



Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos	3	3.12 Heron	12
1.1 Meet-in-the-middle	3	3.13 Uniao-segmentos	12
1.2 Iterative-BS	3	3.14 Minkowski-Sum	12
2 Grafos	3	3.15 Simetria-central	12
2.1 BFS-01	3	3.16 Rotating-Callipers	12
2.2 BFS	3	3.17 2D	13
2.3 2SAT	3	4 ED	15
2.4 Hungarian	4	4.1 Trie	15
2.5 Find-bridges	4	4.2 Prefixsum2D	15
2.6 HLD-Vertice	4	4.3 Delta-Encoding	15
2.7 Kahn	5	4.4 BIT-2D	15
2.8 Dijkstra	5	4.5 BIT-bigger-k	16
2.9 LCA	5	4.6 BIT	16
2.10 HLD-Aresta	6	4.7 Minqueue	16
2.11 Floyd-Warshall	6	4.8 BIT-kth	17
2.12 Kruskal	6	4.9 Mergesorttree	17
2.13 DFS	6	4.10 Sparse-Table	17
2.14 Topological-sort	6	4.11 Union-Find	17
2.15 Kosaraju	7	4.12 CHT	18
2.16 Centroid	7	4.13 Mo	18
2.17 Ford	7	5 DP	18
2.18 Dinic	7	5.1 Mochila	18
2.19 Prim	8	5.2 Dp-digitos	19
3 Geometria	9	5.3 Unbounded-Knapsack	19
3.1 NumIntersectionLine	9	5.4 Largest-KSubmatrix	19
3.2 Inside-polygon	9	5.5 LIS	19
3.3 Polygon-Diameter	9	5.6 Partition-Problem	19
3.4 MinDistPair	9	6 Math	20
3.5 Intersect-polygon	10	6.1 Totient	20
3.6 Sort-by-Angle	10	6.2 Double-gcd	20
3.7 Convex-Hull	10	6.3 Linear-Diophantine-Equation	20
3.8 Inter-Retangulos	10	6.4 Factorization-sqrt	20
3.9 Half-Plane-Intersect	10	6.5 FFT-simple	20
3.10 Tetrahedron-Distance3D	10	6.6 Mulmod	21
3.11 3D	11	6.7 Lagrange-interpolation	21
		6.8 Crt	21

6.9	Miller-Habin	21
6.10	Inverso-Mult	22
6.11	Pollard-Rho	22
6.12	Verif-primo	22
6.13	Mobius	22
6.14	Exponenciacao-matriz	23
6.15	Crivo	23
6.16	Bigmod	23
6.17	Simpson's-formula	23
6.18	FFT-tourist	23
6.19	Modular-Exponentiaion	25
6.20	Next-Permutation	25
6.21	Raiz-primitiva	25
6.22	Kamenetsky	25
7	Misc	25
7.1	Ordered-Set	25
7.2	Bitwise	25
7.3	All-Subsets	26
7.4	Trie-bits	26
7.5	Template	27
7.6	Rand	27
7.7	Safe-Map	27
8	Strings	27
8.1	Trie	27
8.2	KMP	28
8.3	Suffix-array	28
8.4	LCS	28
8.5	LCSubseq	28
8.6	Edit-distance	29
8.7	Pal-int	29
8.8	Z-Func	29
8.9	Hash	29
8.10	Manacher	29
8.11	Suffix-array-radix	30

1 Algoritmos

1.1 Meet-in-the-middle

```
1 // Subsequence with the biggest sum%m value  $O(2^{(n/2)} * n)$ 
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         sort(v.begin(), v.end());
15     }
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
29 int main()
30 {sws;
31
32     vi x, y;
33     cin >> n >> m;
34
35     for(int i=0; i<n; i++)
36         cin >> a[i];
37
38     comb(0, n/2, x);
39     comb(n/2 + 1, n-1, y);
40     cout << merge(x, y) << endl;
41
42     return 0;
43 }
```

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

2 Grafos

2.1 BFS-01

```
1 vector<int> d(n, INF);
2 deque<int> q;
3
4 void bfs(int x){
5     d[x] = 0;
6     q.push_front(x);
7     while(!q.empty()){
8         int u = q.front();
9         q.pop_front();
10        for(auto e: grafo[u]){
11            int v = edge.ff;
12            int w = edge.ss;
13            if(d[v] > d[u] + w){
14                d[v] = d[u] + w;
15                if(w == 1)
16                    q.push_back(v);
17                else
18                    q.push_front(v);
19            }
20        }
21    }
22 }
```

2.2 BFS

```
1 queue<int> q;
2 vector<bool> used(n);
3 vi d(n), p(n);
4
5 void bfs(int x){
6     q.push(x);
7     used[x] = true;
8     p[x] = -1;
9     while(!q.empty()){
10        int u = q.front();
11        q.pop();
12        for(int v: adj[u]) {
13            if(!used[v]){
14                used[v] = true;
15                q.push(v);
16                d[v] = d[u] + 1;
17                p[v] = u;
18            }
19        }
20    }
21 }
22
23 // Restore
24 if(!used[u])
25     cout << "No path!";
26 else{
27     vi path;
28     for(int v = u; v != -1; v = p[v])
29         path.push_back(v);
30     reverse(path.begin(), path.end());
31     cout << "Path: ";
32     for (int v : path)
33         cout << v << " ";
34 }
```

2.3 2SAT

```
1 vector<int> g[MAX], gt[MAX], S; int vis[MAX], cor[MAX]
2 ];
3
4 int val(int n, bool tvalue) {
5     if(tvalue) return 2*n;
6     return 2*n + 1;
7 }
8
9 void dfs(int u) {
10     vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
11 }
```

```

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 }

```

2.4 Hungarian

```

1 template<typename T> struct hungarian {
2     int n, m;
3     vector<vector<T>> a;
4     vector<T> u, v;
5     vector<int> p, way;
6     T inf;
7
8     hungarian(int n_, int m_) : n(n_), m(m_), u(m+1),
9         v(m+1), p(m+1), way(m+1) {
10         a = vector<vector<T>>(n, vector<T>(m));
11         inf = numeric_limits<T>::max();
12     }
13
14     pair<T, vector<int>> assignment() {
15         for (int i = 1; i <= n; i++) {
16             p[0] = i;
17             int j0 = 0;
18             vector<T> minv(m+1, inf);
19             vector<int> used(m+1, 0);
20             do {
21                 used[j0] = true;
22                 int i0 = p[j0], j1 = -1;
23                 T delta = inf;
24                 for (int j = 1; j <= m; j++) if (!
25                     used[j]) {
26                         T cur = a[i0-1][j-1] - u[i0] - v[
27                             j];
28                         if (cur < minv[j]) minv[j] = cur,
29                             way[j] = j0;
30                         if (minv[j] < delta) delta = minv[
31                             j], j1 = j;
32                     }
33                 for (int j = 0; j <= m; j++)

```

```

28                 if (used[j]) u[p[j]] += delta, v[
29                     j] -= delta;
30                 else minv[j] -= delta;
31                 j0 = j1;
32             } while (p[j0] != 0);
33             do {
34                 int j1 = way[j0];
35                 p[j0] = p[j1];
36                 j0 = j1;
37             } while (j0);
38             vector<int> ans(m);
39             for (int j = 1; j <= n; j++) ans[p[j]-1] = j
40                 -1;
41             return make_pair(-v[0], ans);
42         }
43     }
44 }
45
46 };

```

2.5 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 }

```

2.6 HLD-Vertice

```

1 struct Hld {
2     Segtree st;
3     int n;
4     vector<vi> g;
5     vi pos, sz, peso, pai, h, v;
6     int t;
7
8     Hld(int n){
9         this->n=n;
10        st = Segtree(n);
11        g.assign(n, vi());
12        pos.assign(n, 0);sz.assign(n, 0);
13        peso.assign(n, 0);pai.assign(n, 0);
14        h.assign(n, 0);v.assign(n, 0);
15    }
16
17     void build_hld(int k, int p = -1, int f = 1){

```

```

18     v[pos[k] = t++] = peso[k]; sz[k] = 1;
19     for(auto &i: g[k]) if(i!=p){
20         pai[i] = k;
21         h[i] = (i==g[k][0] ? h[k]:i);
22         build_hld(i, k, f); sz[k]+=sz[i];
23
24         if(sz[i]>sz[g[k][0]] or g[k][0]==p) swap(
i, g[k][0]);
25     }
26     if(p*f == -1) build_hld(h[k] = k, -1, t = 0);
27 }
28 void build(int root = 0){
29     t = 0;
30     build_hld(root);
31     for(int i=0;i<n;i++) st.seg[i+n]=v[i];
32     st.build();
33 }
34 ll query_path(int a, int b){
35     if(pos[a]<pos[b]) swap(a, b);
36
37     if(h[a]==h[b]) return st.query(pos[b], pos[a
]);
38     return st.query(pos[h[a]], pos[a]) +
query_path(pai[h[a]], b);
39 }
40 void update_path(int a, int b, int x){
41     if(pos[a]<pos[b]) swap(a, b);
42
43     if(h[a]==h[b]) return (void)st.update(pos[b],
pos[a], x);
44     st.update(pos[h[a]], pos[a], x); update_path(
pai[h[a]], b, x);
45 }
46 ll query_subtree(int a){
47     return st.query(pos[a], pos[a]+sz[a]-1);
48 }
49 void update_subtree(int a, int x){
50     st.update(pos[a], pos[a]+sz[a]-1, x);
51 }
52 int lca(int a, int b){
53     if(pos[a]<pos[b]) swap(a, b);
54     return (h[a]==h[b] ? b:lca(pai[h[a]], b));
55 }
56 };

```

2.7 Kahn

```

1 vi g[MAX];
2 int in[MAX], cor[MAX];
3 void kahn(int n) {
4     int label = 1;
5     priority_queue<int, vector<int>, greater<int>> pq
; // trocar por queue para O(n)
6     for(int i = 1; i <= n; i++) {
7         if(in[i] == 0) {
8             pq.push(i);
9         }
10    }
11
12    while(pq.size()) {
13        int u = pq.top(); pq.pop();
14        cor[u] = label++;
15        for(auto prox : g[u]) {
16            in[prox]--;
17            if(in[prox] == 0) {
18                pq.push(prox);
19            }
20        }
21    }
22 }

```

2.8 Dijkstra

