



Notebook - Maratona de Programação

Tiago de Souza Fernandes

Sumário

1 Algoritmos	2	3.14 Minkowski-Sum	10
1.1 Meet-in-the-middle	2	3.15 Simetria-central	10
1.2 Iterative-BS	2	3.16 Rotating-Callipers	11
2 Grafos	2	3.17 2D	11
2.1 BFS-01	2	4 ED	13
2.2 BFS	2	4.1 Trie	13
2.3 2SAT	2	4.2 Prefixsum2D	13
2.4 Find-bridges	3	4.3 Delta-Encoding	13
2.5 HLD-Vertice	3	4.4 BIT-2D	13
2.6 Kahn	3	4.5 BIT-bigger-k	13
2.7 Dijkstra	4	4.6 BIT	14
2.8 LCA	4	4.7 Minqueue	14
2.9 HLD-Aresta	4	4.8 BIT-kth	14
2.10 Floyd-Warshall	5	4.9 Mergesorttree	15
2.11 Kruskal	5	4.10 Sparse-Table	15
2.12 DFS	5	4.11 Union-Find	15
2.13 Topological-sort	5	4.12 CHT	16
2.14 Kosaraju	5	4.13 Mo	16
2.15 Centroid	5	5 DP	16
2.16 Dinic	6	5.1 Mochila	16
2.17 Prim	6	5.2 Dp-digito	16
3 Geometria	7	5.3 Unbounded-Knapsack	17
3.1 NumIntersectionLine	7	5.4 Largest-KSubmatrix	17
3.2 Inside-polygon	7	5.5 LIS	17
3.3 Polygon-Diameter	7	5.6 Partition-Problem	17
3.4 MinDistPair	8	6 Math	17
3.5 Intersect-polygon	8	6.1 Totient	17
3.6 Sort-by-Angle	8	6.2 Double-gcd	18
3.7 Convex-Hull	8	6.3 Linear-Diophantine-Equation	18
3.8 Inter-Retangulos	8	6.4 Factorization-sqrt	18
3.9 Half-Plane-Intersect	8	6.5 FFT-simple	18
3.10 Tetrahedron-Distance3D	8	6.6 Mulmod	19
3.11 3D	9	6.7 Lagrange-interpolation	19
3.12 Heron	10	6.8 Crt	19
3.13 Uniao-segmentos	10	6.9 Miller-Habin	19
		6.10 Inverso-Mult	19

6.11	Pollard-Rho	20
6.12	Verif-primo	20
6.13	Mobius	20
6.14	Crivo	20
6.15	Bigmod	20
6.16	Simpson's-formula	21
6.17	FFT-tourist	21
6.18	Modular-Exponentiaion	22
6.19	Next-Permutation	22
6.20	Recurso-linear	22
6.21	Raiz-primitiva	23
6.22	Kamenetsky	23
7	Misc	23
7.1	Bitwise	23
7.2	All-Subsets	24
7.3	Trie-bits	24
7.4	Template	25
7.5	Rand	25
7.6	Safe-Map	25
8	Strings	25
8.1	Trie	25
8.2	KMP	26
8.3	Suffix-array	26
8.4	LCS	26
8.5	LCSubseq	26
8.6	Pal-int	26
8.7	Z-Func	27
8.8	Hash	27
8.9	Manacher	27
8.10	Suffix-array-radix	27

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     forn(i, n) {
32         int t = val(i, true), f = val(i, false);
33         if(cor[t] == cor[f]) {
34             return false;
35         }
36         else {
37             if(cor[t] > cor[f])
38                 r.pb(1);
39             else
40                 r.pb(0);
41         }
42     }
43     swap(r, res);
44     return true;
45 }

```

2.4 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 }

```

2.5 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(
25                i, g[k][0]);
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                );
39            return st.query(pos[h[a]], pos[a]) +
40                query_path(pai[h[a]], b);
41        }
42        void update_path(int a, int b, int x){
43            if(pos[a]<pos[b]) swap(a, b);
44
45            if(h[a]==h[b]) return (void)st.update(pos[b],
46                pos[a], x);
47            st.update(pos[h[a]], pos[a], x); update_path(
48                pai[h[a]], b, x);
49        }
50        ll query_subtree(int a){
51            return st.query(pos[a], pos[a]+sz[a]-1);
52        }
53        void update_subtree(int a, int x){
54            st.update(pos[a], pos[a]+sz[a]-1, x);
55        }
56        int lca(int a, int b){
57            if(pos[a]<pos[b]) swap(a, b);
58            return (h[a]==h[b] ? b:lca(pai[h[a]], b));
59        }
60    };

```

2.6 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
6     ; // trocar por queue para 0(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 }

```

2.7 Dijkstra

```

1  // Dijkstra - Shortest Path
2
3  vector<vii> grafo(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]: grafo[u]){
17             if(d[v]>d[u]+w){
18                 d[v]=d[u]+w;
19                 fila.push({d[v], v});
20             }
21         }
22     }
23 }

```

2.8 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)-
8      __builtin_clz(x); }
9      rmq(const vector<T>& v_) : v(v_), n(v.size()),
10     mask(n), t(n) {
11         for (int i = 0, at = 0; i < n; mask[i++] = at
12         |= 1) {
13             at = (at<<1)&((1<<b)-1);
14             while (at and op(i, i-msb(at&-at)) == i)
15                 at ^= at&-at;
16         }
17         for (int i = 0; i < n/b; i++) t[i] = b*i+b-1-
18         msb(mask[b*i+b-1]);
19         for (int j = 1; (1<<j) <= n/b; j++) for (int
20         i = 0; i+(1<<j) <= n/b; i++)
21             t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j
22             -1)+i+(1<<(j-1))]);
23     }
24     int small(int r, int sz = b) { return r-msb(mask[
25     r]&((1<<sz)-1)); }
26 }

```

