

Competitive programming Notebook

Pablo Arruda Araujo

Sumário

1	Algorithm	2	7	String	14
1.1	bsearch-iterative	2	7.1	AllSubPalindromes	14
1.2	counting-inversions	2	7.2	General	14
1.3	kadane	2	7.3	Manacher	14
1.4	merge-sort	2	7.4	Z-function	14
2	Dp	2	8	Templates	15
2.1	LCS	2	8.1	template	15
2.2	coin-change	2			
2.3	kadane-dp	3			
2.4	knapsack	3			
2.5	unbounded-knapsack	3			
3	Ds	3			
3.1	DSU	3			
3.2	Segtree	4			
3.3	SegtreeLazy	4			
3.4	delta-encoding	5			
3.5	easySegtree	5			
3.6	prefix-sum-array	5			
3.7	sparse-table	5			
3.8	teste	6			
4	Geometry	7			
4.1	2D	7			
4.2	ConvexHull	10			
5	Graph	10			
5.1	BFS	10			
5.2	BellmanFord	10			
5.3	Bipartite	11			
5.4	Bridge	11			
5.5	CycleDetection	11			
5.6	DFS	12			
5.7	Dijkstra	12			
5.8	Kruskal	12			
5.9	MCBM	12			
5.10	Warshall	13			
6	Math	13			
6.1	fast-exponentiation	13			
6.2	floor-log	13			
6.3	matrix-exponentiation	13			

1 Algorithm

1.1 bsearch-iterative

```

1 // Binary search in iterative questions
2 // O(log n)
3 bool query(int mid, int x){
4     cout << mid << endl;
5     cout.flush();
6
7     int ans;
8     cin >> ans;
9     return ans == x;
10 }
11
12 int solve(int x){
13     int l = 1, r = n;
14     int res = -1;
15
16     while(l <= r){
17         int mid = (l+r)/2;
18         if(query(mid, x)){
19             res = mid;
20             l = mid+1;
21         }else{
22             r = m-1;
23         }
24     }
25
26     return res;
27 }

```

1.2 counting-inversions

```

1 // Counting inversions in Array
2 // O(n log n)
3 int merge_sort(vector<int>& v){
4     if(v.size() == 1) return 0;
5
6     vector<int> l, r;
7
8     for(int i = 0; i < v.size()/2; i++)
9         l.push_back(v[i]);
10    for(int i = v.size()/2; i < v.size(); i++)
11        r.push_back(v[i]);
12    int ans = 0;
13    ans += merge_sort(l);
14    ans += merge_sort(r);
15
16    l.push_back(1e9);
17    r.push_back(1e9);
18
19    int inil = 0, inir = 0;
20
21    for(int i = 0; i < v.size(); i++){
22        if(l[inil] <= r[inir]) v[i] = l[inil++];
23        else{
24            v[i] = r[inir++];
25            ans += l.size() - inil - 1;
26        }
27    }
28
29    return ans;
30 }

```

1.3 kadane

```

1 // Maximum possible sum in Array
2 // O(n)
3 int array[MAXN];
4

```

```

5 int kadane(){
6     int sum = 0, best = 0;
7     for(int i = 0; i < n; i++){
8         sum = max(array[i], sum+array[i]);
9         best = max(sum, best);
10    }
11
12    return best;
13 }

```

1.4 merge-sort

```

1 // Merge Sort
2 // O(n log n)
3 void merge_sort(vector<int>& v){
4     if(v.size() == 1) return;
5
6     vector<int> l, r;
7
8     for(int i = 0; i < v.size()/2; i++)
9         l.push_back(v[i]);
10    for(int i = v.size()/2; i < v.size(); i++)
11        r.push_back(v[i]);
12
13    merge_sort(l);
14    merge_sort(r);
15
16    l.push_back(INF);
17    r.push_back(INF);
18
19    int inil = 0, inir = 0;
20
21    for(int i = 0; i < v.size(); i++){
22        if(l[inil] < r[inir]) v[i] = l[inil++];
23        else v[i] = r[inir++];
24    }
25
26    return;
27 }

```

2 Dp

2.1 LCS

```

1 // LCS maior subs comum
2 // ** usar s[1 - n]
3 #define MAXN 1010
4
5 int s1[MAXN], s2[MAXN], tab[MAXN][MAXN];
6
7 int lcs(int a, int b){
8
9     if(a == 0 || b == 0) return tab[a][b] = 0;
10
11     if(tab[a][b] != -1) return tab[a][b];
12
13     if(s1[a] == s2[b]) return lcs(a-1, b-1) + 1;
14
15     return tab[a][b] = max(lcs(a-1, b), lcs(a, b-1));
16 }