```

1 // Dijkstra - Shortest Path
2
3 vector<vii> g(MAX+1, vii());
4 vi d(MAX+1, INF);
5 priority_queue<pii, vii, greater<pii> > fila;
6
7 void dijkstra(int k){
8     d[k]=0;
9     fila.push({0, k});
10
11     while(!fila.empty()){
12         int w=fila.top().ff, u=fila.top().ss;
13         fila.pop();
14         if(w>d[u]) continue;
15
16         for(auto [v, w]: g[u]){
17             if(d[v]>d[u]+w){
18                 d[v]=d[u]+w;
19                 fila.push({d[v], v});
20             }
21         }
22     }
23 }

```

2.9 LCA

```

1 template<typename T> struct rmq {
2     vector<T> v;
3     int n; static const int b = 30;
4     vector<int> mask, t;
5
6     int op(int x, int y) { return v[x] < v[y] ? x : y
; }
7     int msb(int x) { return __builtin_clz(1)-
__builtin_clz(x); }
8     rmq() {}
9     rmq(const vector<T>& v_) : v(v_), n(v.size()),
mask(n), t(n) {
10         for (int i = 0, at = 0; i < n; mask[i++] = at
|= 1) {
11             at = (at<<1)&((1<<b)-1);
12             while (at and op(i, i-msb(at&-at)) == i)
at ^= at&-at;
13         }
14         for (int i = 0; i < n/b; i++) t[i] = b*i+b-1-
msb(mask[b*i+b-1]);
15         for (int j = 1; (1<<j) <= n/b; j++) for (int
i = 0; i+(1<<j) <= n/b; i++)
16             t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j
-1)+i+(1<<(j-1))]);
17     }
18     int small(int r, int sz = b) { return r-msb(mask[
r]&((1<<sz)-1)); }
19     T query(int l, int r) {
20         if (r-l+1 <= b) return small(r, r-l+1);
21         int ans = op(small(l+b-1), small(r));
22         int x = l/b+1, y = r/b-1;
23         if (x <= y) {
24             int j = msb(y-x+1);
25             ans = op(ans, op(t[n/b*j+x], t[n/b*j+y
-(1<<j)+1]));
26         }
27         return ans;
28     }
29 };
30
31 namespace lca {
32     vector<int> g[MAX];
33     int v[2*MAX], pos[MAX], dep[2*MAX];
34     int t;
35     rmq<int> RMQ;
36
37     void dfs(int i, int d = 0, int p = -1) {

```

```

38     v[t] = i, pos[i] = t, dep[t++] = d;
39     for (int j : g[i]) if (j != p) {
40         dfs(j, d+1, i);
41         v[t] = i, dep[t++] = d;
42     }
43 }
44 void build(int n, int root) {
45     t = 0;
46     dfs(root);
47     RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
48 }
49 int lca(int a, int b) {
50     a = pos[a], b = pos[b];
51     return v[RMQ.query(min(a, b), max(a, b))];
52 }
53 int dist(int a, int b) {
54     return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[
55 lca(a, b)]];
56 }
57 }

```

2.10 HLD-Aresta

```

1 struct Hld {
2     Segtree st;
3     int n;
4     vector<vii> g;
5     vi pos, sz, sobe, pai, h, v;
6     int t;
7
8     Hld(int n){
9         this->n=n;
10        st = Segtree(n);
11        g.assign(n, vii());
12        pos.assign(n, 0);sz.assign(n, 0);
13        sobe.assign(n, 0);pai.assign(n, 0);
14        h.assign(n, 0);v.assign(n, 0);
15    }
16
17    void build_hld(int k, int p = -1, int f = 1){
18        v[pos[k] = t++] = sobe[k]; sz[k] = 1;
19        for(auto &i: g[k]) if(i.ff != p){
20            sobe[i.ff] = i.ss; pai[i.ff] = k;
21            h[i.ff] = (i==g[k][0] ? h[k]:i.ff);
22            build_hld(i.ff, k, f); sz[k]+=sz[i.ff];
23
24            if(sz[i.ff]>sz[g[k][0].ff] or g[k][0].ff
25 ==p) swap(i, g[k][0]);
26        }
27        if(p*f == -1) build_hld(h[k] = k, -1, t = 0);
28    }
29    void build(int root = 0){
30        t = 0;
31        build_hld(root);
32        for(int i=0;i<n;i++) st.seg[i+n]=v[i];
33        st.build();
34    }
35    ll query_path(int a, int b){
36        if(a==b) return 0;
37        if(pos[a]<pos[b]) swap(a, b);
38
39        if(h[a]==h[b]) return st.query(pos[b]+1, pos[
40 a]);
41        return st.query(pos[h[a]], pos[a]) +
42 query_path(pai[h[a]], b);
43    }
44    void update_path(int a, int b, int x){
45        if(a==b) return;
46        if(pos[a]<pos[b]) swap(a, b);
47
48        if(h[a]==h[b]) return (void)st.update(pos[b
49 ]+1, pos[a], x);
50    }
51 }

```

```

46        st.update(pos[h[a]], pos[a], x); update_path(
47 pai[h[a]], b, x);
48    }
49    ll query_subtree(int a){
50        if(sz[a]==1) return 0;
51        return st.query(pos[a]+1, pos[a]+sz[a]-1);
52    }
53    void update_subtree(int a, int x){
54        if(sz[a]==1) return;
55        st.update(pos[a]+1, pos[a]+sz[a]-1, x);
56    }
57    int lca(int a, int b){
58        if(pos[a] < pos[b]) swap(a, b);
59        return (h[a]==h[b] ? b:lca(pai[h[a]], b));
60    }
61 }

```

2.11 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 }

```

2.12 Kruskal

```

1 // Uses DSU .join() and .find()
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 DSU dsu(n);
11 vector<Edge> edges, result;
12 int cost = 0;
13
14 sort(edges.begin(), edges.end());
15
16 for(auto e : edges) {
17     if (dsu.find(e.u) != dsu.find(e.v)) {
18         cost += e.weight;
19         result.push_back(e); // vector com as arestas
20         dsu.join(e.u, e.v);
21     }
22 }

```

2.13 DFS

```

1 void DFS(int u, int pai){
2     for(auto v: grafo[u]) if(v!=pai){
3         DFS(v, u);
4     }
5 }

```

2.14 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     vi resp;
6     for(int i=1;i<=n;i++)
7         if(!grau[i])
8             resp.push_back(i);
9
10    int k=0;
11    while(k < (int)resp.size()){
12        int u = resp[k];
13        k++;
14        for(auto v: grafo[u]){
15            grau[v]--;
16            if(!grau[v])
17                resp.pb(v);
18        }
19    }
20
21    if((int)resp.size() < n)
22        cout << "impossivel\n";
23
24    return resp;
25 }

```

2.15 Kosaraju

```

1 int n;
2 vi g[MAX], gi[MAX]; // grafo invertido
3 int vis[MAX], comp[MAX]; // componente conexo de cada
4   vertice
5 stack<int> S;
6 void dfs(int u){
7     vis[u] = 1;
8     for(auto v: g[u]) if(!vis[v]) dfs(v);
9     S.push(u);
10 }
11
12 void scc(int u, int c){
13     vis[u] = 1; comp[u] = c;
14     for(auto v: gi[u]) if(!vis[v]) scc(v, c);
15 }
16
17 void kosaraju(){
18     for(int i=0;i<n;i++) vis[i] = 0;
19     for(int i=0;i<n;i++) if(!vis[i]) dfs(i);
20     for(int i=0;i<n;i++) vis[i] = 0;
21     while(S.size()){
22         int u = S.top();
23         S.pop();
24         if(!vis[u]) scc(u, u);
25     }
26 }

```

2.16 Centroid

```

1 int sz[MAX];
2 bool erased[MAX];
3 vi grafo[MAX];
4
5 void dfs(int u, int p=-1){
6     sz[u] = 1;
7     for(int v: grafo[u]) if(v!=p and !erased[v]){
8         dfs(v, u);
9         sz[u] += sz[v];
10    }
11 }
12
13 int centroid(int u, int p=-1, int size=-1){
14     if(size==-1) size = sz[u];
15     for(int v: grafo[u])
16         if(v!=p and !erased[v] and sz[v]>size/2)
17             return centroid(v, u, size);

```

```

18     return u;
19 }
20
21 pii centroids(int u=1){ // idx 1
22     dfs(u);
23     int c1=centroid(u), c2=c1;
24     for(int v: grafo[c1]) if(2*sz[v]==sz[u]) c2=v;
25     return {c1, c2};
26 }

```

2.17 Ford

```

1 const int N = 2000010;
2
3 struct Ford {
4     struct Edge {
5         int to, f, c;
6     };
7
8     int vis[N];
9     vector<int> adj[N];
10    vector<Edge> edges;
11    int cur = 0;
12
13    void addEdge(int a, int b, int cap, int rcap) {
14        Edge e;
15        e.to = b; e.c = cap; e.f = 0;
16        edges.pb(e);
17        adj[a].pb(cur++);
18
19        e = Edge();
20        e.to = a; e.c = rcap; e.f = 0;
21        edges.pb(e);
22        adj[b].pb(cur++);
23    }
24
25    int dfs(int s, int t, int f, int tempo) {
26        if(s == t)
27            return f;
28        vis[s] = tempo;
29
30        for(int e : adj[s]) {
31            if(vis[edges[e].to] < tempo and (edges[e]
32            ].c - edges[e].f) > 0) {
33                if(int a = dfs(edges[e].to, t, min(f,
34                edges[e].c-edges[e].f), tempo)) {
35                    edges[e].f += a;
36                    edges[e^1].f -= a;
37                    return a;
38                }
39            }
40        }
41        return 0;
42    }
43
44    int flow(int s, int t) {
45        int mflow = 0, tempo = 1;
46        while(int a = dfs(s, t, INF, tempo)) {
47            mflow += a;
48            tempo++;
49        }
50        return mflow;
51    }
52 };

```

2.18 Dinic

```

1 const int N = 300;
2
3 struct Dinic {
4     struct Edge{
5         int from, to; ll flow, cap;

```