```

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
26          -(1<<j)+1]));
27      }
28      return ans;
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  }

```

2.9 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     }

```

```

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(a==b) return 0;
36     if(pos[a]<pos[b]) swap(a, b);
37
38     if(h[a]==h[b]) return st.query(pos[b]+1, pos[
a]);
39     return st.query(pos[h[a]], pos[a]) +
query_path(pai[h[a]], b);
40 }
41 void update_path(int a, int b, int x){
42     if(a==b) return;
43     if(pos[a]<pos[b]) swap(a, b);
44
45     if(h[a]==h[b]) return (void)st.update(pos[b
+1, pos[a], x);
46     st.update(pos[h[a]], pos[a], x); update_path(
pai[h[a]], b, x);
47 }
48 ll query_subtree(int a){
49     if(sz[a]==1) return 0;
50     return st.query(pos[a]+1, pos[a]+sz[a]-1);
51 }
52 void update_subtree(int a, int x){
53     if(sz[a]==1) return;
54     st.update(pos[a]+1, pos[a]+sz[a]-1, x);
55 }
56 int lca(int a, int b){
57     if(pos[a] < pos[b]) swap(a, b);
58     return (h[a]==h[b] ? b:lca(pai[h[a]], b));
59 }
60 };

```

2.10 Floyd-Warshall

```

1 // Floyd Warshall
2
3 int dist[MAX][MAX];
4
5 void Floydwarshall()
6 {
7     for(int k = 1;k <= n;k++)
8         for(int i = 1;i <= n;i++)
9             for(int j = 1;j <= n;j++)
10                 dist[i][j] = min(dist[i][j], dist[i][
k] + dist[k][j]);
11 }

```

2.11 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
da MST

```

```

20         dsu.join(e.u, e.v);
21     }
22 }

```

2.12 DFS

```

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

```

2.13 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 }

```

2.14 Kosaraju

```

1 int n;
2 vi g[MAX], gi[MAX]; // grafo invertido
3 int vis[MAX], comp[MAX]; // componente conexo de cada
vertice
4 stack<int> S;
5
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 }

```

2.15 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 }

```

2.16 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.
cap)
                continue; // v.cap - v.flow
< lim
            ll tmp = run(v.to, sink, min(minE, v.cap-v
.flow));
            v.flow += tmp, rev.flow -= tmp;
            ans += tmp, minE -= tmp;
            if(minE == 0) break;
        }
        return ans;
    }
30 }
31 bool bfs(int source, int sink) {
32     qt = 0;
33     qu[qt++] = source;
34     lvl[source] = 1;
35     vis[source] = ++pass;
36     for(int i = 0; i < qt; i++) {
37         int u = qu[i];
38         px[u] = 0;
39         if(u == sink) return true;
40         for(auto& ed : g[u]) {
41             auto v = edge[ed];

```

```

42             if(v.flow >= v.cap || vis[v.to] ==
pass)
43                 continue; // v.cap - v.flow < lim
44             vis[v.to] = pass;
45             lvl[v.to] = lvl[u]+1;
46             qu[qt++] = v.to;
47         }
48     }
49     return false;
50 }
51 ll flow(int source, int sink) {
52     reset_flow();
53     ll ans = 0;
54     //for(lim = (1LL << 62); lim >= 1; lim /= 2)
55     while(bfs(source, sink))
56         ans += run(source, sink, LLINF);
57     return ans;
58 }
59 void addEdge(int u, int v, ll c, ll rc) {
60     Edge e = {u, v, 0, c};
61     edge.pb(e);
62     g[u].push_back(ne++);
63
64     e = {v, u, 0, rc};
65     edge.pb(e);
66     g[v].push_back(ne++);
67 }
68 void reset_flow() {
69     for(int i = 0; i < ne; i++)
70         edge[i].flow = 0;
71     memset(lvl, 0, sizeof(lvl));
72     memset(vis, 0, sizeof(vis));
73     memset(qu, 0, sizeof(qu));
74     memset(px, 0, sizeof(px));
75     qt = 0; pass = 0;
76 }
77 };

```

2.17 Prim

```

1 // Prim Algorithm
2 #define MAXN 10100
3 #define INFINTO 999999999
4
5 int n, m;
6 int distancia[MAXN];
7 int processado[MAXN];
8 vector<pii> vizinhos[MAXN];
9
10 int Prim()
11 {
12     for(int i = 2; i <= n; i++) distancia[i] = INFINTO
;
13     distancia[1] = 0;
14
15     priority_queue< pii, vector<pii>, greater<pii> >
fila;
16     fila.push( pii(distancia[1], 1) );
17
18     while(1){
19         int davez = -1;
20
21         while(!fila.empty()){
22             int atual = fila.top().second;
23             fila.pop();
24
25             if(!processado[atual]){
26                 davez = atual;
27                 break;
28             }
29         }
30
31         if(davez == -1)

```

```

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

```

3 Geometria

3.1 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 }
47 }

```

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]) >
area(h[m - 1], h[0], h[k]))
10         ++k;
11     double res = 0;
12     for (int i = 0, j = k; i <= k && j < m; i++) {
13         res = max(res, dist(h[i], h[j]));
14         while (j < m && area(h[i], h[(i + 1) % m], h
[(j + 1) % m]) > area(h[i], h[(i + 1) % m], h[j]))
15             res = max(res, dist(h[i], h[(j + 1) % m]));
16     }
17 }

```



```

16         ++j;
17     }
18 }
19 return res;
20 }

```

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)
26            }
27        }
28
29        s.insert(point(vet[i].y, vet[i].x));
30    }
31    return best_dist;
32 }

```

3.5 Intersect-polygon