```

2.2 coin-change

```

1 // You have n coins {c1, ..., cn}
2 // Find min quantity of coins to sum K
3 // O(n.c)
4 int dp(int acc){ // Recursive version
5     if(acc < 0) return oo;
6     if(acc == 0) return 0;
7
8     if(memo[acc] != -1) return memo[acc];

```

```

9
10     int best = oo;
11
12     for(auto c : coins){
13         best = min(best, dp(acc-c)+1);
14     }
15
16     return memo[acc] = best;
17 }
18
19 int dp(){ // Iterative version
20     memo[0] = 0
21     for(int i = 1; i <= n; i++){
22         memo[i] = oo;
23         for(auto c : coins){
24             if(i-c >= 0)
25                 memo[i] = min(memo[i], memo[i-c]+1);
26         }
27     }
28 }

```

2.3 kadane-dp

```

1 #include <bits/stdc++.h>
2 #define pb push_back
3 #define ll long long int
4 #define sws ios_base::sync_with_stdio(false);cin.tie(
5     NULL);cout.tie(NULL)
6 #define forn(i, n) for(int i = 0; i < (int)n; i++)
7 #define forne(i, a, b) for(int i = a; i <= b; i++)
8 using namespace std;
9 // End Template //
10
11 #define MAXN 10001
12
13 int n;
14 int tab[MAXN];
15 bool foi[MAXN];
16 vector<ll> v;
17
18 ll dp(int i){
19     if(i == 0) return v[0];
20     if(foi[i]) return tab[i];
21     foi[i] = true;
22     return tab[i] = max(v[i], dp(i-1) + v[i]);
23 }
24
25 int main(){
26     sws;
27
28     cin >> n;
29
30     v.assign(n, 0);
31     forn(i, n) cin >> v[i];
32
33     ll ans = 0;
34     forn(i, n) ans = max(ans, dp(i));
35
36     cout << ans << endl;
37
38     return 0;
39 }

```

2.4 knapsack

```

1 // Knapsack problem
2 // O(n.w)
3 int valor[MAXN], peso[MAXN], memo[MAXN];
4
5 ll solve(int i, int w){ // Recursive version
6     if(i <= 0 || w <= 0) return 0;

```

```

7     if(memo[i][w] != -1) return memo[i][w];
8     ll pegar=-1e9;
9
10    if(peso[i] <= w){
11        pegar = solve(i-1,w-peso[i])+valor[i];
12    }
13
14    ll naopegar = solve(i-1,w);
15
16    memo[i][w] = max(pegar,naopegar);
17
18    return memo[i][w];
19 }
20
21 int dp[MAXN][MAXN], valor[MAXN], peso[MAXN];
22 int solve(int n, w){ // Iterative version
23     // n objects | max weight
24     for(int i = 0; i <= n; i++){
25         for(int j=0; j <= w;j++){
26             dp[i][j] = 0;
27
28             for(int i = 0; i <= n; i++){
29                 for(int j = 0; j <= w; j++){
30                     if(i == 0 || j == 0) return dp[i][j];
31                     else if(peso[i-1] <= j)
32                         dp[i][j] = max(dp[i-1][j-peso[i-1]]+
33                             valor[i-1],dp[i-1][j]);
34                     else
35                         dp[i][j] = dp[i-1][j];
36                 }
37             }
38             return dp[n][w];
39 }
40
41 int val[MAX], wt[MAX], dp[MAX]; // Optimization for
42     space
43 int solve(int n, int W){
44     for(int i=0; i < n; i++)
45         for(int j=W; j>=wt[i]; j--)
46             dp[j] = max(dp[j],dp[j-wt[i]]+val[i]);
47     return dp[W];
48 }

```

2.5 unbounded-knapsack

```

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

```

3 Ds

3.1 DSU

```

1 // Disjoint union set
2 // Operation ~ O(1)
3 struct DSU {
4     int n = 0;
5     vector<int> p;

```

```

6     vector<int> sz;
7
8     DSU(int nn){
9         n = nn;
10        sz.assign(n + 5, 1);
11        p.assign(n + 5, 0);
12        iota(p.begin(), p.end(), 0);
13    }
14
15    int find(int x){
16        return p[x] = (p[x] == x ? x : find(p[x]));
17    }
18
19    void join(int a, int b){
20        a = find(a); b = find(b);
21        if(a == b) return;
22        if(sz[a] < sz[b]) swap(a, b);
23        p[b] = a;
24        sz[a] += sz[b];
25    }
26 };
27
28 // Initializing values in main()
29 // DSU(n+1)

