[i]=merge(t[i<<1], t[i<<1|1]);
29     }
30
31     no query(int l, int r){ // idx 0
32         no a(0), b(0);
33         for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
34             if(l&1)
35                 a=merge(a, t[l++]);
36             if(r&1)
37                 b=merge(t[--r], b);
38         }
39         return merge(a, b);
40     }
41
42     void update(int p, int value){
43         for(t[p+=n] = no(value); p >>= 1;){
44             t[p] = merge(t[p<<1], t[p<<1|1]);
45         }
46     }
47 };

```

4.7 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 }

```

4.8 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 }

```

4.9 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 };

```

4.10 BIT-kth

```

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 kth(int x){
11        int resp = 0;
12        x--;
13        for(int i=26;i>=0;i--){
14            if(resp + (1<<i) >= n) continue;
15            if(bit[resp + (1<<i)] <= x){
16                x -= bit[resp + (1<<i)];
17                resp += (1<<i);
18            }
19        }
20        return resp + 1;
21    }
22
23    void upd(int pos, int val){
24        for(int i = pos; i < n; i += (i&-i))
25            bit[i] += val;
26    }
27 };

```

4.11 DQUERY-Seg-Persistente

```

1 // distinct elements in the range [l, r]
2 struct node{
3     int val;
4     int l, r;
5     node(int a=-1, int b=-1, int c=0){
6         val=c;l=a;r=b;
7     }
8 };
9
10 node tree[8600010]; // nlog(n) space = 8600010
11 int idx=0;
12
13 int build(int l, int r){
14     if(l==r)
15         return idx++;

```

```

16
17     int mid = (l+r)/2;
18
19     tree[idx].l = build(l, mid);
20     tree[idx].r = build(mid+1, r);
21
22     return idx++;
23 }
24
25 int update(int l, int r, int root, int e, int o){
26     int plus=0;
27     if((l>e or r<e) and (l>o or r<o or o==-1))
28         return root;
29     if(l<=e and e<=r) plus++;
30     if(l<=o and o<=r and o!=-1) plus--;
31
32     if(l==e and r==e){
33         tree[idx]=node(-1, -1, 1);
34         return idx++;
35     }
36     if(l==o and r==o){
37         tree[idx]=node(-1, -1, 0);
38         return idx++;
39     }
40     int mid = (l+r)/2;
41     tree[idx]=node(update(l, mid, tree[root].l, e, o)
42         ,
43         update(mid+1, r, tree[root].r, e,
44         o), tree[root].val+plus);
45     return idx++;
46 }
47
48 int query(int a, int b, int l, int r, int root){
49     if(l>b or r<a)
50         return 0;
51     if(l<=a and b<=r)
52         return tree[root].val;
53     int mid = (a+b)/2;
54
55     return query(a, mid, l, r, tree[root].l) + query(
56     mid+1, b, l, r, tree[root].r);
57 }
58
59 int main()
60 {sws;
61
62     int n, m, a, b;
63     int v[MAX], aux[MAX];
64     int root[MAX];
65
66     cin >> n >> m;
67
68     for(int i=0;i<n;i++){
69         cin >> v[i]; aux[i]=v[i];
70     }
71
72     sort(v, v+n);
73
74     map<int, int> comp;
75     for(int i=0, j=0;i<n;i++){
76         if(i==0 or v[i]!=v[i-1])
77             comp[v[i]]=j++;
78     }
79
80     root[0]=build(0, n-1);
81
82     int last[MAX];
83     memset(last, -1, sizeof(last));
84
85     for(int i=0;i<n;i++){
86         root[i+1] = update(0, n-1, root[i], i, last[
87         comp[aux[i]]]);
88         last[comp[aux[i]]]=i;
89     }
90 }

```

```

85     }
86
87     for(int i=0;i<m;i++){
88         cin >> a >> b;
89         cout << query(0, n-1, a-1, n-1, root[b]) <<
90         endl;
91     }
92     return 0;
93 }

```

4.12 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 };

```

4.13 Union-Find

```

1 struct DSU {
2     int n;
3     vi 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 parent[v]=find(parent[v]);
18    }
19
20    void join(int a, int b) {

```