```

1 bool intersect(vector<point> A, vector<point> B) //
2 Ordered ccw
3 {
4     for(auto a: A)
5         if(inside(B, a))
6             return true;
7     for(auto b: B)
8         if(inside(A, b))
9             return true;
10
11     if(inside(B, center(A)))
12         return true;
13
14     return false;
15 }

```

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();
20    L.insert(L.end(), U.begin(), U.end()-1);
21    return L;
22 }

```

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
        prisma (va^vb eh nulo),
        //ele esta dentro do prisma (poderia
        desconsiderar pois distancia
        //vai ser a msm da distancia do ponto ao
        segmento)
22         if(!nulo(va^vb) and (v[(i+2)%3]*ve>d) ^ (p*ve
        >d)) return LLINF;
23     }
24
25     //se ponto for coplanar ao triangulo (e dentro do
26     triangulo)
27     //vai retornar zero corretamente
28     return fabs(misto(p-v[0],v1,v2)/norm(n));
29 }
30
31
32 ld dist_pt_seg(point p, vp li){
33     return norm((li[1]-li[0])^(p-li[0]))/norm(li[1]-
34     li[0]);
35 }
36
37 ld dist_line(vp l1, vp l2){
38     point n = (l1[1]-l1[0])^(l2[1]-l2[0]);
39     if(nulo(n)) //retas paralelas - dist ponto a reta
40         return dist_pt_seg(l2[0],l1);
41
42     point o1o2 = l2[0]-l1[0];
43     return fabs((o1o2*n)/norm(n));
44 }
45 // retas paralelas e intersecao nao nula
46 ld dist_seg(vp l1, vp l2){
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     }
57
58     //verificando distancia de ponto extremo com
59     ponto interno dos segs
60     for(int t=0;t<2;t++){
61         for(int i=0;i<2;i++){
62             bool c=true;
63             for(int k=0;k<2;k++){
64                 point va = l1[i]-l2[k];
65                 point vb = l2[!k]-l2[k];
66                 ld ang = atan2(norm((vb^va)), vb*va);
67                 if(ang>PI/2) c = false;
68             }
69             if(c)
70                 ans = min(ans, dist_pt_seg(l1[i],l2));
71             swap(l1,l2);
72         }
73     }
74
75     //ponto interno com ponto interno dos segmentos
76     point v1 = l1[1]-l1[0], v2 = l2[1]-l2[0];
77     point n = v1^v2;
78     if(!nulo(n)){
79         bool ok = true;
80         for(int t=0;t<2;t++){
81             point n2 = v2^v1;
82             point o1o2 = l2[0]-l1[0];
83             ld escalar = (o1o2*n2)/(v1*n2);
84             if(escalar<0 or escalar>1) ok = false;
85             swap(l1,l2);
86             swap(v1,v2);
87         }
88         if(ok) ans = min(ans, dist_line(l1,l2));
89     }
90
91     return ans;
92 }
93
94 ld ver(vector<vp> &vet){
95     ld ans = LLINF;
96     // vertice - face
97     for(int k=0;k<2;k++){
98         for(int pt=0;pt<4;pt++){
99             for(int i=0;i<4;i++){
100                 vp v;
101                 for(int j=0;j<4;j++){
102                     if(i!=j) v.pb(vet[!k][j]);
103                 }
104                 ans = min(ans, dist_pt_face(vet[k][pt
105                 ], v));
106             }
107         }
108     }
109
110     // edge - edge
111     for(int i1=0;i1<4;i1++){
112         for(int j1=0;j1<i1;j1++){
113             for(int i2=0;i2<4;i2++){
114                 for(int j2=0;j2<i2;j2++){
115                     ans = min(ans, dist_seg({vet[0][
116                     i1], vet[0][j1]},
117                     {vet[1][
118                     i2], vet[1][j2]}));
119                 }
120             }
121         }
122     }
123
124     return ans;
125 }

```

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
18     point operator-(const point &o) const{
19         return {x-o.x, y-o.y, z-o.z};
20     }
21 }

```

```

19 point operator*(cod t) const{
20     return {x*t, y*t, z*t};
21 }
22 point operator/(cod t) const{
23     return {x/t, y/t, z/t};
24 }
25 bool operator==(const point &o) const{
26     return eq(x, o.x) and eq(y, o.y) and eq(z, o.
27 z);
28 }
29 cod operator*(const point &o) const{ // dot
30     return x*o.x + y*o.y + z*o.z;
31 }
32 point operator^(const point &o) const{ // cross
33     return point(y*o.z - z*o.y,
34                 z*o.x - x*o.z,
35                 x*o.y - y*o.x);
36 };
37
38 ld dist(point a, point b){
39     return sqrt((a-b)*(a-b));
40 }
41 bool nulo(point a){
42     return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0))
43 ;
44 }
45 ld norm(point a){ // Modulo
46     return sqrt(a*a);
47 }
48 ld proj(point a, point b){ // a sobre b
49     return (a*b)/norm(b);
50 }
51 ld angle(point a, point b){ // em radianos
52     return acos((a*b) / norm(a) / norm(b));
53 }
54
55 cod triple(point a, point b, point c){
56     return dot(a, b^c); // Area do paralelepipedo
57 }
58
59
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 p1, 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 }