```

3.2 Segtree

```

1 // Segtree Sum
2 // O(log n) operations
3
4 // DESCRIPTION:
5 // sti: id do nodo que estamos na segment tree
6 // stl: limite inferior do intervalo que aquele nodo
   representa(inclusivo)
7 // str: limite superior do intervalo que aquele nodo
   representa(inclusivo)
8 // l : limite inferior do intervalo que queremos
   fazer a consulta
9 // r : limite superior do intervalo que queremos
   fazer a consulta
10 // i : indice do vetor que queremos atualizar
11 // amm: novo valor daquele indice no vetor
12 // obs: intervalo incluso [l, r]
13
14 struct SegTree {
15
16     int n;
17     vector<ll> st;
18
19     ll neutro = 0LL;
20
21     SegTree(vector<ll>& arr){
22         n = (int)arr.size();
23         st = vector<ll>(4*n, 0);
24
25         build(0, 0, n-1, arr);
26     }
27
28     SegTree(int nn){
29         n = nn;
30         st = vector<ll>(4*n, 0);
31     }
32
33     ll merge(ll a, ll b){
34         return a + b;
35     }
36
37     void build(int sti, int l, int r, vector<ll>& arr)
38     ){
39         if(l == r){
40             st[sti] = 1LL * arr[l];
41         } else {

```

```

42             int mid = 1 + (r-l)/2;
43             build(sti+sti+1, l, mid, arr);
44             build(sti+sti+2, mid+1, r, arr);
45             st[sti] = merge(st[sti+sti+1], st[sti+sti
46             +2]);
47         }
48     }
49
50     void upd(int sti, int stl, int str, int i, ll amm)
51     ){
52         if(stl == str && stl == i){
53             st[sti] += amm;
54             return;
55         }
56
57         if(stl > i || str < i) return;
58
59         int mid = stl + (str-stl)/2;
60
61         upd(sti+sti+1, stl, mid, i, amm);
62         upd(sti+sti+2, mid+1, str, i, amm);
63
64         st[sti] = merge(st[sti+sti+1], st[sti+sti+2]);
65     };
66
67     ll qq(int sti, int stl, int str, int l, int r){
68         if(str < l || stl > r) return neutro;
69
70         // Completamente incluso
71         if(stl >= l && str <= r){
72             return st[sti];
73         }
74
75         int mid = stl + (str-stl)/2;
76
77         return merge(qq(sti+sti+1, stl, mid, l, r),
78             qq(sti+sti+2, mid+1, str, l, r));
79     }
80
81     void update(int i, ll amm){ upd(0,0,n-1,i,amm); }
82     ll query(int l, int r){ return qq(0,0,n-1,l,r); }
83 };

```

3.3 SegtreeLazy

```

1 // Segtree Sum
2 // O(log n) operations
3
4 // DESCRIPTION:
5 // sti: id do nodo que estamos na segment tree
6 // stl: limite inferior do intervalo que aquele nodo
   representa(inclusivo)
7 // str: limite superior do intervalo que aquele nodo
   representa(inclusivo)
8 // l : limite inferior do intervalo que queremos
   fazer a consulta
9 // r : limite superior do intervalo que queremos
   fazer a consulta
10 // i : indice do vetor que queremos atualizar
11 // amm: novo valor daquele indice no vetor
12 // obs: intervalo incluso [l, r]
13
14 struct SegTree {
15
16     int n;
17     vector<ll> st;
18     vector<ll> lazy;
19
20