```

6   };
7   vector<Edge> edge;
8
9   vector<int> g[N];
10  int ne = 0;
11  int lvl[N], vis[N], pass;
12  int qu[N], px[N], qt;
13
14  ll run(int s, int sink, ll minE) {
15      if(s == sink) return minE;
16
17      ll ans = 0;
18
19      for(; px[s] < (int)g[s].size(); px[s]++) {
20          int e = g[s][px[s]];
21          auto &v = edge[e], &rev = edge[e^1];
22          if(lvl[v.to] != lvl[s]+1 || v.flow >= v.
23              continue; // v.cap - v.flow
24          < lim
25          ll tmp = run(v.to, sink, min(minE, v.cap-v
26          .flow));
27          v.flow += tmp, rev.flow -= tmp;
28          ans += tmp, minE -= tmp;
29          if(minE == 0) break;
30      }
31      return ans;
32  }
33  bool bfs(int source, int sink) {
34      qt = 0;
35      qu[qt++] = source;
36      lvl[source] = 1;
37      vis[source] = ++pass;
38      for(int i = 0; i < qt; i++) {
39          int u = qu[i];
40          px[u] = 0;
41          if(u == sink) return true;
42          for(auto& ed : g[u]) {
43              auto v = edge[ed];
44              if(v.flow >= v.cap || vis[v.to] ==
45                  continue; // v.cap - v.flow < lim
46              vis[v.to] = pass;
47              lvl[v.to] = lvl[u]+1;
48              qu[qt++] = v.to;
49          }
50      }
51      return false;
52  }
53  ll flow(int source, int sink) {
54      reset_flow();
55      ll ans = 0;
56      //for(lim = (1LL << 62); lim >= 1; lim /= 2)
57      while(bfs(source, sink))
58          ans += run(source, sink, LLINF);
59      return ans;
60  }
61  void addEdge(int u, int v, ll c, ll rc) {
62      Edge e = {u, v, 0, c};
63      edge.pb(e);
64      g[u].push_back(ne++);
65
66      e = {v, u, 0, rc};
67      edge.pb(e);
68      g[v].push_back(ne++);
69  }
70  void reset_flow() {
71      for(int i = 0; i < ne; i++)
72          edge[i].flow = 0;
73      memset(lvl, 0, sizeof(lvl));
74      memset(vis, 0, sizeof(vis));
75      memset(qu, 0, sizeof(qu));
76      memset(px, 0, sizeof(px));

```

```

75      qt = 0; pass = 0;
76  }
77  };

```

2.19 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     ;
14     distancia[1] = 0;
15
16     priority_queue< pii, vector<pii>, greater<pii> >
17     fila;
18     fila.push( pii(distancia[1], 1) );
19
20     while(1){
21         int davez = -1;
22
23         while(!fila.empty()){
24             int atual = fila.top().second;
25             fila.pop();
26
27             if(!processado[atual]){
28                 davez = atual;
29                 break;
30             }
31         }
32
33         if(davez == -1)
34             break;
35
36         processado[davez] = true;
37
38         for(int i = 0; i < (int)vizinhos[davez].size()
39             ; i++){
40             int dist = vizinhos[davez][i].first;
41             int atual = vizinhos[davez][i].second;
42
43             if( distancia[atual] > dist && !
44                 processado[atual]){
45                 {
46                     distancia[atual] = dist;
47                     fila.push( pii(distancia[atual],
48                         atual) );
49                 }
50             }
51         }
52     }
53
54     int custo_arvore = 0;
55     for(int i = 1; i <= n; i++){
56         custo_arvore += distancia[i];
57     }
58
59     return custo_arvore;
60 }
61
62 int main(){
63     cin >> n >> m;
64
65     for(int i = 1; i <= m; i++){
66         int x, y, tempo;
67         cin >> x >> y >> tempo;

```



```

63         vizinhos[x].pb( pii(tempo, y) );
64         vizinhos[y].pb( pii(tempo, x) );
65     }
66 }
67
68 cout << Prim() << endl;
69
70 return 0;
71 }

```

3 Geometria

3.1 NumIntersectionLine

```

1 int main()
2 {
3     int lim = 1e6;
4     Segtree st(lim+100);
5     int n, m, y, x, l, r;
6     cin >> n >> m;
7
8     int open=-1, close=INF; // open -> check -> close
9     vector< pair<int, pii> > sweep;
10
11     ll ans = 0;
12     for(int i=0;i<n;i++){ // horizontal
13         cin >> y >> l >> r;
14         sweep.pb({l, {open, y}});
15         sweep.pb({r, {close, y}});
16     }
17     for(int i=0;i<m;i++){ // vertical
18         cin >> x >> l >> r;
19         sweep.pb({x, {l, r}});
20     }
21     sort(sweep.begin(), sweep.end());
22
23     // set<int> on;
24     for(auto s: sweep){
25         if(s.ss.ff==open){
26             st.update(s.ss.ss, 1);
27             // on.insert(s.ss.ss);
28         }
29         else if(s.ss.ff==close){
30             st.update(s.ss.ss, -1);
31             // on.erase(s.ss.ss);
32         }
33         else{
34             ans += st.query(s.ss.ff, s.ss.ss);
35             // auto it1 = on.lower_bound(s.ss.ff);
36             // auto it2 = on.upper_bound(s.ss.ss);
37             // for(auto it = it1; it!=it2; it++){
38             //     intersection -> (s.ff, it);
39             // }
40         }
41     }
42
43     cout << ans << endl;
44
45     return 0;
46 }

```

3.2 Inside-polygon

```

1 bool insideT(point a, point b, point c, point e){
2     int x = ccw(a, b, e);
3     int y = ccw(b, c, e);
4     int z = ccw(c, a, e);
5     // if(!x or !y or !z) return false; // bordo
6     return !((x==1 or y==1 or z==1) and (x==-1 or y
==-1 or z==-1));

```

```

7 }
8
9 bool inside(vp &vet, point e){ // ccw
10     int l=2, r=(int)vet.size()-1;
11     int res=r;
12     while(l<r){
13         int mid = (l+r)/2;
14         if(ccw(vet[0], vet[mid], e)==1)
15             l=mid+1;
16         else{
17             r=mid;
18             res=mid;
19         }
20     }
21
22     return insideT(vet[0], vet[res-1], vet[res], e);
23 }

```

3.3 Polygon-Diameter

```

1 double diameter(const vector<point> &p) {
2     vector<point> h = convexHull(p);
3     int m = h.size();
4     if (m == 1)
5         return 0;
6     if (m == 2)
7         return dist(h[0], h[1]);
8     int k = 1;
9     while (area(h[m - 1], h[0], h[(k + 1) % m]) >
10         area(h[m - 1], h[0], h[k]))
11         ++k;
12     double res = 0;
13     for (int i = 0, j = k; i <= k && j < m; i++) {
14         res = max(res, dist(h[i], h[j]));
15         while (j < m && area(h[i], h[(i + 1) % m], h
16             [(j + 1) % m]) > area(h[i], h[(i + 1) % m], h[j]))
17             ++j;
18         res = max(res, dist(h[i], h[(j + 1) % m]));
19         ++j;
20     }
21     return res;
22 }

```

3.4 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
16            ].x});
17        auto it2 = s.upper_bound({vet[i].y + d, vet[i
18            ].x});
19
20        for(auto it=it1; it!=it2; it++){
21            ll dx = vet[i].x - it->x;
22            ll dy = vet[i].y - it->y;
23            if(best_dist > dx*dx + dy*dy){
24                best_dist = dx*dx + dy*dy;
25                // vet[i] e inv(it)

```

```

24         }
25     }
26
27     s.insert(point(vet[i].y, vet[i].x));
28 }
29 return best_dist;
30 }

```

3.5 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 }

```

3.6 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 (a^b)>0;
17     else
18         return qa<qb;
19 }
20
21 c = center(A);
22 sort(A.begin(), A.end(), comp);

```

3.7 Convex-Hull

```

1 vp convex_hull(const vp &points)
2 {
3     vp P(points);
4     sort(P.begin(), P.end());
5     vp L, U;
6     for(auto p: P){
7         while(L.size()>=2 and ccw(L[L.size()-2], L.
8             back(), p)!=1)
9             L.pop_back();
10        L.push_back(p);
11    }
12    reverse(P.begin(), P.end());
13    for(auto p: P){
14        while(U.size()>=2 and ccw(U[U.size()-2], U.
15            back(), p)!=1)
16            U.pop_back();
17        U.push_back(p);
18    }
19    L.pop_back();

```

```

18     L.insert(L.end(), U.begin(), U.end()-1);
19     return L;
20 }

```

3.8 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 }

```

3.9 Half-Plane-Intersect

```

1 // Half plane intersect O(n3)
2 vp half_plane_intersect(vector<line> &v){
3     vp ret;
4     int n = v.size();
5     for(int i=0; i<n; i++){
6         for(int j=i+1; j<n; j++){
7             point crs = inter(v[i], v[j]);
8             if(crs.x == INF) continue;
9             bool bad = 0;
10            for(int k=0; k<n; k++){
11                if(v[k].eval(crs) < -EPS){
12                    bad = 1;
13                    break;
14                }
15            }
16            if(!bad) ret.push_back(crs);
17        }
18    }
19    return ret;
20 }