```

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     }
7     for(int k = 0; k < (int)b.size(); k++){if(b[k]<b[j]
8 ]){
9         j = k;
10    }
11    vp c;
12    c.reserve(a.size() + b.size());
13    for(int k = 0; k < int(a.size()+b.size()); k++){
14        point pt{a[i] + b[j]};
15        if((int)c.size() >= 2 and !ccw(c[c.size()-2],
16 c.back(), pt))
17            c.pop_back();
18        c.pb(pt);
19        int q = i+1, w = j+1;
20        if(q == int(a.size())) q = 0;
21        if(w == int(b.size())) w = 0;
22        if(ccw(c.back(), a[i]+b[w], a[q]+b[j]) < 0) i
23 = q;
24        else j = w;
25    }
26    if(!ccw(c[0], c[(int)c.size()-1], c[(int)c.size()
27 -2]))
28        c.pop_back();
29    if(!ccw(c.back(), c[0], c[1])){
30        c[0]=c.back();
31        c.pop_back();
32    }
33    c.shrink_to_fit();
34
35    return c;
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;
10 }

```

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);
21            while(sum(p4, 1)!=i and areaT(vet[p4],
22                vet[i], vet[j]) < areaT(vet[sum(p4, 1)], vet[i],
23                vet[j]))
24                p4 = sum(p4, 1);
25
26            ans = max(ans, area(vet[i], vet[p2], vet[
27                j], vet[p4]));
28        }
29    }
30
31    return ans;
32 }

```

3.17 2D

```

1 #define PI acos(-1)
2 #define vp vector<point>
3
4 // typedef int cod;
5 // bool eq(cod a, cod b){ return (a==b); }
6 typedef ld cod;
7 bool eq(cod a, cod b){ return fabs1(a - b) <= EPS; }
8
9 struct point{
10     cod x, y;
11     int id;
12     point(cod x=0, cod y=0): x(x), y(y){}
13
14
15     point operator+(const point &o) const{
16         return {x+o.x, y+o.y};
17     }
18     point operator-(const point &o) const{
19         return {x-o.x, y-o.y};
20     }
21     point operator*(cod t) const{
22         return {x*t, y*t};
23     }
24     point operator/(cod t) const{
25         return {x/t, y/t};
26     }
27     cod operator*(const point &o) const{ // dot
28         return x * o.x + y * o.y;
29     }
30     cod operator^(const point &o) const{ // cross
31         return x * o.y - y * o.x;
32     }
33     bool operator<(const point &o) const{
34         if(!eq(x, o.x)) return x < o.x;
35         return y < o.y;

```

```

36     }
37     bool operator==(const point &o) const{
38         return eq(x, o.x) and eq(y, o.y);
39     }
40 }
41 };
42
43 ld norm(point a){ // Modulo
44     return sqrt(a*a);
45 }
46 bool nulo(point a){
47     return (eq(a.x, 0) and eq(a.y, 0));
48 }
49 ld proj(point a, point b){ // a sobre b
50     return a*b/norm(b);
51 }
52 ld angle(point a, point b){ // em radianos
53     ld ang = a*b / norm(a) / norm(b);
54     return acos(max(min(ang, (ld)1), (ld)-1));
55 }
56 ld angle_vec(point v){
57     // return 180/PI*atan2(v.x, v.y);
58     return atan2(v.x, v.y);
59 }
60 ld order_angle(point a, point b){ // from a to b ccw
61     // (a in front of b)
62     ld aux = angle(a,b)*180/PI;
63     return ((a^b)<=0 ? aux:360-aux);
64 }
65 bool angle_less(point a1, point b1, point a2, point
66     b2){ // ang(a1,b1) <= ang(a2,b2)
67     point p1((a1*b1), abs((a1^b1)));
68     point p2((a2*b2), abs((a2^b2)));
69     return (p1^p2) <= 0;
70 }
71 int ccw(point a, point b, point e){ // -1=dir; 0=
72     collinear; 1=esq;
73     cod tmp = (b-a)^(e-a); // from a to b
74     return (tmp > EPS) - (tmp < -EPS);
75 }
76 point rotccw(point p, ld a){
77     // a = PI*a/180; // graus
78     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
79         +p.x*sin(a)));
80 }
81 bool collinear(point a, point b, point c){
82     return eq((a-c)^(b-c), 0);
83 }
84 point rot90cw(point a) { return point(a.y, -a.x); };
85 point rot90ccw(point a) { return point(-a.y, a.x); };
86
87 ld area(vp &p){ // (points sorted)
88     ld ret = 0;
89     for(int i=2;i<(int)p.size();i++)
90         ret += (p[i]-p[0])^(p[i-1]-p[0]);
91     return fabs1(ret/2);
92 }
93 ld areaT(point &a, point &b, point &c){
94     return fabs1((b-a)^(c-a))/2.0;
95 }
96
97 point center(vp &A){
98     point cA = point();
99     int len = A.size();
100    for(int i=0;i<len;i++)
101        cA=cA+A[i];
102    return cA/len;
103 }
104 point forca_mod(point p, ld m){
105     ld cm = norm(p);

```

```

105     if(cm<EPS) return point();
106     return point(p.x*m/cm,p.y*m/cm);
107 }
108
109 ///////////////
110 // Line //
111 ///////////////
112
113
114 struct line{
115     point p1, p2;
116     cod a, b, c; // ax+by+c = 0;
117     line(point p1=0, point p2=0): p1(p1), p2(p2){
118         a = p1.y-p2.y;
119         b = p2.x-p1.x;
120         c = -(a*p1.x + b*p1.y);
121     }
122     line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
123     {}
124
125     cod eval(point p){
126         return a*p.x+b*p.y+c;
127     }
128     bool inside(point p){
129         return eq(eval(p), 0);
130     }
131     point normal(){
132         return point(a, b);
133     }
134
135     bool inside_seg(point p){
136         return (inside(p) and
137             min(p1.x, p2.x)<=p.x and p.x<=max(p1
138             x, p2.x) and
139             min(p1.y, p2.y)<=p.y and p.y<=max(p1
140             y, p2.y));
141     }
142
143     point inter(line &l1, line &l2){
144         ld det = l1.a*l2.b - l1.b*l2.a;
145         if(det==0) return point(INF, INF);
146         ld x = (l1.b*l2.c - l1.c*l2.b)/det;
147         ld y = (l1.c*l2.a - l1.a*l2.c)/det;
148         return point(x, y);
149     }
150
151     point inter_seg(line &l1, line &l2){
152         point ans = inter(l1, l2);
153         if(ans.x==INF or !l1.inside_seg(ans) or !l2.
154         inside_seg(ans))
155             return point(INF, INF);
156         return ans;
157     }
158
159     cod dseg(point p, point a, point b){ // point - seg
160         if(((p-a)*(b-a)) < EPS) return norm(p-a);
161         if(((p-b)*(a-b)) < EPS) return norm(p-b);
162         return fabs((p-a)^(b-a))/norm(b-a);
163     }
164
165     cod dline(point p, line l){ // point - line
166         return fabs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
167     }
168
169     line mediatrix(point a, point b){
170         point d = (b-a)*2;
171         return line(d.x, d.y, a*a - b*b);
172     }
173
174     line perpendicular(line l, point p){ // passes
175         through p