```

21     a = find(a);
22     b = find(b);
23     if(a!=b) {
24         if(size[a]<size[b])
25             swap(a, b);
26
27         parent[b]=a;
28         size[a]+=size[b];
29     }
30 }
31 };

```

4.14 CHT

```

1  const ll is_query = -LLINF;
2  struct Line{
3      ll m, b;
4      mutable function<const Line*> succ;
5      bool operator<(const Line& rhs) const{
6          if(rhs.b != is_query) return m < rhs.m;
7          const Line* s = succ();
8          if(!s) return 0;
9          ll x = rhs.m;
10         return b - s->b < (s->m - m) * x;
11     }
12 };
13 struct Cht : public multiset<Line>{ // maintain max
14     bool bad(iterator y){
15         auto z = next(y);
16         if(y == begin()){
17             if(z == end()) return 0;
18             return y->m == z->m && y->b <= z->b;
19         }
20         auto x = prev(y);
21         if(z == end()) return y->m == x->m && y->b <=
22             x->b;
23         return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)
24             (y->b - z->b)*(y->m - x->m);
25     }
26     void insert_line(ll m, ll b){
27         auto y = insert({ m, b });
28         y->succ = [=]{ return next(y) == end() ? 0 :
29             &*next(y); };
30         if(bad(y)){ erase(y); return; }
31         while(next(y) != end() && bad(next(y))) erase
32             (next(y));
33         while(y != begin() && bad(prev(y))) erase(
34             prev(y));
35     }
36     ll eval(ll x){
37         auto l = *lower_bound((Line) { x, is_query })
38         ;
39         return l.m * x + l.b;
40     }
41 };

```

4.15 MKTHNUM-Seg-Persistente

```

1  // kth number in range [l, r] if it was ordered
2  struct node{
3      int val;
4      int l, r;
5      node(int a=-1, int b=-1, int c=0){
6          val=c;l=a;r=b;
7      }
8  };
9
10 node tree[8600010]; // 4*nlog(4*n) space = 8600010
11 int idx=0;
12
13 int build(int l, int r){
14     if(l==r)

```

```

15         return idx++;
16
17     int mid = (l+r)/2;
18
19     tree[idx].l = build(l, mid);
20     tree[idx].r = build(mid+1, r);
21
22     return idx++;
23 }
24
25 int update(int l, int r, int root, int e){
26     if(l>e or r<e)
27         return root;
28     if(l==e and r==e){
29         tree[idx]=node(-1, -1, tree[root].val+1);
30         return idx++;
31     }
32     int mid = (l+r)/2;
33     tree[idx]=node(update(l, mid, tree[root].l, e),
34         update(mid+1, r, tree[root].r, e),
35         tree[root].val+1);
36     return idx++;
37 }
38 int query(int l, int r, int root1, int root2, int k){
39     while(l!=r)
40     {
41         int mid=(l+r)/2;
42         if(tree[tree[root2].l].val-tree[tree[root1].l
43             ].val>=k)
44         {
45             r = mid;
46             root1 = tree[root1].l;
47             root2 = tree[root2].l;
48         }else
49         {
50             l = mid+1;
51             k-=tree[tree[root2].l].val-tree[tree[
52                 root1].l].val;
53             root1 = tree[root1].r;
54             root2 = tree[root2].r;
55         }
56     }
57     return l;
58 }
59 int main()
60 {sws;
61
62     int n, m, a, b, k;
63     int v[MAX], aux[MAX];
64     int root[MAX];
65
66     cin >> n >> m;
67
68     for(int i=0;i<n;i++){
69         cin >> v[i]; aux[i]=v[i];
70     }
71
72     sort(v, v+n);
73
74     map<int, int> comp;
75     for(int i=0, j=0;i<n;i++)
76         if(i==0 or v[i]!=v[i-1])
77             comp[v[i]]=j++;
78
79     root[0]=build(0, n-1);
80
81     for(int i=1;i<=n;i++)
82         root[i] = update(0, n-1, root[i-1], comp[aux[
83             i-1]]);

```