```

3.10 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 }

```

```

27 //se ponto for coplanar ao triangulo (e dentro do
28 triangulo)
29 //vai retornar zero corretamente
30 return fabs(misto(p-v[0],v1,v2)/norm(n));
31 }
32 ld dist_pt_seg(point p, vp li){
33 return norm((li[i]-li[0])^(p-li[0]))/norm(li[i]-
34 li[0]);
35 }
36 ld dist_line(vp l1, vp l2){
37 point n = (l1[1]-l1[0])^(l2[1]-l2[0]);
38 if(nulo(n)) //retas paralelas - dist ponto a reta
39 return dist_pt_seg(l2[0],l1);
40
41 point o1o2 = l2[0]-l1[0];
42 return fabs((o1o2*n)/norm(n));
43 }
44 // retas paralelas e intersecao nao nula
45 ld dist_seg(vp l1, vp l2){
46
47 assert(l2.size()==2);
48 assert(l1.size()==2);
49
50 //pontos extremos do segmento
51 ld ans = LLINF;
52 for(int i=0;i<2;i++){
53 for(int j=0;j<2;j++){
54 ans = min(ans, norm(l1[i]-l2[j]));
55
56 //verificando distancia de ponto extremo com
57 ponto interno dos segs
58 for(int t=0;t<2;t++){
59 for(int i=0;i<2;i++){
60 bool c=true;
61 for(int k=0;k<2;k++){
62 point va = l1[i]-l2[k];
63 point vb = l2[!k]-l2[k];
64 ld ang = atan2(norm((vb^va)), vb*va);
65 if(ang>PI/2) c = false;
66 }
67 if(c)
68 ans = min(ans,dist_pt_seg(l1[i],l2));
69 }
70 swap(l1,l2);
71 }
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
102 ], v));
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][
111 i1], vet[0][j1],
112 {vet[1][
113 i2], vet[1][j2]}));
114
115 return ans;
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 }
1000 }

```

3.11 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
14     point operator+(const point &o) const{
15         return {x+o.x, y+o.y, z+o.z};
16     }
17     point operator-(const point &o) const{
18         return {x-o.x, y-o.y, z-o.z};
19     }
20     point operator*(cod t) const{
21         return {x*t, y*t, z*t};
22     }
23     point operator/(cod t) const{
24         return {x/t, y/t, z/t};
25     }
26     bool operator==(const point &o) const{
27         return eq(x, o.x) and eq(y, o.y) and eq(z, o.
28         z);
29     }
30     cod operator*(const point &o) const{ // dot
31         return x*o.x + y*o.y + z*o.z;
32     }
33     point operator^(const point &o) const{ // cross
34         return point(y*o.z - z*o.y,
35             z*o.x - x*o.z,
36             x*o.y - y*o.x);
37     }
38 };
39
40 ld dist(point a, point b){
41     return sqrt((a-b)*(a-b));
42 }
43
44 bool nulo(point a){
45     return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0))
46     ;
47 }
48
49 ld norm(point a){ // Modulo
50     return sqrt(a*a);
51 }
52 }

```

```

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
60 struct plane{
61     point p1, p2, p3;
62     plane(point p1=0, point p2=0, point p3=0): p1(p1)
63     , p2(p2), p3(p3){
64
65     point aux = (p1-p3)^(p2-p3);
66     cod a = aux.x, b = aux.y, c = aux.z;
67     cod d = -a*p1.x - b*p1.y - c*p1.z;
68     // ax+by+cz+d = 0;
69 };
70
71 cod dist(plane pl, point p){
72     return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d
73     ) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
74 }
75
76 point rotate(point v, point k, ld theta){
77     // Rotaciona o vetor v theta graus em torno do
78     // eixo k
79     // theta *= PI/180; // graus
80     return rotated = (v*cos(theta)) +
81     ((k^v)*sin(theta)) +
82     (k*(k^v))*(1-cos(theta));
83 }

```

3.12 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 }

```

3.13 Uniao-segmentos

```

1 int length_union(const vector<pii> &a){
2     int n = a.size();
3     vector<pair<int, bool>> x(n*2);
4     for(int i = 0; i < n; i++){
5         x[i*2] = {a[i].ff, false};
6         x[i*2+1] = {a[i].ss, true};
7     }
8
9     sort(x.begin(), x.end());
10
11     int result=0;
12     int c=0;
13     for(int i=0;i<2*n;i++){
14         if(i and c and x[i].ff>x[i-1].ff)
15             result += x[i].ff-x[i-1].ff;
16
17         if(x[i].ss) c--;
18         else c++;
19     }
20     return result;
21 }

```

3.14 Minkowski-Sum

```

1 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         }
10        vp c;
11        c.reserve(a.size() + b.size());
12        for(int k = 0; k < int(a.size()+b.size()); k++){
13            point pt{a[i] + b[j]};
14            if((int)c.size() >= 2 and !ccw(c[c.size()-2],
15                c.back(), pt))
16                c.pop_back();
17            c.pb(pt);
18            int q = i+1, w = j+1;
19            if(q == int(a.size())) q = 0;
20            if(w == int(b.size())) w = 0;
21            if(ccw(c.back(), a[i]+b[w], a[q]+b[j]) < 0) i
22                = q;
23            else j = w;
24        }
25
26        if(!ccw(c[0], c[(int)c.size()-1], c[(int)c.size()-2]))
27            c.pop_back();
28        if(!ccw(c.back(), c[0], c[1])){
29            c[0]=c.back();
30            c.pop_back();
31        }
32        c.shrink_to_fit();
33
34        return c;
35    }
36 }

```

3.15 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     }
9     return true;

```

3.16 Rotating-Callipers

```

1 int N;
2
3 int sum(int i, int x){
4     if(i+x>N-1) return (i+x-N);
5     return i+x;
6 }
7
8 ld rotating_callipers(vp &vet){
9     N = vet.size();
10    ld ans = 0;
11    // 2 triangulos (p1, p3, p4) (p1, p2, p3);
12    for(int i=0;i<N;i++){ // p1
13        int p2 = sum(i, 1); // p2
14        int p4 = sum(i, 3); // p4
15        for(int j=sum(i, 2);j!=i;j=sum(j, 1)){ // p3
16            if(j==p2) p2 = sum(p2, 1);
17            while(sum(p2, 1)!=j and areaT(vet[p2],
18                vet[i], vet[j]) < areaT(vet[sum(p2, 1)], vet[i],
19                vet[j]))
20                p2 = sum(p2, 1);

```

```

19         while(sum(p4, 1)!=i and areaT(vet[p4],
    vet[i], vet[j]) < areaT(vet[sum(p4, 1)], vet[i],
    vet[j]))
20             p4 = sum(p4, 1);
21
22         ans = max(ans, area(vet[i], vet[p2], vet[
23             j], vet[p4]));
24     }
25 }
26 return ans;
27 }

```

3.17 2D

```

1 #define vp vector<point>
2
3 // typedef int cod;
4 // bool eq(cod a, cod b){ return (a==b); }
5 typedef ld cod;
6 bool eq(cod a, cod b){ return abs(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
42 ld norm(point a){ // Modulo
43     return sqrt(a*a);
44 }
45 bool nulo(point a){
46     return (eq(a.x, 0) and eq(a.y, 0));
47 }
48
49 int ccw(point a, point b, point e){ //-1=dir; 0=
    collinear; 1=esq;
50     cod tmp = (b-a)^(e-a); // from a to b
51     return (tmp > EPS) - (tmp < -EPS);
52 }
53 point rotccw(point p, ld a){
54     // a = PI*a/180; // graus
55     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.x*sin(a)));
56 }
57 point rot90cw(point a) { return point(a.y, -a.x); };
58 point rot90ccw(point a) { return point(-a.y, a.x); };
59
60 ld proj(point a, point b){ // a sobre b
61     return a*b/norm(b);
62 }
63 ld angle(point a, point b){ // em radianos
64     ld ang = a*b / norm(a) / norm(b);
65     return acos(max(min(ang, (ld)1), (ld)-1));
66 }
67 ld angle_vec(point v){
68     // return 180/PI*atan2(v.x, v.y); // graus
69     return atan2(v.x, v.y);
70 }
71 ld order_angle(point a, point b){ // from a to b ccw
    (a in front of b)
72     ld aux = angle(a,b)*180/PI;
73     return ((a^b)<=0 ? aux:360-aux);
74 }
75 bool angle_less(point a1, point b1, point a2, point
    b2){ // ang(a1,b1) <= ang(a2,b2)
76     point p1((a1*b1), abs((a1^b1)));
77     point p2((a2*b2), abs((a2^b2)));
78     return (p1^p2) <= 0;
79 }
80
81 ld area(vp &p){ // (points sorted)
82     ld ret = 0;
83     for(int i=2;i<(int)p.size();i++)
84         ret += (p[i]-p[0])^(p[i-1]-p[0]);
85     return abs(ret/2);
86 }
87 ld areaT(point &a, point &b, point &c){
88     return abs((b-a)^(c-a))/2.0;
89 }
90
91 point center(vp &A){
92     point c = point();
93     int len = A.size();
94     for(int i=0;i<len;i++)
95         c=c+A[i];
96     return c/len;
97 }
98
99 point forca_mod(point p, ld m){
100     ld cm = norm(p);
101     if(cm<EPS) return point();
102     return point(p.x*m/cm,p.y*m/cm);
103 }
104
105
106 ///////////////
107 // Line //
108 ///////////////
109
110 struct line{
111     point p1, p2;
112     cod a, b, c; // ax+by+c = 0;
113     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
114     line(point p1=0, point p2=0): p1(p1), p2(p2){
115         a = p1.y-p2.y;
116         b = p2.x-p1.x;
117         c = -(a*p1.x + b*p1.y);
118     }
119     line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
    {
120         // Gera os pontos p1 p2 dados os coeficientes
121         // isso aqui eh um lixo mas quebra um galho
122         kkkkkk
123         if(b==0){
124             p1 = point(1, -c/a);
125             p2 = point(0, -c/a);
126         }else{