```

```

173         return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
174     }
175
176
177     ///////////////
178     // Circle //
179     ///////////////
180
181     struct circle{
182         point c; cod r;
183         circle() : c(0, 0), r(0){}
184         circle(const point o) : c(o), r(0){}
185         circle(const point a, const point b){
186             c = (a+b)/2;
187             r = norm(a-c);
188         }
189         circle(const point a, const point b, const point
190         cc){
191             c = inter(mediatrix(a, b), mediatrix(b, cc));
192             r = norm(a-c);
193         }
194         bool inside(const point &a) const{
195             return norm(a - c) <= r;
196         }
197         pair<point, point> getTangentPoint(point p) {
198             ld d1 = norm(p-c), theta = asin(r/d1);
199             point p1 = rotccw(c-p,-theta);
200             point p2 = rotccw(c-p,theta);
201             p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
202             p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
203             return {p1,p2};
204         }
205
206         // minimum circle cover O(n) amortizado
207         circle min_circle_cover(vector<point> v){
208             random_shuffle(v.begin(), v.end());
209             circle ans;
210             int n = v.size();
211             for(int i=0;i<n;i++) if(!ans.inside(v[i])){
212                 ans = circle(v[i]);
213                 for(int j=0;j<i;j++) if(!ans.inside(v[j])){
214                     ans = circle(v[i], v[j]);
215                     for(int k=0;k<j;k++) if(!ans.inside(v[k]))
216                         ans = circle(v[i], v[j], v[k]);
217                 }
218             }
219             return ans;
220         }
221     }
222
223     circle incircle( point p1, point p2, point p3 ){
224         ld m1=norm(p2-p3);
225         ld m2=norm(p1-p3);
226         ld m3=norm(p1-p2);
227         point c = (p1*m1+p2*m2+p3*m3)*(1/(m1+m2+m3));
228         ld s = 0.5*(m1+m2+m3);
229         ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3))/s;
230         return circle(c, r);
231     }
232
233     circle circumcircle(point a, point b, point c) {
234         circle ans;
235         point u = point((b-a).y, -(b-a).x);
236         point v = point((c-a).y, -(c-a).x);
237         point n = (c-b)*0.5;
238         ld t = (u^v)/(v^u);
239         ans.c = ((a+c)*0.5) + (v*t);
240         ans.r = norm(ans.c-a);
241         return ans;
242     }
243 }

```

4 ED

4.1 Trie

```
1 class Trie {
2 private:
3     struct Node {
4         map<char, Node*> children;
5         int qt = 0;
6         ll size = 0;
7     };
8
9     Node* root;
10
11 void dfs(Node* cur) {
12     ll sz = 1;
13
14     for(auto prox : cur->children) {
15         dfs(prox.second);
16         sz += (prox.second->size;
17     }
18
19     cur->size = sz;
20 }
21
22 void del(Node* cur, int dep, string &s) {
23     if(dep >= 32)
24         return;
25
26     Node* prox = cur->children[s[dep]];
27     prox->qt--;
28     del(prox, dep+1, s);
29
30     if(prox->qt == 0)
31         cur->children.erase(s[dep]);
32 }
33
34 public:
35     Trie() {
36         root = new Node();
37         root->qt = 1;
38     }
39
40 void add(string s) {
41     Node* cur = root;
42
43     for(auto c : s) {
44         if(cur->children.count(c) == 0) {
45             cur->children[c] = new Node();
46         }
47         cur->children[c]->qt++;
48         cur = cur->children[c];
49     }
50 }
51
52 void del(string &s) {
53     Node* cur = root;
54     del(cur, 0, s);
55 }
56
57 void size() {
58     this->dfs(root);
59 }
60 };
```

4.2 Prefixsum2D

```
1 ll find_sum(vector<vi> &mat, int x1, int y1, int x2,
2 int y2){
3     // superior-esq(x1,y1) (x2,y2)inferior-dir
4     return mat[x2][y2]-mat[x2][y1-1]-mat[x1-1][y2]+
5     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<(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+1, 0);
8 }
9
10 int sum(int idx) {
11     int ret = 0;
12     for (; 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 < 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    ll pop() {
11        if(out.empty()) {
12            while(!in.empty()) {
13                ll val = in.top().ff;
14                in.pop();
15                ll minimum = out.empty() ? val : min(
16                val, out.top().ss);
17                out.push({val, minimum});
18            }
19            ll res = out.top().ff;
20            out.pop();
21            return res;
22        }
23
24        ll minn() {
25            ll minimum = LLINF;
26            if(in.empty() || out.empty())
27                minimum = in.empty() ? (ll)out.top().ss :
28                (ll)in.top().ss;
29            else
30                minimum = min((ll)in.top().ss, (ll)out.
31                top().ss);
32            return minimum;
33        }
34
35        ll size() {
36            return in.size() + out.size();
37        }
38    };

```

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, O(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
lx, int rx) {
39        if(r <= lx or l >= rx) return 0;
40        if(l <= lx && rx <= r) {
41            auto it = lower_bound(all(v[x]), k);
42            return (v[x].end() - it);
43        }
44        int m = (lx + rx)/2;

```