```

84     for(int i=0;i<m;i++){
85         cin >> a >> b >> k;
86         cout << v[query(0, n-1, root[a-1], root[b], k
87     )] << endl;
88     }
89     return 0;
90 }

```

4.16 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  // 0(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         if(m > 1) {
13             res /= m;
14             res *= (m-1);
15         }
16         return res;
17     }
18
19     // modificacao do crivo, 0(n*log(log(n)))
20     vl phi_to_n(ll n){
21         vector<bool> isprime(n+1, true);
22         vl tot(n+1);
23         tot[0] = 0; tot[1] = 1;
24         for(ll i=1;i<=n; i++){
25             tot[i] = i;
26         }
27
28         for(ll p=2;p<=n;p++){
29             if(isprime[p]){
30                 tot[p] = p-1;
31                 for(ll i=p;i<=n;i+=p){
32                     isprime[i] = false;
33                     tot[i] = (tot[i]/p)*(p-1);
34                 }
35             }
36         }
37         return tot;
38     }

```

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 Factorization-sqrt

```

1 // Factorization of a number in sqrt(n)
2
3 vi fact(ll n){
4     vector<int> div;
5     for(ll i=2;i*i<=n;i++){
6         if(n%i==0){
7             vet.pb(i);
8             while(n%i==0)
9                 n/=i;
10        }
11        if(n!=1) vet.pb(n);
12        return div;
13 }

```

5.5 FFT-simple

```

1 struct num{
2     ld a {0.0}, b {0.0};
3     num(){ }
4     num(ld na) : a{na}{}
5     num(ld na, ld nb) : a{na}, b{nb} {}
6     const num operator+(const num &c) const{
7         return num(a + c.a, b + c.b);
8     }
9     const num operator-(const num &c) const{
10        return num(a - c.a, b - c.b);
11    }
12    const num operator*(const num &c) const{
13        return num(a*c.a - b*c.b, a*c.b + b*c.a);
14    }
15    const num operator/(const int &c) const{
16        return num(a/c, b/c);
17    }
18 };
19
20 void fft(vector<num> &a, bool invert){
21     int n = a.size();
22     for(int i=1,j=0;i<n;i++){
23         int bit = n>>1;
24         for(; j&bit; bit>>=1)
25             j^=bit;
26         j^=bit;
27         if(i<j)
28             swap(a[i], a[j]);
29     }
30     for(int len = 2; len <= n; len <= 1){
31         ld ang = 2 * PI / len * (invert ? -1 : 1);
32         num wlen(cos(ang), sin(ang));
33         for(int i=0;i<n;i+=len){
34             num w(1);
35             for (int j=0;j<len/2;j++){
36                 num u = a[i+j], v = a[i+j+len/2] * w;
37                 a[i+j] = u + v;
38                 a[i+j+len/2] = u - v;
39                 w = w * wlen;
40             }

```

```

41         }
42     }
43     if(invert)
44         for(num &x: a)
45             x = x/n;
46 }
47
48
49 vl multiply(vi const& a, vi const& b){
50     vector<num> fa(a.begin(), a.end());
51     vector<num> fb(b.begin(), b.end());
52     int n = 1;
53     while(n < int(a.size() + b.size()) )
54         n <= 1;
55     fa.resize(n);
56     fb.resize(n);
57     fft(fa, false);
58     fft(fb, false);
59     for(int i=0;i<n;i++)
60         fa[i] = fa[i]*fb[i];
61     fft(fa, true);
62     vl result(n);
63     for(int i=0;i<n;i++)
64         result[i] = round(fa[i].a);
65     while(result.back()==0) result.pop_back();
66     return result;
67 }

```

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 Lagrange-interpolation

```

1 // Lagrange's interpolation (n+1 points)
2 ld interpolate(vii d, ld x){
3     ld y = 0;
4     int n = d.size();
5     for(int i=0;i<n;i++){
6         ld yi = d[i].ss;
7         for(int j=0;j<n;j++){
8             if(j!=i)
9                 yi = yi*(x - d[j].ff)/(ld)(d[i].ff - d[j].ff);
10        }
11        y += yi;
12    }
13    return y;
14 }
15
16 ld inv_interpolate(vii d, ld y){
17     ld x = 0;
18     int n = d.size();
19     for(int i=0;i<n;i++){
20         ld xi = d[i].ff;
21         for(int j=0;j<n;j++){
22             if(j!=i)
23                 xi = xi*(y - d[j].ss)/(ld)(d[i].ss - d[j].ss);
24        }
25        x += xi;
26    }
27    return x;
28 }