```

```

126         p1 = point(1, (-c-a*1)/b);
127         p2 = point(0, -c/b);
128     }
129 }
130
131 cod eval(point p){
132     return a*p.x+b*p.y+c;
133 }
134 bool inside(point p){
135     return eq(eval(p), 0);
136 }
137 point normal(){
138     return point(a, b);
139 }
140
141 bool inside_seg(point p){
142     return (inside(p) and
143         min(p1.x, p2.x)<=p.x and p.x<=max(p1
144 x, p2.x) and
145         min(p1.y, p2.y)<=p.y and p.y<=max(p1
146 y, p2.y));
147 }
148
149 vp inter_line(line l1, line l2){
150     ld det = l1.a*l2.b - l1.b*l2.a;
151     if(det==0) return {};
152     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
153     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
154     return {point(x, y)};
155 }
156
157 vp inter_seg(line l1, line l2){
158     vp ans = inter_line(l1, l2);
159     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
160 inside_seg(ans[0]))
161         return {};
162     return ans;
163 }
164 ld dseg(point p, point a, point b){ // point - seg
165     if(((p-a)*(b-a)) < EPS) return norm(p-a);
166     if(((p-b)*(a-b)) < EPS) return norm(p-b);
167     return abs((p-a)^(b-a))/norm(b-a);
168 }
169
170 ld dline(point p, line l){ // point - line
171     return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
172 }
173
174 line mediatrix(point a, point b){
175     point d = (b-a)*2;
176     return line(d.x, d.y, a*a - b*b);
177 }
178
179 line perpendicular(line l, point p){ // passes
180     through p
181     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
182 }
183
184 ///////////////
185 // Circle //
186 ///////////////
187
188 struct circle{
189     point c; cod r;
190     circle() : c(0, 0), r(0){}
191     circle(const point o) : c(o), r(0){}
192     circle(const point a, const point b){
193         c = (a+b)/2;
194         r = norm(a-c);
195     }
196     circle(const point a, const point b, const point
197 cc){
198         c = inter_line(mediatrix(a, b), mediatrix(b,
199 cc));
200         r = norm(a-c);
201     }
202     bool inside(const point &a) const{
203         return norm(a - c) <= r;
204     }
205     pair<point, point> getTangentPoint(point p) {
206         ld d1 = norm(p-c), theta = asin(r/d1);
207         point p1 = rotccw(c-p,-theta);
208         point p2 = rotccw(c-p,theta);
209         p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
210         p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
211         return {p1,p2};
212     }
213 };
214 // minimum circle cover O(n) amortizado
215 circle min_circle_cover(vector<point> v){
216     random_shuffle(v.begin(), v.end());
217     circle ans;
218     int n = v.size();
219     for(int i=0;i<n;i++){
220         if(!ans.inside(v[i])){
221             ans = circle(v[i]);
222             for(int j=0;j<i;j++){
223                 if(!ans.inside(v[j])){
224                     ans = circle(v[i], v[j]);
225                     for(int k=0;k<j;k++){
226                         if(!ans.inside(v[k])){
227                             ans = circle(v[i], v[j], v[k]);
228                         }
229                     }
230                 }
231             }
232         }
233     }
234     return ans;
235 }
236
237 circle incircle( point p1, point p2, point p3 ){
238     ld m1=norm(p2-p3);
239     ld m2=norm(p1-p3);
240     ld m3=norm(p1-p2);
241     point c = (p1*m1+p2*m2+p3*m3)*(1/(m1+m2+m3));
242     ld s = 0.5*(m1+m2+m3);
243     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3))/s;
244     return circle(c, r);
245 }
246
247 circle circumcircle(point a, point b, point c) {
248     circle ans;
249     point u = point((b-a).y, -(b-a).x);
250     point v = point((c-a).y, -(c-a).x);
251     point n = (c-b)*0.5;
252     ld t = (u^v)/(v^u);
253     ans.c = ((a+c)*0.5) + (v*t);
254     ans.r = norm(ans.c-a);
255     return ans;
256 }
257
258 vp inter_circle_line(circle C, line L){
259     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
260 p1)*(ab) / (ab*ab));
261     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
262 / (ab*ab);
263     if (h2 < 0) return {};
264     if (h2 == 0) return {p};
265     point h = (ab/norm(ab)) * sqrt(h2);
266     return {p - h, p + h};
267 }
268
269 vp inter_circle(circle C1, circle C2){
270     if(C1.c == C2.c) { assert(C1.r != C2.r); return

```

```

    {};}
263 point vec = C2.c - C1.c;
264 ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
    ;
265 ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
    C1.r*C1.r - p*p*d2;
266 if (sum*sum < d2 or dif*dif > d2) return {};
267 point mid = C1.c + vec*p, per = point(-vec.y, vec
    .x) * sqrt(max((ld)0, h2) / d2);
268 if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
269 return {mid + per, mid - per};
270 }

```

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 }

```

4.2 Prefixsum2D

```

1 ll find_sum(vector<vi> &mat, int x1, int y1, int x2,
    int y2){
2     // superior-esq(x1,y1) (x2,y2)inferior-dir
3     return mat[x2][y2]-mat[x2][y1-1]-mat[x1-1][y2]+
        mat[x1-1][y1-1];
4 }
5
6 int main(){
7
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 }

```

4.3 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 }

```

4.4 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 }

```

4.5 BIT-bigger-k

```
1 struct node{
2     int pos, l, r, val;
3     bool operator<(const node &o){
4         if(val==o.val) return l>o.l;
5         return val>o.val;
6     }
7 };
8
9 struct FT {
10     vector<int> bit; // indexado em 0
11     int n;
12
13     FT(int n) {
14         this->n = n+1;
15         bit.assign(n+1, 0);
16     }
17
18     int sum(int idx) {
19         int ret = 0;
20         for (; idx > 0; idx -= idx & -idx)
21             ret += bit[idx];
22         return ret;
23     }
24
25     int sum(int l, int r) {
26         return sum(r) - sum(l - 1);
27     }
28
29     void add(int idx, int delta) {
30         for (; idx < n; idx += idx & -idx)
31             bit[idx] += delta;
32     }
33 };
34
35 vi solveQuery(vi arr, vi ql, vi qr, vi qk){
36     // indexing [l, r] in 1
37     int n = arr.size();
38     int q = qk.size();
39     node a[n+q];
40
41     for(int i=0; i<n; i++){
42         a[i].val = arr[i];
43         a[i].pos = a[i].l = 0;
44         a[i].r = i+1;
45     }
46
47     for(int i=n; i<n+q; i++){
48         a[i].pos = i+1-n;
49         a[i].val = qk[i-n];
50         a[i].l = ql[i-n];
51         a[i].r = qr[i-n];
52     }
53     sort(a, a+n+q);
54
55     FT ft(n);
56     vi ans(q+1, 0);
57
58     for(int i=0; i<n+q; i++){
59         if(a[i].pos != 0)
60             ans[a[i].pos] = ft.sum(a[i].l, a[i].r);
61         else
62             ft.add(a[i].r, 1);
63     }
64     return ans;
65 }
66 int main()
67 {
68     vi arr = { 7, 3, 9, 13, 5, 4 };
69
70     vi QueryL = { 1, 2 };
71     vi QueryR = { 4, 6 };
```

```
72
73     vi QueryK = { 6, 8 };
74
75     solveQuery(arr, QueryL, QueryR, QueryK);
76
77     return 0;
78 }
```

4.6 BIT

```
1 struct FT {
2     vi bit; // indexado em 1
3     int n;
4
5     FT(int n) {
6         this->n = n+1;
7         bit.assign(n+2, 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) { // [l, 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 };
```

4.7 Minqueue

```
1 struct MinQ {
2     stack<pair<ll, ll>> in;
3     stack<pair<ll, ll>> out;
4
5     void add(ll val) {
6         ll minimum = in.empty() ? val : min(val, in.
7         top().ss);
8         in.push(mp(val, minimum));
9     }
10
11    ll pop() {
12        if(out.empty()) {
13            while(!in.empty()) {
14                ll val = in.top().ff;
15                in.pop();
16                ll minimum = out.empty() ? val : min(
17                val, out.top().ss);
18                out.push({val, minimum});
19            }
20        }
21        ll res = out.top().ff;
22        out.pop();
23        return res;
24    }
25
26    ll minn() {
27        ll minimum = LLINF;
28        if(in.empty() || out.empty())
29            minimum = in.empty() ? (ll)out.top().ss :
30            (ll)in.top().ss;
31        else
32            minimum = min((ll)in.top().ss, (ll)out.
33            top().ss);
34    }
```



```

31     return minimum;
32 }
33
34 ll size() {
35     return in.size() + out.size();
36 }
37 };

```

4.8 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 };

```

4.9 Mergesorttree

```

1 struct ST { // indexado em 0, 0(n * log^2(n) )
2     int size;
3     vector<vl> v;
4
5     vl f(vl a, vl& b) {
6         vl res = a;
7         for(auto val : b) {
8             res.pb(val);
9         }
10        sort(all(res));
11        return res;
12    }
13
14    void init(int n) {
15        size = 1;
16        while(size < n) size *= 2;
17        v.assign(2*size, vl());
18    }
19
20    void build(vector<ll>& a, int x, int lx, int rx) {
21        if(rx-lx == 1) {
22            if(lx < (int)a.size()) {
23                v[x].pb(a[lx]);
24            }
25            return;
26        }
27        int m = (lx+rx)/2;
28        build(a, 2*x + 1, lx, m);
29        build(a, 2*x + 2, m, rx);
30        v[x] = f(v[2*x + 1], v[2*x + 2]);
31    }

```

```

32
33    void build(vector<ll>& a) {
34        init(a.size());
35        build(a, 0, 0, size);
36    }
37
38    ll greaterequal(int l, int r, int k, int x, int
39    lx, int rx) {
40        if(r <= lx or l >= rx) return 0;
41        if(l <= lx && rx <= r) {
42            auto it = lower_bound(all(v[x]), k);
43            return (v[x].end() - it);
44        }
45        int m = (lx + rx)/2;
46        ll s1 = greaterequal(l, r, k, 2*x + 1, lx, m);
47        ll s2 = greaterequal(l, r, k, 2*x + 2, m, rx);
48
49        return s1 + s2;
50    }
51
52    ll greaterequal(int l, int r, int k) {
53        return greaterequal(l, r+1, k, 0, 0, size);
54    }
55 };

```

4.10 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 struct Sparse {
8     int n;
9     vector<vi> st;
10
11    Sparse(vi& v) {
12        n = v.size();
13        int k = logv[n];
14        st.assign(n+1, vi(k+1, 0));
15
16        forn(i, n) {
17            st[i][0] = v[i];
18        }
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
27        int f(int a, int b) {
28            return min(a, b);
29        }
30
31        int query(int l, int r) {
32            int k = logv[r-l+1];
33            return f(st[l][k], st[r - (1 << k) + 1][k]);
34        }
35 };

```

4.11 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 };

```

4.12 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 m
14     *x+b
15     bool bad(iterator y){
16         auto z = next(y);
17         if(y == begin()){
18             if(z == end()) return 0;
19             return y->m == z->m && y->b <= z->b;
20         }
21         auto x = prev(y);
22         if(z == end()) return y->m == x->m && y->b <=
23         x->b;
24         return (ld)(x->b - y->b)*(z->m - y->m) >= (ld
25         )(y->b - z->b)*(y->m - x->m);
26     }
27     void insert_line(ll m, ll b){ // min -> insert (-
28         m,-b) -> -eval()
29         auto y = insert({ m, b });
30         y->succ = [=]{ return next(y) == end() ? 0 :
31         &*next(y); };
32         if(bad(y)){ erase(y); return; }
33         while(next(y) != end() && bad(next(y))) erase
34         (next(y));
35         while(y != begin() && bad(prev(y))) erase(
36         prev(y));
37     }
38     ll eval(ll x){
39         auto l = *lower_bound((Line) { x, is_query })
40         ;
41         return l.m * x + l.b;
42     }
43 };

```

4.13 Mo