```

45        ll s1 = greaterEqual(l, r, k, 2*x + 1, lx, m);
46        ll s2 = greaterEqual(l, r, k, 2*x + 2, m, rx);
47
48        return s1 + s2;
49    }
50
51    ll greaterEqual(int l, int r, int k) {
52        return greaterEqual(l, r+1, k, 0, 0, size);
53    }
54 };

```

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 <<
(j-1))][j-1]);
23             }
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
4  struct Query {
5      int l, r, idx;
6      Query(int l, int r, int idx) {
7          this->l = l;
8          this->r = r;
9          this->idx = idx;
10     }
11     bool operator<(Query other) const {
12         if(l/BLK != other.l/BLK)

```

```

12         return l/BLK < other.l/BLK;
13         return (l/BLK & 1) ? r < other.r : r > other.
14     };
15 };
16
17 inline void add() {}
18 inline void remove() {} // implementar operacoes de
19     acordo com o problema
20
21 vector<int> mo(vector<Query>& queries) {
22     vector<int> res(queries.size());
23     sort(queries.begin(), queries.end());
24     resposta = 0;
25
26     int l = 0, r = -1;
27     for(Query q : queries) {
28         while(l > q.l) {
29             l--;
30             add(l);
31         }
32         while(r < q.r) {
33             r++;
34             add(r);
35         }
36         while(l < q.l) {
37             remove(l);
38             l++;
39         }
40         while(r > q.r) {
41             remove(r);
42             r--;
43         }
44         res[q.idx] = resposta; // adicionar resposta
45         de acordo com o problema
46     }
47     return res; // ordenar o vetor pelo indice e
48     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      }
15      return dp[n][m];

```

5.2 Dp-digitos

```

1  // dp de quantidade de numeros <= r com ate qt
2      digitos diferentes de 0
3  ll dp(int idx, string& r, bool menor, int qt, vector<
4      vector<vi>>& tab) {
5      if(qt > 3) return 0;
6      if(idx >= r.size()) {
7          return 1;
8      }

```

```

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     return dp[w];
14 }

```

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
4 // numbers
5 int largestKSubmatrix(){
6     int dp[n][m];
7     memset(dp, 0, sizeof(dp));
8
9     int result = 0;
10    for(int i = 0 ; i < n ; i++){
11        for(int j = 0 ; j < m ; j++){
12            if(!i or !j)
13                dp[i][j] = 1;
14            else if(a[i][j] == a[i-1][j] and
15                    a[i][j] == a[i][j-1] and
16                    a[i][j] == a[i-1][j-1])
17                dp[i][j] = min(min(dp[i-1][j], dp[i][j-1]),
18                                dp[i-1][j-1]) + 1;
19            else dp[i][j] = 1;
20
21            result = max(result, dp[i][j]);
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
13 // -180208
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-1];
24        }
25    }
26    return part[sum / 2][n];
27 }

```

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;

```

```

27 }
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        if(n!=1) div.pb(n);
12    }
13    return div;

```

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; i<n; i++){
23         int bit = n>>1;
24         for(; j&bit; bit>>=1)
25             j^=bit;
26         if(i<j)
27             swap(a[i], a[j]);
28     }
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     }
62     fft(fa, true);
63     vl result(n);
64     for(int i=0;i<n;i++){
65         result[i] = round(fa[i].a);
66     }
67     while(result.back()==0) result.pop_back();
68     return result;
69 }

```

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        }
12        y += yi;
13    }
14    return y;
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        }
26        x += xi;
27    }
28    return x;
29 }

```

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
24     // para n <= 2^64
25     // funciona para n <= 3*10^24 com os primos ate
26     // 41
27     for (int i : {2, 325, 9375, 28178, 450775,
28     9780504, 1795265022}) {
29         if (i >= n) break;
30         ll x = expo(mul(i, i, n), d, n);
31         if (x == 1 or x == n - 1) continue;
32
33         bool deu = 1;
34         for (int j = 0; j < r - 1; j++) {
35             x = mul(x, x, n);
36             if (x == n - 1) {
37                 deu = 0;
38                 break;
39             }
40         }
41         if (deu) return 0;
42     }
43     return 1;
44 }

```

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 || 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     return 1;
49 }
50
51 ll rho(ll n) {
52     if (n == 1 || prime(n)) return n;
53     if (n % 2 == 0) return 2;
54
55     while (1) {
56         ll x = 2, y = 2, ciclo = 2, i = 0, d = 1;
57         ll c = uniform(1, n-1);
58
59         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 || 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 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;

```

6.15 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         return res;
12     }
13 }

6.16 Simpson's-formula

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

```

6.17 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     rev.resize(1 << nbase);
23     for(int i = 0; i < (1 << nbase); i++){
24         rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (
25         nbase - 1));
26     }
27     roots.resize(1 << nbase);
28     while(base < nbase){
29         ld angle = 2*PI / (1 << (base + 1));
30         for(int i = 1 << (base - 1); i < (1 << base);
31         i++){
32             roots[i << 1] = roots[i];
33             ld angle_i = angle * (2 * i + 1 - (1 <<
34             base));
35             roots[(i << 1) + 1] = num(cos(angle_i),
36             sin(angle_i));
37         }
38         base = nbase;
39     }
40 }