```

5.8 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 }
```

5.9 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 }
```

5.10 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 return d;
54 }
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

```

5.11 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 }

```

5.12 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;

```

5.13 Bigmod

```

1 ll mod(string a, ll p) {
2     ll res = 0, b = 1;
3     reverse(all(a));
4
5     for(auto c : a) {
6         ll tmp = (((ll)c-'0')*b) % p;
7         res = (res + tmp) % p;
8
9         b = (b * 10) % p;
10    }
11
12    return res;
13 }

```

5.14 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    aprox. 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 }

```

5.15 FFT-tourist

```

1 struct num{
2     ld x, y;
3     num() { x = y = 0; }
4     num(ld x, ld 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 vi rev = {0, 1};

```

```

18         return;
19
20     rev.resize(1 << nbase);
21     for(int i = 0; i < (1 << nbase); i++)
22         rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (
nbase - 1));
23
24     roots.resize(1 << nbase);
25
26     while(base < nbase){
27         ld angle = 2*PI / (1 << (base + 1));
28         for(int i = 1 << (base - 1); i < (1 << base);
i++){
29             roots[i << 1] = roots[i];
30             ld angle_i = angle * (2 * i + 1 - (1 <<
base));
31             roots[(i << 1) + 1] = num(cos(angle_i),
sin(angle_i));
32         }
33         base++;
34     }
35 }
36
37 void fft(vector<num> &a, int n = -1){
38     if(n == -1)
39         n = a.size();
40
41     assert((n & (n-1)) == 0);
42     int zeros = __builtin_ctz(n);
43     ensure_base(zeros);
44     int shift = base - zeros;
45     for(int i = 0; i < n; i++){
46         if(i < (rev[i] >> shift))
47             swap(a[i], a[rev[i] >> shift]);
48
49         for(int k = 1; k < n; k <= 1)
50             for(int i = 0; i < n; i += 2 * k)
51                 for(int j = 0; j < k; j++){
52                     num z = a[i+j+k] * roots[j+k];
53                     a[i+j+k] = a[i+j] - z;
54                     a[i+j] = a[i+j] + z;
55                 }
56     }
57
58     vector<num> fa, fb;
59     vi multiply(vi &a, vi &b){
60         int need = a.size() + b.size() - 1;
61         int nbase = 0;
62         while((1 << nbase) < need) nbase++;
63         ensure_base(nbase);
64         int sz = 1 << nbase;
65         if(sz > (int) fa.size())
66             fa.resize(sz);
67
68         for(int i = 0; i < sz; i++){
69             int x = (i < (int) a.size() ? a[i] : 0);
70             int y = (i < (int) b.size() ? b[i] : 0);
71             fa[i] = num(x, y);
72         }
73         fft(fa, sz);
74         num r(0, -0.25 / sz);
75         for(int i = 0; i <= (sz >> 1); i++){
76             int j = (sz - i) & (sz - 1);
77             num z = (fa[j] * fa[j] - conj(fa[i] * fa[i]))
* r;
78             if(i != j) {
79                 fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[
j])) * r;
80             }
81             fa[i] = z;
82         }
83         fft(fa, sz);
84         vi res(need);
85         for(int i = 0; i < need; i++)
86             res[i] = fa[i].x + 0.5;
87
88         return res;
89     }
90
91     vi multiply_mod(vi &a, vi &b, int m, int eq = 0){
92         int need = a.size() + b.size() - 1;
93         int nbase = 0;
94         while((1 << nbase) < need) nbase++;
95         ensure_base(nbase);
96         int sz = 1 << nbase;
97         if(sz > (int) fa.size())
98             fa.resize(sz);
99
100         for(int i=0;i<(int)a.size();i++){
101             int x = (a[i] % m + m) % m;
102             fa[i] = num(x & ((1 << 15) - 1), x >> 15);
103         }
104         fill(fa.begin() + a.size(), fa.begin() + sz, num
{0, 0});
105         fft(fa, sz);
106         if(sz > (int) fb.size())
107             fb.resize(sz);
108         if(eq)
109             copy(fa.begin(), fa.begin() + sz, fb.begin())
;
110         else{
111             for(int i = 0; i < (int) b.size(); i++){
112                 int x = (b[i] % m + m) % m;
113                 fb[i] = num(x & ((1 << 15) - 1), x >> 15)
;
114             }
115             fill(fb.begin() + b.size(), fb.begin() + sz,
num {0, 0});
116             fft(fb, sz);
117
118             ld ratio = 0.25 / sz;
119             num r2(0, -1);
120             num r3(ratio, 0);
121             num r4(0, -ratio);
122             num r5(0, 1);
123             for(int i=0;i<=(sz >> 1);i++) {
124                 int j = (sz - i) & (sz - 1);
125                 num a1 = (fa[i] + conj(fa[j]));
126                 num a2 = (fa[i] - conj(fa[j])) * r2;
127                 num b1 = (fb[i] + conj(fb[j])) * r3;
128                 num b2 = (fb[i] - conj(fb[j])) * r4;
129                 if(i != j){
130                     num c1 = (fa[j] + conj(fa[i]));
131                     num c2 = (fa[j] - conj(fa[i])) * r2;
132                     num d1 = (fb[j] + conj(fb[i])) * r3;
133                     num d2 = (fb[j] - conj(fb[i])) * r4;
134                     fa[i] = c1 * d1 + c2 * d2 * r5;
135                     fb[i] = c1 * d2 + c2 * d1;
136                 }
137                 fa[j] = a1 * b1 + a2 * b2 * r5;
138                 fb[j] = a1 * b2 + a2 * b1;
139             }
140             fft(fa, sz);
141             fft(fb, sz);
142             vi res(need);
143             for(int i=0;i<need;i++){
144                 ll aa = fa[i].x + 0.5;
145                 ll bb = fb[i].x + 0.5;
146                 ll cc = fa[i].y + 0.5;
147                 res[i] = (aa + ((bb % m) << 15) + ((cc % m)
<< 30)) % m;
148             }
149             return res;
150         }
151     }
152 }