```

1  const int BLK = 600; // tamanho do bloco, algo entre
2      500 e 700 eh nice
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         if(1/BLK != other.l/BLK)
12             return 1/BLK < other.l/BLK;
13         return (1/BLK & 1) ? r < other.r : r > other.
14         r;
15     }
16 };
17 inline void add() {}
18 inline void remove() {} // implementar operacoes de
19     acordo com o problema
20 vector<int> mo(vector<Query>& queries) {
21     vector<int> res(queries.size());
22     sort(queries.begin(), queries.end());
23     resposta = 0;
24
25     int l = 0, r = -1;
26     for(Query q : queries) {
27         while(l > q.l) {
28             l--;
29             add(l);
30         }
31         while(r < q.r) {
32             r++;
33             add(r);
34         }
35         while(l < q.l) {
36             remove(l);
37             l++;
38         }
39         while(r > q.r) {
40             remove(r);
41             r--;
42         }
43         res[q.idx] = resposta; // adicionar resposta
44         de acordo com o problema
45     }
46     return res; // ordenar o vetor pelo indice e
47     responder queries na ordem

```

5 DP

5.1 Mochila

```

1  int val[MAXN], peso[MAXN], dp[MAXN][MAXS];
2
3  int knapsack(int n, int m){ // n Objetos | Peso max
4      for(int i=0;i<=n;i++){
5          for(int j=0;j<=m;j++){
6              if(i==0 or j==0)
7                  dp[i][j] = 0;
8              else if(peso[i-1]<=j)
9                  dp[i][j] = max(val[i-1]+dp[i-1][j-
10                  peso[i-1]], dp[i-1][j]);
11              else
12                  dp[i][j] = dp[i-1][j];

```

```

13     }
14     return dp[n][m];
15 }
16
17 // space optimized
18
19 int val[MAX], wt[MAX], dp[MAX];
20 int knapsack(int n, int W){
21     for(int i=0; i < n; i++)
22         for(int j=W; j>=wt[i]; j--)
23             dp[j] = max(dp[j], val[i] + dp[j-wt[i]]);
24     return dp[W];
25 }
26 }

```

5.2 Dp-digitos

```

1 // dp de quantidade de numeros <= r com ate qt
  digitos diferentes de 0
2 ll dp(int idx, string& r, bool menor, int qt, vector<
  vector<vi>>& tab) {
3     if(qt > 3) return 0;
4     if(idx >= r.size()) {
5         return 1;
6     }
7     if(tab[idx][menor][qt] != -1)
8         return tab[idx][menor][qt];
9
10    ll res = 0;
11    for(int i = 0; i <= 9; i++) {
12        if(menor or i <= r[idx]-'0') {
13            res += dp(idx+1, r, menor or i < (r[idx]-
14                '0'), qt+(i>0), tab);
15        }
16    }
17    return tab[idx][menor][qt] = res;
18 }

```

5.3 Unbounded-Knapsack

```

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

```

5.4 Largest-KSubmatrix

```

1 int n, m;
2 int a[MAX][MAX];
3 // Largest K such that exists a block K*K with equal
  numbers
4 int largestKSubmatrix(){
5     int dp[n][m];
6     memset(dp, 0, sizeof(dp));
7
8     int result = 0;
9     for(int i = 0 ; i < n ; i++){
10         for(int j = 0 ; j < m ; j++){
11             if(!i or !j)
12                 dp[i][j] = 1;

```

```

13         else if(a[i][j] == a[i-1][j] and
14             a[i][j] == a[i][j-1] and
15             a[i][j] == a[i-1][j-1])
16             dp[i][j] = min(min(dp[i-1][j], dp[i][
17                 j-1]),
18                     dp[i-1][j-1]) + 1;
19         else dp[i][j] = 1;
20         result = max(result, dp[i][j]);
21     }
22 }
23
24 return result;
25 }

```

5.5 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();
10
11 // see that later
12 // https://codeforces.com/blog/entry/13225?#comment
  -180208
13
14 vi LIS(const vi &elements){
15     auto compare = [&](int x, int y) {
16         return elements[x] < elements[y];
17     };
18     set< int, decltype(compare) > S(compare);
19
20     vi previous( elements.size(), -1 );
21     for(int i=0; i<int( elements.size() ); ++i){
22         auto it = S.insert(i).first;
23         if(it != S.begin())
24             previous[i] = *prev(it);
25         if(*it == i and next(it) != S.end())
26             S.erase(next(it));
27     }
28
29     vi answer;
30     answer.push_back( *S.rbegin() );
31     while ( previous[answer.back()] != -1 )
32         answer.push_back( previous[answer.back()] );
33     reverse( answer.begin(), answer.end() );
34     return answer;
35 }

```

5.6 Partition-Problem

```

1 // Partition Problem DP O(n2)
2 bool findPartition(vi &arr){
3     int sum = 0;
4     int n = arr.size();
5
6     for(int i=0; i<n; i++)
7         sum += arr[i];
8
9     if(sum&1) return false;
10
11     bool part[sum/2+1][n+1];
12
13     for(int i=0; i<=n; i++){
14         part[0][i] = true;
15
16         for(int i=1; i<=sum/2; i++)

```

```

17     part[i][0] = false;
18
19     for(int i=1; i<=sum/2; i++){
20         for(int j=1; j<=n; j++){
21             part[i][j] = part[i][j-1];
22             if(i >= arr[j-1])
23                 part[i][j] |= part[i - arr[j-1]][j
24             -1];
25         }
26     }
27     return part[sum / 2][n];
28 }

```

6 Math

6.1 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, O(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+p; i<=n; i+=p){
32                 isprime[i] = false;
33                 tot[i] = (tot[i]/p)*(p-1);
34             }
35         }
36     }
37     return tot;
38 }

```

6.2 Double-gcd

```

1 ld gcdf(ld a, ld b){
2     if(a<b) return gcdf(b, a);
3
4     if(fabs(b)<EPS)
5         return a;
6     else
7         return (gcdf(b, a - floor(a/b)*b));
8 }

```

6.3 Linear-Diophantine-Equation

```

1 // Linear Diophantine Equation
2 int gcd(int a, int b, int &x, int &y)

```

```

3 {
4     if (a == 0)
5     {
6         x = 0; y = 1;
7         return b;
8     }
9     int x1, y1;
10    int d = gcd(b%a, a, x1, y1);
11    x = y1 - (b / a) * x1;
12    y = x1;
13    return d;
14 }
15
16 bool find_any_solution(int a, int b, int c, int &x0,
17 int &y0, int &g)
18 {
19     g = gcd(abs(a), abs(b), x0, y0);
20     if (c % g)
21         return false;
22
23     x0 *= c / g;
24     y0 *= c / g;
25     if (a < 0) x0 = -x0;
26     if (b < 0) y0 = -y0;
27     return true;
28 }
29 // All solutions
30 // x = x0 + k*b/g
31 // y = y0 - k*a/g

```

6.4 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             div.pb(i);
8             while(n%i==0)
9                 n/=i;
10        }
11    }
12    if(n!=1) div.pb(n);
13    return div;
14 }

```

6.5 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 }

```

6.6 Mulmod

```

1 ll mulmod(ll a, ll b) {
2     if(a == 0) {
3         return 0LL;
4     }
5     if(a%2 == 0) {
6         ll val = mulmod(a/2, b);
7         return (val + val) % MOD;
8     }
9     else {
10        ll val = mulmod((a-1)/2, b);
11        val = (val + val) % MOD;
12        return (val + b) % MOD;
13    }
14 }

```

6.7 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
10                [j].ff);
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 -
24                d[j].ss);
25         x += xi;
26     }
27     return x;
28 }

```

6.8 Crt

```

1 tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
2     if (!a) return {b, 0, 1};
3     auto [g, x, y] = ext_gcd(b%a, a);
4     return {g, y - b/a*x, x};
5 }
6
7 struct crt {
8     ll a, m;
9
10    crt() : a(0), m(1) {}
11    crt(ll a_, ll m_) : a(a_), m(m_) {}
12    crt operator * (crt C) {
13        auto [g, x, y] = ext_gcd(m, C.m);
14        if ((a - C.a) % g) a = -1;
15        if (a == -1 or C.a == -1) return crt(-1, 0);
16        ll lcm = m/g*C.m;
17        ll ans = a + (x*(C.a-a)/g % (C.m/g))*m;
18        return crt((ans % lcm + lcm) % lcm, lcm);
19    }
20 };

```

6.9 Miller-Habin

```

1 ll mul(ll a, ll b, ll m) {
2     return (a*b-ll(a*(long double)b/m+0.5)*m+m)%m;
3 }
4
5 ll expo(ll a, ll b, ll m) {
6     if (!b) return 1;
7     ll ans = expo(mul(a, a, m), b/2, m);
8     return b%2 ? mul(a, ans, m) : ans;
9 }
10
11 bool prime(ll n) {
12     if (n < 2) return 0;
13     if (n <= 3) return 1;
14     if (n % 2 == 0) return 0;
15
16     ll d = n - 1;
17     int r = 0;
18     while (d % 2 == 0) {
19         r++;
20         d /= 2;
21     }
22
23     // com esses primos, o teste funciona garantido
    para n <= 2^64

```

```

24 // funciona para n <= 3*10^24 com os primos ate
25 41
26 for (int i : {2, 325, 9375, 28178, 450775,
27 9780504, 1795265022}) {
28     if (i >= n) break;
29     ll x = expo(i, d, n);
30     if (x == 1 or x == n - 1) continue;
31
32     bool deu = 1;
33     for (int j = 0; j < r - 1; j++) {
34         x = mul(x, x, n);
35         if (x == n - 1) {
36             deu = 0;
37             break;
38         }
39     }
40     if (deu) return 0;
41 }

```

6.10 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 }

```

6.11 Pollard-Rho

```

1 mt19937 rng((int) chrono::steady_clock::now().
2     time_since_epoch().count());
3
4 ll uniform(ll l, ll r){
5     uniform_int_distribution<ll> uid(l, r);
6     return uid(rng);
7 }
8
9 ll mul(ll a, ll b, ll m) {
10     ll ret = a*b - ll(a*(long double)b/m+0.5)*m;
11     return ret < 0 ? ret+m : ret;
12 }
13
14 ll expo(ll a, ll b, ll m) {
15     if (!b) return 1;
16     ll ans = expo(mul(a, a, m), b/2, m);
17     return b%2 ? mul(a, ans, m) : ans;
18 }
19
20 bool prime(ll n) {
21     if (n < 2) return 0;
22     if (n <= 3) return 1;
23     if (n % 2 == 0) return 0;
24
25     ll d = n - 1;
26     int r = 0;
27     while (d % 2 == 0) {
28         r++;
29         d /= 2;
30     }
31
32     for (int i : {2, 325, 9375, 28178, 450775,
33 9780504, 795265022}) {

```