```

```

32     }
33     base++;
34 }
35 }
36
37 void fft(vector<num> &a, int n = -1){
38     if(n == -1)
39         n = a.size();
40
41     assert((n & (n-1)) == 0);
42     int zeros = __builtin_ctz(n);
43     ensure_base(zeros);
44     int shift = base - zeros;
45     for(int i = 0; i < n; i++){
46         if(i < (rev[i] >> shift))
47             swap(a[i], a[rev[i] >> shift]);
48     }
49     for(int k = 1; k < n; k <= 1)
50         for(int i = 0; i < n; i += 2 * k)
51             for(int j = 0; j < k; j++){
52                 num z = a[i+j+k] * roots[j+k];
53                 a[i+j+k] = a[i+j] - z;
54                 a[i+j] = a[i+j] + z;
55             }
56     }
57
58 vector<num> fa, fb;
59 vi multiply(vi &a, vi &b){
60     int need = a.size() + b.size() - 1;
61     int nbase = 0;
62     while((1 << nbase) < need) nbase++;
63     ensure_base(nbase);
64     int sz = 1 << nbase;
65     if(sz > (int) fa.size())
66         fa.resize(sz);
67
68     for(int i = 0; i < sz; i++){
69         int x = (i < (int) a.size() ? a[i] : 0);
70         int y = (i < (int) b.size() ? b[i] : 0);
71         fa[i] = num(x, y);
72     }
73     fft(fa, sz);
74     num r(0, -0.25 / sz);
75     for(int i = 0; i <= (sz >> 1); i++){
76         int j = (sz - i) & (sz - 1);
77         num z = (fa[j] * fa[j] - conj(fa[i] * fa[i]))
78         * r;
79         if(i != j) {
80             fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[
81             j])) * r;
82         }
83         fa[i] = z;
84     }
85     fft(fa, sz);
86     vi res(need);
87     for(int i = 0; i < need; i++){
88         res[i] = fa[i].x + 0.5;
89     }
90     return res;
91 }
92
93 vi multiply_mod(vi &a, vi &b, int m, int eq = 0){
94     int need = a.size() + b.size() - 1;
95     int nbase = 0;
96     while((1 << nbase) < need) nbase++;
97     ensure_base(nbase);
98     int sz = 1 << nbase;
99     if(sz > (int) fa.size())
100         fa.resize(sz);
101
102     for(int i=0;i<(int)a.size();i++){
103         int x = (a[i] % m + m) % m;
104     }

```

```

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

```

6.18 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=e>>1LL;
9         b=(b*b)%mod;
10    }
11    return res;
12 }

```

6.19 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));

```

6.20 Recursao-linear

```

1 vector<vl> id(int n) {
2     vector<vl> res(n, vl(n, 0));
3     for(int i = 0; i < n; i++) res[i][i] = 1;
4     return res;
5 }
6
7 vector<vl> mult(vector<vl> a, vector<vl> b, int n) {
8     vector<vl> res(n, vl(n, 0));
9
10    for(int row = 0; row < n; row++) {
11        for(int col = 0; col < n; col++) {
12            ll val = 0;
13            for(int k = 0; k < n; k++) {
14                ll delta = (a[row][k] * b[k][col]) %
MOD;
15                val = (val + delta) % MOD;
16            }
17            res[row][col] = val;
18        }
19    }
20
21    return res;
22 }
23
24 vector<vl> fexp(vector<vl> b, ll e, int n) {
25     if(e == 0) {
26         return id(n);
27     }
28
29     vector<vl> res = fexp(b, e/2, n);
30     res = mult(res, res, n);
31
32     if(e%2)
33         res = mult(res, b, n);
34
35     return res;
36 }
37
38 // k = tamanho da recorrência/matriz, n = n-esimo
termo
39 // f(n) = c1*f(n-1) + c2*f(n-2) + ... + ck*f(n-k)
40 // base -> [f(k-1), f(k-2), ..., f(0)]
41 // coeficientes -> [c1, c2, ..., ck]
42 vl solve(int k, int n, vl base, vl coef) {
43     vector<vl> inicial;
44     inicial.pb(coef);

```

```

45     for(int row = 0; row < k-1; row++) {
46         vl tmp;
47         for(int col = 0; col < k; col++) {
48             if(col == row)
49                 tmp.pb(1);
50             else
51                 tmp.pb(0);
52         }
53         inicial.pb(tmp);
54     }
55
56     vector<vl> matexp = fexp(inicial, max(0, n-k+1),
57 k);
58     vl res(k);
59
60     for(int row = 0; row < k; row++) {
61         ll val = 0;
62         for(int aux = 0; aux < k; aux++) {
63             val += matexp[row][aux]*base[aux];
64         }
65         res[row] = val; // res = (f(n), f(n-1), ...,
66 f(n-k+1))
67     }
68     return res;
69 }

```

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     return true;
34 }
35
36 // mods com raizes primitivas: 2, 4, p^k, 2*p^k, p eh
37 // primo impar, k inteiro --- O(n log^2(n))
38 ll achar_raiz(ll mod, ll phi) {
39     if(mod == 2) return 1;
40     vl fat, elementos;
41     fat = fatorar(phi);
42
43     for(ll i = 2; i <= mod-1; i++) {

```

```

43         if(raiz_prim(i, mod, phi, fat))
44             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     return raizes;
61 }
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 Bitwise

```

1 // Bitwise
2 #pragma GCC target("popcnt")
3 unsigned char a = 5, b = 9; // a = (00000101), b
4 = (00001001)
5
6 AND -           a&b // The result is 00000001
7 (1)
8 OR -           a|b // The result is 00001101
9 (13)
10 XOR -          a^b // The result is 00001100
11 (12)
12 NOT -          ~a // The result is 11111010
13 (250)
14 Left shift -   b<<1 // The result is 00010010
15 (18)
16 Right shift -  b>>1 // The result is 00000100
17 (4)
18
19 // Exchange two int variables
20
21 a^=b;
22 b^=a;
23 a^=b;
24
25
26
27