```

```

21 ll neutro = 0LL;
22
23 SegTree(vector<ll>& arr){
24     n = (int)arr.size();
25     st = vector<ll>(4*n, 0);
26     lazy = vector<ll>(4*n, 0);
27
28     build(0, 0, n-1, arr);
29 }
30
31 SegTree(int nn){
32     n = nn;
33     st = vector<ll>(4*n, 0);
34     lazy = vector<ll>(4*n, 0);
35 }
36
37 ll merge(ll a, ll b){
38     return a + b;
39 }
40
41 void build(int sti, int l, int r, vector<ll>& arr)
42 ){
43     if(l == r){
44         st[sti] = 1LL * arr[l];
45     } else {
46         int mid = l + (r-l)/2;
47         build(sti+sti+1, l, mid, arr);
48         build(sti+sti+2, mid+1, r, arr);
49         st[sti] = merge(st[sti+sti+1], st[sti+sti
50 +2]);
51     }
52 }
53
54 void propagate(int sti, int l, int r){
55     if(lazy[sti] != 0){
56         st[sti] += lazy[sti] * (r-l+1);
57         if(l != r){
58             lazy[2*sti+1] += lazy[sti];
59             lazy[2*sti+2] += lazy[sti];
60         }
61         lazy[sti] = 0;
62     }
63 }
64
65 void upd(int sti, int stl, int str, int l, int r,
66 ll amm){
67     propagate(sti, stl, str);
68
69     if(stl > r || str < l) return;
70
71     if(stl >= l && str <= r){
72         lazy[sti] += amm;
73         propagate(sti, stl, str);
74         return;
75     }
76
77     int mid = stl + (str-stl)/2;
78
79     upd(sti+sti+1, stl, mid, l, r, amm);
80     upd(sti+sti+2, mid+1, str, l, r, amm);
81
82     st[sti] = merge(st[sti+sti+1], st[sti+sti+2])
83 ;
84 }
85
86 ll qq(int sti, int stl, int str, int l, int r){
87     propagate(sti, stl, str);
88     if(str < l || stl > r) return neutro;
89
90     // Completamente incluso
91     if(stl >= l && str <= r){

```

```

90         return st[sti];
91     }
92
93     int mid = stl + (str-stl)/2;
94
95     return merge(qq(sti+sti+1, stl, mid, l, r),
96 qq(sti+sti+2, mid+1, str, l, r));
97 }
98
99 void update_range(int l, int r, ll amm){ upd(0,0,
100 n-1,l,r,amm); }
101 ll query(int l, int r){ return qq(0,0,n-1,l,r); }
102 };

```

3.4 delta-encoding

```

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

```

3.5 easySegtree

3.6 prefix-sum-array

```

1 // Prefix sum 1D
2 // O(n)
3 int v[MAXN];
4 int psum[MAXN];
5
6 int create_psum(){
7     int acc = 0;
8     for(int i = 0; i < v.size(); i++){
9         acc+=v[i];
10        psum[i] = acc;
11    }
12 }
13
14 int query(int l, int r){
15     return l == 0 ? psum[r] : psum[r]-psum[l-1];
16 }

```

3.7 sparse-table

```

1 // Sparse-Table
2 // O(log n)
3 const int logn = 22; // max log
4
5 int logv[MAX];
6 // Pre comp log values
7 void make_log(){
8     logv[1] = 0;
9     for(int i = 2; i <= MAX; i++){
10         logv[i] = logv[i/2]+1;
11     }
12 }
13 struct Sparse {
14     vector<vector<int> > st;

```

```

15 Sparse(vector<int>& v) {
16     int n = v.size();
17     st.assign(n, vector<int>(logn, 0));
18     // Unitary values st[i][0] = v[i, i+2^0] = v[i]
19     for(int i = 0; i < n; i++){
20         st[i][0] = v[i];
21     }
22     // Constructing Sparse Table in O(log n)
23     for(int k = 1; k < logn; k++){
24         for(int i = 0; i < n; i++){
25             if(i + (1 << k) - 1 >= n)
26                 continue;
27             int prox = i + (1 << (k-1));
28             st[i][k] = min(st[i][k-1], st[prox][k-1]);
29         }
30     }
31 }
32
33 int f(int a, int b){
34     // Can be: min, max, gcd
35     // f must have idempotent property
36     return min(a, b);
37 }
38 // Queries in O(1)
39 int query(int l, int r){
40     int size = r-l+1;
41     int k = logv[size];
42     // cat jump for queries in O(1)
43     int res = f(st[l][k], st[r - ((1 << k) - 1)][k]);
44     return res;
45 }
46
47 };

```

3.8 teste

```

1 // SÃS AC GOSTOSO
2 #include <bits/stdc++.h>
3 #define ff first
4 #define ss second
5 #define pii pair<int, int>
6 #define vi vector<int>
7 #define ll long long int
8 #define ld long double
9 #define pb push_back
10 #define sws ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL)
11 #define forn(i, n) for(int i = 0; i < (int)n; i++)
12 #define forne(i, a, b) for(int i = a; i <= b; i++)
13 #define all(x) x.begin(), x.end()
14 #define teto(a, b) ((a+b-1)/(b))
15 #define dbg(msg, var) cerr << msg << " " << var << endl;
16
17 using namespace std;
18
19 const int INF = 0x3f3f3f3f;
20 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
21 const int MOD = 1000000007;
22 const ld EPS = 1e-8;
23 const ld PI = acos(-1);
24
25 // End Template //
26
27 struct SegTree {
28     int n;
29     vector<ll> st;
30     vector<ll> lazy;