```

34     if (i >= n) break;
35     ll x = expo(i, d, n);
36     if (x == 1 or x == n - 1) continue;
37
38     bool deu = 1;
39     for (int j = 0; j < r - 1; j++) {
40         x = mul(x, x, n);
41         if (x == n - 1) {
42             deu = 0;
43             break;
44         }
45     }
46     if (deu) return 0;
47 }
48
49 ll rho(ll n) {
50     if (n == 1 or prime(n)) return n;
51     if (n % 2 == 0) return 2;
52
53     while (1) {
54         ll x = 2, y = 2, ciclo = 2, i = 0, d = 1;
55         ll c = uniform(1, n-1);
56
57         while (d == 1) {
58             if (++i == ciclo) ciclo *= 2, y = x;
59             x = (mul(x, x, n) + c) % n;
60
61             if (x == y) break;
62
63             d = __gcd(abs(x-y), n);
64         }
65
66         if (x != y) return d;
67     }
68 }
69
70 void fact(ll n, vector<ll>& v) {
71     if (n == 1) return;
72     if (prime(n)) v.pb(n);
73     else {
74         ll d = rho(n);
75         fact(d, v);
76         fact(n / d, v);
77     }
78 }

```

6.12 Verif-primo

```

1 // Prime verification sqrt(N)
2
3 bool prime(ll x){
4     if(x==2) return true;
5     else if(x==1 or x%2==0) return false;
6     for(ll i=3;i*i<=x;i+=2)
7         if(x%i==0)
8             return false;
9     return true;
10 }

```

6.13 Mobius

```

1 vi mobius(int n) {
2     // g(n) = sum{f(d)} => f(n) = sum{mu(d)*g(n/d)}
3     vi mu(n+1);
4     mu[1] = 1; mu[0] = 0;
5     for(int i = 1; i <= n; i++)
6         for(int j = i + i; j <= n; j += i)
7             mu[j] -= mu[i];
8
9     return mu;
10 }

```

6.14 Exponenciacao-matriz

```
1 struct Matrix {
2     vector<vl> m;
3     int r, c;
4
5     Matrix(vector<vl> mat) {
6         m = mat;
7         r = mat.size();
8         c = mat[0].size();
9     }
10
11     Matrix(int row, int col, bool ident=false) {
12         r = row; c = col;
13         m = vector<vl>(r, vl(c, 0));
14         if(ident) {
15             for(int i = 0; i < min(r, c); i++) {
16                 m[i][i] = 1;
17             }
18         }
19     }
20
21     Matrix operator*(const Matrix &o) const {
22         assert(c == o.r); // garantir que da pra
23         multiplicar
24         vector<vl> res(r, vl(o.c, 0));
25
26         for(int i = 0; i < r; i++) {
27             for(int j = 0; j < o.c; j++) {
28                 for(int k = 0; k < c; k++) {
29                     res[i][j] = (res[i][j] + m[i][k]*
30                     o.m[k][j]) % MOD;
31                 }
32             }
33         }
34         return Matrix(res);
35     };
36
37     Matrix fexp(Matrix b, int e, int n) {
38         if(e == 0) return Matrix(n, n, true); //
39         identidade
40         Matrix res = fexp(b, e/2, n);
41         res = (res * res);
42         if(e%2) res = (res * b);
43         return res;
44     }
45 }
```

6.15 Crivo

```
1 // Sieve of Eratosthenes
2
3 vector<bool> primos(n+1, true);
4
5 primos[0]=primos[1]=false;
6
7 for(int i=2;i<=n;i++)
8     if(primos[i])
9         for(int j=i+i; j<=n; j+=i)
10             primos[j]=false;
```

6.16 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 }
```

6.17 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 }
```

6.18 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
19 void ensure_base(int nbase){
20     if(nbase <= base)
21         return;
22
23     rev.resize(1 << nbase);
24     for(int i = 0; i < (1 << nbase); i++)
25         rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (
26         nbase - 1));
27
28     roots.resize(1 << nbase);
29
30     while(base < nbase){
31         ld angle = 2*PI / (1 << (base + 1));
32         for(int i = 1 << (base - 1); i < (1 << base);
33         i++){
34             roots[i << 1] = roots[i];
35             ld angle_i = angle * (2 * i + 1 - (1 <<
36             base));
```

```

31         roots[(i << 1) + 1] = num(cos(angle_i),
32         sin(angle_i));
33     }
34     base++;
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]))
78             * r;
79         if(i != j) {
80             fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[
81 j])) * r;
82             fa[i] = z;
83         }
84         fft(fa, sz);
85         vi res(need);
86         for(int i = 0; i < need; i++)
87             res[i] = fa[i].x + 0.5;
88     }
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
101     for(int i=0;i<(int)a.size();i++){
102         int x = (a[i] % m + m) % m;
103         fa[i] = num(x & ((1 << 15) - 1), x >> 15);
104     }
105     fill(fa.begin() + a.size(), fa.begin() + sz, num
106 {0, 0});
107     fft(fa, sz);
108     if(sz > (int) fb.size())
109         fb.resize(sz);
110     if(eq)
111         copy(fa.begin(), fa.begin() + sz, fb.begin())
112 ;
113     else{
114         for(int i = 0; i < (int) b.size(); i++){
115             int x = (b[i] % m + m) % m;
116             fb[i] = num(x & ((1 << 15) - 1), x >> 15)
117 ;
118         }
119         fill(fb.begin() + b.size(), fb.begin() + sz,
120 num {0, 0});
121         fft(fb, sz);
122     }
123     ld ratio = 0.25 / sz;
124     num r2(0, -1);
125     num r3(ratio, 0);
126     num r4(0, -ratio);
127     num r5(0, 1);
128     for(int i=0;i<=(sz >> 1);i++) {
129         int j = (sz - i) & (sz - 1);
130         num a1 = (fa[i] + conj(fa[j]));
131         num a2 = (fa[i] - conj(fa[j])) * r2;
132         num b1 = (fb[i] + conj(fb[j])) * r3;
133         num b2 = (fb[i] - conj(fb[j])) * r4;
134         if(i != j){
135             num c1 = (fa[j] + conj(fa[i]));
136             num c2 = (fa[j] - conj(fa[i])) * r2;
137             num d1 = (fb[j] + conj(fb[i])) * r3;
138             num d2 = (fb[j] - conj(fb[i])) * r4;
139             fa[i] = c1 * d1 + c2 * d2 * r5;
140             fb[i] = c1 * d2 + c2 * d1;
141         }
142         fa[j] = a1 * b1 + a2 * b2 * r5;
143         fb[j] = a1 * b2 + a2 * b1;
144     }
145     fft(fa, sz);
146     fft(fb, sz);
147     vi res(need);
148     for(int i=0;i<need;i++){
149         ll aa = fa[i].x + 0.5;
150         ll bb = fb[i].x + 0.5;
151         ll cc = fa[i].y + 0.5;
152         res[i] = (aa + ((bb % m) << 15) + ((cc % m)
153 << 30)) % m;
154     }
155     return res;
156 }
157
158 int main()
159 {sws;
160
161     //FFT
162     vi fx{1, 2, 3}; // 1+2x+3x^2
163     vi gx{4, 5}; // 4+5x
164     vi res;
165
166     res = multiply(fx,gx); //4 + 13x + 22x^2 + 15x^3
167
168     return 0;
169 }

```


6.19 Modular-Exponentiaion

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

6.20 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));
```

6.21 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
29     for(auto f : fat) {
30         if(fexp(a, phi/f, mod) == 1)
31             return false;
32     }
33
34     return true;
35 }
36
37 // mods com raizes primitivas: 2, 4, p^k, 2*p^k, p eh
38 // primo impar, k inteiro --- O(n log^2(n))
39 ll achar_raiz(ll mod, ll phi) {
40     if(mod == 2) return 1;
41     vl fat, elementos;
42     fat = fatorar(phi);
43
44     for(ll i = 2; i <= mod-1; i++) {
45         if(raiz_prim(i, mod, phi, fat))
46             return i;
```

```
45     }
46
47     return -1; // retorna -1 se nao existe
48 }
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 }
```

6.22 Kamenetsky

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

7 Misc

7.1 Ordered-Set

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3
4 #include <ext/pb_ds/detail/standard_policies.hpp>
5
6 using namespace __gnu_pbds; // or pb_ds;
7
8 template<typename T, typename B = null_type>
9 using ordered_set = tree<T, B, less<T>, rb_tree_tag,
10 tree_order_statistics_node_update>;
11 // find_by_order / order_of_key
```

7.2 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)
```

```

8 NOT - ~a // The result is 11111010
(250)
9 Left shift - b<<1 // The result is 00010010
(18)
10 Right shift - b>>1 // The result is 00000100
(4)

11 // Exchange two int variables
12
13 a^=b;
14 b^=a;
15 a^=b;
16
17 // Even or Odd
18
19 (x & 1)? printf("Odd"): printf("Even");
20
21 // Turn on the j-th bit
22
23 int S = 34; //(100010)
24 int j = 3;
25
26 S = S | (1<<j);
27
28 // Turn off the j-th bit
29
30 int S = 42; //(101010)
31 int j = 1;
32
33 S &= ~(1<<j)
34
35 S == 40 //(101000)
36
37 // Check the j-th element
38
39 int S = 42; //(101010)
40 int j = 3;
41
42 T = S & (1<<j); // T = 0
43
44 // Least significant bit (lsb)
45
46 int lsb(int x){ return x&-x; }
47
48 // Exchange o j-th element
49
50 S ^= (1<<j)
51
52 // Position of the first bit on
53
54 T = (S & (-S))
55 T -> 4 bit ligado //(1000)
56
57 // Most significant digit of N
58
59 double K = log10(N);
60 K = K - floor(K);
61 int X = pow(10, K);
62
63 // Number of digits in N
64
65 X =floor(log10(N)) + 1;
66
67 // Power of two
68
69 bool isPowerOfTwo(int x){ return x && !(x&(x
-1))); }
70
71 // Turn off the first bit 1
72 m = m & (m-1);
73
74 // Built-in functions
75
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); }