```

```

153
154
155
156 int main()
157 {sws;
158
159 //FFT
160 vi fx{1, 2, 3}; // 1+2x+3x^2
161 vi gx{4, 5}; // 4+5x
162 vi res;
163
164 res = multiply(fx,gx); //4 + 13x + 22x^2 + 15x^3
165
166 return 0;
167
168 }

```

5.16 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));

```

5.17 Fast-Exponentiation

```

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

```

5.18 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 }

```

5.19 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 }

```

```

27
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 --- 0(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 }

```

5.20 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    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)

```

```

11
12 // Exchange two int variables
13
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
71 -1)))}; }
72
73 // Turn off the first bit 1
74     m = m & (m-1);
75
76 // Built-in functions
77
78     // Number of bits 1
79     __builtin_popcount()
80     __builtin_popcountll()
81
82     // Number of leading zeros
83     __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
94         int flog2ll(ll x){ return 64-1-
95         __builtin_clzll(x); }

```

6.4 All-Subsets

```

1 vi a, subset;
2 vector<vi> subsets;
3
4 // Iterative
5 void search(int k){
6     if(k==(int)a.size())
7         subsets.pb(subset);
8     else{
9         search(k+1);
10        subset.pb(a[k]);
11        search(k+1);
12        subset.pop_back();
13    }
14 }
15 search(0);
16
17 // Binary
18 for(int b=0;b<(1<<n);b++){
19     vi subset;
20     for(int i=0;i<n;i++){
21         if(b&(1<<i)) subset.pb(a[i]);
22     }
23     subsets.pb(subset);
24 }

```

6.5 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 vl vector<ll>
13 #define vii vector<pii>
14 #define sws ios_base::sync_with_stdio(false);cin.tie(
15     NULL)
16 #define endl '\n'
17 #define teto(a, b) (a+b-1)/(b)
18
19 const int MAX = 300010;
20 const int MOD = 1e9;
21 const int INF = 0x3f3f3f3f;
22 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
23 const ld EPS = 1e-8;
24 const ld PI = acosl(-1.0);
25
26 using namespace std;

```

6.6 Rand

```

1 mt19937 rng(chrono::steady_clock::now().
  time_since_epoch().count());
2 uniform_int_distribution<int> distribution(1,n);
3
4 num = distribution(rng); // num no range [1, n]
5 shuffle(vec.begin(), vec.end(), rng); // shuffle

```

6.7 Meet-in-the-middle