```



```

18 // Even or Odd
19
20 (x & 1)? printf("Odd"): printf("Even");
21
22 // Turn on the j-th bit
23
24 int S = 34; //(100010)
25 int j = 3;
26
27 S = S | (1<<j);
28
29 // Turn off the j-th bit
30
31 int S = 42; //(101010)
32 int j = 1;
33
34 S &= ~(1<<j)
35
36 S == 40 //(101000)
37
38 // Check the j-th element
39
40 int S = 42; //(101010)
41 int j = 3;
42
43 T = S & (1<<j); // T = 0
44
45 // Least significant bit (lsb)
46
47 int lsb(int x){ return x&-x; }
48
49 // Exchange o j-th element
50
51 S ^= (1<<j)
52
53 // Position of the first bit on
54
55 T = (S & (-S))
56 T -> 4 bit ligado //(1000)
57
58 // Most significant digit of N
59
60 double K = log10(N);
61 K = K - floor(K);
62 int X = pow(10, K);
63
64 // Number of digits in N
65
66 X =floor(log10(N)) + 1;
67
68 // Power of two
69
70 bool isPowerOfTwo(int x){ return x && (!(x&(x
-1)))}; }
71
72 // Turn off the first bit 1
73 m = m & (m-1);
74
75 // Built-in functions
76
77 // Number of bits 1
78 __builtin_popcount()
79 __builtin_popcountll()
80
81 // Number of leading zeros
82 __builtin_clz()
83 __builtin_clzll()
84
85 // Number of trailing zeros
86 __builtin_ctz()
87 __builtin_ctzll()
88
89 // floor(log2(x))

```

```

90
91 int flog2(int x){ return 32-1-__builtin_clz(x
); }
92
93 int flog2ll(ll x){ return 64-1-
__builtin_clzll(x); }

```

7.2 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 }

```

7.3 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 }

```

```

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         return ans;
61     }
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 }

```

7.4 Template

```

1  #include <bits/stdc++.h>
2  #define ff first
3  #define ss second
4  #define ll long long
5  #define ld long double
6  #define pb push_back
7  #define eb emplace_back
8  #define mp make_pair
9  #define mt make_tuple
10 #define pii pair<int, int>
11 #define vi vector<int>
12 #define 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;
23
24 using namespace std;
25
26 const int MAX = 200010;
27 const int MOD = 1000000007;
28 const int INF = 1e8;
29 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
30 const ld EPS = 1e-7;
31
32 // End Template //

```

7.5 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

```

7.6 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    return p;
29 }

```

8.4 LCS

```

1 string LCSubStr(string X, string Y)
2 {
3     int m = X.size();
4     int n = Y.size();
5
6     int result = 0, end;
7     int len[2][n];
8     int currRow = 0;

```

```

9
10    for(int i=0; i<=m; i++){
11        for(int j=0; j<=n; j++){
12            if(i==0 || j==0)
13                len[currRow][j] = 0;
14            else if(X[i-1] == Y[j-1]){
15                len[currRow][j] = len[1-currRow][j-1]
16                + 1;
17                if(len[currRow][j] > result){
18                    result = len[currRow][j];
19                    end = i - 1;
20                }
21            }
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 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 }

```

8.7 Z-Func

```

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

```

8.8 Hash

```

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

```

8.9 Manacher

```

1 // O(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;

```

```

12             r = i + k;
13         }
14     }
15     for(int i = 0, l = 0, r = -1; i < n; i++) {
16         int k = (i > r) ? 0 : min(d2[l + r - i + 1],
17         r - i + 1);
18         while(0 <= i - k - 1 && i + k < n && s[i - k
19         - 1] == s[i + k]) {
20             k++;
21         }
22         d2[i] = k--;
23         if(i + k > r) {
24             l = i - k - 1;
25             r = i + k;
26         }
27     }

```

8.10 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     for(int i = 0; i < (int)rnk.size(); i++) swap(rnk
28     [i].ff, rnk[i].ss);
29     counting_sort(rnk, ind);
30
31     vi suffix_array(string s) {
32         int n = s.size();
33         vector<pii> rnk(n, mp(0, 0));
34         vi ind(n);
35         forn(i, n) {
36             rnk[i].ff = (s[i] == '$') ? 0 : s[i]-'a'+1;
37             // manter '$' como 0
38             ind[i] = i;
39         }
40
41         for(int k = 1; k <= n; k = (k << 1)) {
42             for(int i = 0; i < n; i++) {
43                 if(ind[i]+k >= n) {
44                     rnk[ind[i]].ss = 0;
45                 }
46                 else {
47                     rnk[ind[i]].ss = rnk[ind[i]+k].ff;
48                 }
49             }
50             radix_sort(rnk, ind); // sort(all(rnk), cmp)
51             pra n*log(n), cmp com rnk[i] < rnk[j]

```

```

48     vector<pii> tmp = rnk;
49     tmp[ind[0]] = mp(1, 0); // rnk.fff comecar em
50     1 pois '$' eh o 0
51     for(int i = 1; i < n; i++) {
52         tmp[ind[i]].ff = tmp[ind[i-1]].ff;
53         if(rnk[ind[i]] != rnk[ind[i-1]]) {
54             tmp[ind[i]].ff++;
55         }
56     }
57     swap(rnk, tmp);
58 }
59 return ind;
60 }
61
62 vi lcp_array(string s, vi sarray) {
63     vi inv(s.size());
64     for(int i = 0; i < (int)s.size(); i++) {

```

```

65         inv[sarray[i]] = i;
66     }
67     vi lcp(s.size());
68     int k = 0;
69     for(int i = 0; i < (int)s.size()-1; i++) {
70         int pi = inv[i];
71         if(pi-1 < 0) continue;
72         int j = sarray[pi-1];
73
74         while(s[i+k] == s[j+k]) k++;
75         lcp[pi] = k;
76         k = max(k-1, 0);
77     }
78
79     return vi(lcp.begin()+1, lcp.end()); // LCP(i, j)
80     = min(lcp[i], ..., lcp[j-1])

```