```

7.3 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 }

```

7.4 Trie-bits

```

1 struct Trie{
2
3     int trie[MAX][10];
4     bool finish[MAX];
5     int nxt = 1, len = 0;
6
7     void add(string &s){
8         int node = 0;
9         for(auto c: s){
10             if(trie[node][c-'0'] == 0){
11                 node = trie[node][c-'0'] = nxt;
12                 nxt++;
13             }else
14                 node = trie[node][c-'0'];
15         }
16         if(!finish[node]){
17             finish[node] = true;
18             len++;
19         }
20     }
21
22     bool find(string &s, bool remove){
23         int idx = 0;
24         for(auto c: s)

```

```

25         if(trie[idx][c-'0'] == 0)
26             return false;
27         else
28             idx = trie[idx][c-'0'];
29         if(remove and finish[idx]){
30             finish[idx]=false;
31             len--;
32         }
33         return finish[idx];
34     }
35
36     bool find(string &s){
37         return find(s, 0);
38     }
39
40     void del(string &s){
41         find(s, 1);
42     }
43
44     string best_xor(string s){
45         int idx = 0;
46         string ans;
47         for(auto c: s){
48             char other='1'; if(c=='1') other='0';
49
50             if(trie[idx][other-'0'] != 0){
51                 idx = trie[idx][other-'0'];
52                 if(other=='1') ans.pb('1');
53                 else ans.pb('0');
54             }else{
55                 idx = trie[idx][c-'0'];
56                 if(c=='1') ans.pb('1');
57                 else ans.pb('0');
58             }
59         }
60
61         return ans;
62     }
63
64 };
65
66 string sbits(ll n){
67     string ans;
68     for(int i=0;i<64;i++)
69         ans.pb(!(n & 1LL<<i)+'0');
70     return ans;
71 }

```

7.5 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);cout.tie(NULL);
16 #define endl '\n'
17 #define teto(a, b) ((a+b-1)/(b))
18 #define all(x) x.begin(), x.end()
19 #define forn(i, n) for(int i = 0; i < (int)n; i++)
20 #define forne(i, a, b) for(int i = a; i <= b; i++)
21 #define dbg(msg, var) cerr << msg << " " << var <<
22     endl;

```

```

22 using namespace std;
23
24 const int MAX = 200010;
25 const int MOD = 1000000007;
26 const int INF = 1e8;
27 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
28 const ld EPS = 1e-7;
29
30 // End Template //

```

7.6 Rand

```

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

```

7.7 Safe-Map

```

1 struct custom_hash {
2     static uint64_t splitmix64(uint64_t x) {
3         // http://xorshift.di.unimi.it/splitmix64.c
4         x += 0x9e3779b97f4a7c15;
5         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
6         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
7         return x ^ (x >> 31);
8     }
9
10    size_t operator()(uint64_t x) const {
11        static const uint64_t FIXED_RANDOM = chrono::
12            steady_clock::now().time_since_epoch().count();
13        return splitmix64(x + FIXED_RANDOM);
14    }
15 };
16 unordered_map<long long, int, custom_hash> safe_map;

```

8 Strings

8.1 Trie

```

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

```

```

29         if(remove and finish[idx]){
30             finish[idx]=false;
31             len--;
32         }
33         return finish[idx];
34     }
35
36     bool find(string &s){
37         return find(s, 0);
38     }
39
40     void del(string &s){
41         find(s, 1);
42     }
43
44 };

```

8.2 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 }

```

8.3 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    }
29    return p;

```

```

29 }

```

8.4 LCS

```

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

```

8.5 LCSubseq

```

1 // Longest Common Subsequence
2 string lcs(string x, string y){
3     int n = x.size(), m = y.size();
4     vector<vi> dp(n+1, vi(m+1, 0));
5
6     for(int i=0; i<=n; i++){
7         for(int j=0; j<=m; j++){
8             if(!i or !j)
9                 dp[i][j]=0;
10            else if(x[i-1] == y[j-1])
11                dp[i][j]=dp[i-1][j-1]+1;
12            else
13                dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
14        }
15    }
16
17    // int len = dp[n][m];
18    string ans="";
19
20    // recover string
21    int i = n-1, j = m-1;
22    while(i>=0 and j>=0){
23        if(x[i] == y[j]){
24            ans.pb(x[i]);
25            i--; j--;
26        }else if(dp[i][j+1]>dp[i+1][j])
27            i--;
28        else
29            j--;
30    }
31
32    reverse(ans.begin(), ans.end());

```

```

33
34     return ans;
35 }

```

8.6 Edit-distance

```

1 int edit_distance(int a, int b, string& s, string& t)
2 {
3     // indexado em 0, transforma s em t
4     if(a == -1) return b+1;
5     if(b == -1) return a+1;
6     if(tab[a][b] != -1) return tab[a][b];
7
8     int ins = INF, del = INF, mod = INF;
9     ins = edit_distance(a-1, b, s, t) + 1;
10    del = edit_distance(a, b-1, s, t) + 1;
11    mod = edit_distance(a-1, b-1, s, t) + (s[a] != t[
12    b]);
13
14    return tab[a][b] = min(ins, min(del, mod));
15 }

```

8.7 Pal-int

```

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

```

8.8 Z-Func

```

1 vi z_algo(const string &s)
2 {
3     // returns vector for each idx where a prefix of
4     // size i starts.
5     int n = s.size();
6     int L = 0, R = 0;
7     vi z(n, 0);
8     for(int i = 1; i < n; i++){
9         if(i <= R)
10            z[i] = min(z[i-L], R - i + 1);
11        while(z[i]+i < n and s[ z[i]+i ] == s[ z[i]
12        ])
13            z[i]++;
14        if(i+z[i]-1 > R){
15            L = i;
16            R = i + z[i] - 1;
17        }
18    }
19    return z;
20 }

```

8.9 Hash

```

1 struct Hash {
2     vector<unordered_set<ll>> h;
3     vector<ll> mods = {
4
5         1000000009,1000000021,1000000033,1000000087,1000000093,
6
7         1000000123,1000000181,1000000207,1000000223,1000000241,
8
9     };
10    ll p = 31;
11    int num;
12
13    Hash(int qt) {
14        srand(time(0));
15        num = qt;
16        h.assign(num, unordered_set<ll>());
17        random_shuffle(all(mods));
18    }
19
20    ll compute_hash(string const& s, ll p, ll m) {
21        ll res = 0, p_pow = 1;
22
23        for(char c : s) {
24            res = ( res + (c-'a'+1) * p_pow) % m;
25            p_pow = (p_pow * p) % m;
26        }
27        return res;
28    }
29
30    void add(string const& s) {
31        forn(i, num) {
32            ll value = compute_hash(s, p, mods[i]);
33            h[i].insert(value);
34        }
35    }
36
37    bool query(string const& s) {
38        forn(i, num) {
39            ll val = compute_hash(s, p, mods[i]);
40            if(!h[i].count(val))
41                return false;
42        }
43        return true;
44    }
45 };

```

8.10 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    }
27 }

```

```

20     }
21     d2[i] = k--;
22     if(i + k > r) {
23         l = i - k - 1;
24         r = i + k ;
25     }
26 }
27 }

```

8.11 Suffix-array-radix

```

1 void radix_sort(vector<pii>& rnk, vi& ind) {
2     auto counting_sort = [](vector<pii>& rnk, vi& ind
3 ) {
4         int n = ind.size(), maxx = -1;
5         for(auto p : rnk) maxx = max(maxx, p.ff);
6
7         vi cnt(maxx+1, 0), pos(maxx+1), ind_new(n);
8         for(auto p : rnk) cnt[p.ff]++;
9         pos[0] = 0;
10
11         for(int i = 1; i <= maxx; i++) {
12             pos[i] = pos[i-1] + cnt[i-1];
13         }
14
15         for(auto idx : ind) {
16             int val = rnk[idx].ff;
17             ind_new[pos[val]] = idx;
18             pos[val]++;
19         }
20
21         swap(ind, ind_new);
22     };
23
24     for(int i = 0; i < (int)rnk.size(); i++) swap(rnk
25 [i].ff, rnk[i].ss);
26     counting_sort(rnk, ind);
27 }
28
29 vi suffix_array(string s) {
30     int n = s.size();
31     vector<pii> rnk(n, mp(0, 0));
32     vi ind(n);
33     forn(i, n) {
34         rnk[i].ff = (s[i] == '$') ? 0 : s[i] - 'a' + 1;
35         // manter '$' como 0

```

```

35         ind[i] = i;
36     }
37
38     for(int k = 1; k <= n; k = (k << 1)) {
39         for(int i = 0; i < n; i++) {
40             if(ind[i]+k >= n) {
41                 rnk[ind[i]].ss = 0;
42             }
43             else {
44                 rnk[ind[i]].ss = rnk[ind[i]+k].ff;
45             }
46         }
47         radix_sort(rnk, ind); // sort(all(rnk), cmp)
48         pra n*log(n), cmp com rnk[i] < rnk[j]
49
50         vector<pii> tmp = rnk;
51         tmp[ind[0]] = mp(1, 0); // rnk.ff começar em
52         1 pois '$' eh o 0
53         for(int i = 1; i < n; i++) {
54             tmp[ind[i]].ff = tmp[ind[i-1]].ff;
55             if(rnk[ind[i]] != rnk[ind[i-1]]) {
56                 tmp[ind[i]].ff++;
57             }
58         }
59         swap(rnk, tmp);
60     }
61     return ind;
62 }
63
64 vi lcp_array(string s, vi sarray) {
65     vi inv(s.size());
66     for(int i = 0; i < (int)s.size(); i++) {
67         inv[sarray[i]] = i;
68     }
69     vi lcp(s.size());
70     int k = 0;
71     for(int i = 0; i < (int)s.size()-1; i++) {
72         int pi = inv[i];
73         if(pi-1 < 0) continue;
74         int j = sarray[pi-1];
75
76         while(s[i+k] == s[j+k]) k++;
77         lcp[pi] = k;
78         k = max(k-1, 0);
79     }
80
81     return vi(lcp.begin()+1, lcp.end()); // LCP(i, j)
82     = min(lcp[i], ..., lcp[j-1])

```