```

1 // Subsequence with the biggest sum%m value 0(2^(n/2)
  *n)
2
3 int n, m, a[40];
4
5 void comb(int l, int r, vi &v){
6     int sz = r-l+1;
7     for(int i=0;i<(1<<sz);i++){
8         int sum = 0;
9         for(int j=0;j<sz;j++){
10             if(i & (1<<j))
11                 sum = (sum + a[l+j])%m;
12             v.pb(sum);
13         }
14
15         sort(v.begin(), v.end());
16     }
17
18     int merge(vi &x, vi &y){
19         int k=y.size()-1, ans=0;
20         for(auto v: x){
21             while(k>0 and v+y[k]>=m)
22                 k--;
23             ans = max(ans, v+y[k]);
24         }
25         return ans;
26     }
27
28     int main()
29     {sws;
30
31         vi x, y;
32         cin >> n >> m;
33
34         for(int i=0;i<n;i++){
35             cin >> a[i];
36
37             comb(0, n/2, x);
38             comb(n/2 + 1, n-1, y);
39             cout << merge(x, y) << endl;
40
41             return 0;
42         }
43     }

```

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 vi pi(const string &s){
2     int n=s.size();
3     vi p(n);
4     for(int i=1, j=0; i<n; i++){
5         while(j>0 and s[i]!=s[j]) j=p[j-1];
6         if(s[j]==s[i]) j++;
7         p[i]=j;
8     }
9     return p;
10 }
11
12 vi kmp(const string &t, const string &s){
13     vi p = pi(s+'$'), match;
14     int n=t.size(), m=s.size();
15     for(int i=0, j=0; i<n; i++){
16         while(j>0 and t[i]!=s[j]) j=p[j-1];
17         if(t[i]==s[j]) j++;
18         if(j==m) match.pb(i-j+1);
19     }
20     return match;
21 }

```

7.3 Suffix-array

```

1 vi suffix_array(string s){
2     s.pb('$');
3     int n = s.size();
4
5     vi p(n), c(n);
6     vector< pair<char, int> > a(n);
7     for(int i=0;i<n;i++) a[i] = {s[i], i};
8     sort(a.begin(), a.end());
9
10     for(int i=0;i<n;i++) p[i] = a[i].ss;
11     c[p[0]]=0;
12     for(int i=1;i<n;i++)
13         c[p[i]] = c[p[i-1]] + (a[i].ff!=a[i-1].ff);
14
15     int k=0;
16     while((1<<k) < n){
17         vector< pair<pii, int> > a(n);
18         for(int i=0;i<n;i++)
19             a[i] = {{c[i], c[(i+(1<<k))%n]}, i};
20         sort(a.begin(), a.end());
21
22         for(int i=0;i<n;i++) p[i] = a[i].ss;
23         c[p[0]]=0;
24         for(int i=1;i<n;i++)
25             c[p[i]] = c[p[i-1]] + (a[i].ff!=a[i-1].ff);
26     };
27     k++;
28     return p;
29 }

```

7.4 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            } else
22                len[currRow][j] = 0;
23        }
24        currRow = 1 - currRow;
25    }
26
27    if(result==0)
28        return string();
29
30    return X.substr(end - result + 1, result);
31 }
32 }

```

7.5 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        divisor = divisor/100;
18    }
19
20    return true;
21 }
22 }

```

7.6 Z-Func

```

1 vi z_algo(const string &s)
2 { // returns vector for each idx where a prefix of
3   size i starts.
4   int n = s.size();
5   int L = 0, R = 0;
6   vi z(n, 0);

```

```

6   for(int i = 1; i < n; i++){
7       if(i <= R)
8           z[i] = min(z[i-L], R - i + 1);
9       while(z[i]+i < n and s[z[i]+i] == s[z[i]
10          ])
11           z[i]++;
12       if(i+z[i]-1 > R){
13           L = i;
14           R = i + z[i] - 1;
15       }
16   }
17   return z;
18 }

```

7.7 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 }

```

7.8 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);
22        while(0 <= i - k - 1 && i + k < n && s[i - k
23            - 1] == s[i + k]) {
24            k++;
25        }
26        d2[i] = k--;
27        if(i + k > r) {
28            l = i - k - 1;
29            r = i + k;
30        }
31    }
32 }

```