```

```

33 ll neutro = 0LL;
34
35 SegTree(vector<ll>& arr){
36     n = (int)arr.size();
37     st = vector<ll>(4*n, 0);
38     lazy = vector<ll>(4*n, 0);
39
40     build(0, 0, n-1, arr);
41 }
42
43 SegTree(int nn){
44     n = nn;
45     st = vector<ll>(4*n, 0);
46     lazy = vector<ll>(4*n, 0);
47 }
48
49 ll merge(ll a, ll b){
50     return a + b;
51 }
52
53 void build(int sti, int l, int r, vector<ll>& arr)
54 {
55     if(l == r){
56         st[sti] = 1LL * arr[l];
57     } else {
58         int mid = l + (r-l)/2;
59         build(sti+sti+1, l, mid, arr);
60         build(sti+sti+2, mid+1, r, arr);
61         st[sti] = merge(st[sti+sti+1], st[sti+sti+2]);
62     }
63 }
64
65 void propagate(int sti, int l, int r){
66     if(lazy[sti] != 0){
67         st[sti] += lazy[sti] * (r-l+1);
68         if(l != r){
69             lazy[2*sti+1] += lazy[sti];
70             lazy[2*sti+2] += lazy[sti];
71         }
72         lazy[sti] = 0;
73     }
74 }
75
76 void upd(int sti, int stl, int str, int l, int r, ll amm){
77     propagate(sti, stl, str);
78
79     if(stl > r || str < l) return;
80
81     if(stl >= l && str <= r){
82         lazy[sti] += amm;
83         propagate(sti, stl, str);
84         return;
85     }
86
87     int mid = stl + (str-stl)/2;
88
89     upd(sti+sti+1, stl, mid, l, r, amm);
90     upd(sti+sti+2, mid+1, str, l, r, amm);
91
92     st[sti] = merge(st[sti+sti+1], st[sti+sti+2]);
93 }
94
95 ll qq(int sti, int stl, int str, int l, int r){
96     propagate(sti, stl, str);
97     if(str < l || stl > r) return neutro;
98
99     // Completamente incluso

```

```

102         if(stl >= l && str <= r){
103             return st[sti];
104         }
105
106         int mid = stl + (str-stl)/2;
107
108         return merge(qq(sti+sti+1, stl, mid, l, r),
qq(sti+sti+2, mid+1, str, l, r));
109     }
110
111     void update_range(int l, int r, ll amm){ upd(0,0,
n-1,l,r,amm); }
112     ll query(int l, int r){ return qq(0,0,n-1,l,r); }
113 }
114
115 };
116
117 void solve(){
118
119     int n, q;
120     cin >> n >> q;
121
122     vector<ll> pref(n + 1);
123
124     vector<ll> last(n);
125
126     for(int i = 0; i < n; i++){
127         cin >> last[i];
128     }
129
130     for(int i = 0; i < n; i++) {
131         pref[i] = last[i];
132         if(i) pref[i] += pref[i - 1];
133     }
134
135     SegTree st(pref);
136
137     while(q--) {
138
139         int t;
140         cin >> t;
141
142         if( t == 1 ) {
143
144             int i, x;
145             cin >> i >> x;
146
147             ll dif = x - last[i - 1];
148             st.update_range(i - 1, n-1, dif);
149
150             last[i - 1] = x;
151
152         } else {
153
154             int l, r;
155             cin >> l >> r;
156             cout << st.query(l - 1, r - 1) << "\n";
157
158         }
159     }
160 }
161
162 }
163
164 int main(){
165
166     int t = 1;
167     //cin >> t;
168
169     while(t--)
170         solve();
171
172 }

```

4 Geometry

4.1 2D

```

1 // 2D Geometry lib
2 // Good questions: Corner cases? Imprecisions?
3
4 typedef ld T;
5 bool eq(T a, T b){ return fabs(a - b) <= EPS; }
6
7 // typedef int T; // or int
8 // bool eq(T a, T b){ return (a==b); }
9
10 #define sq(x) ((x)*(x))
11 #define rad_to_deg(x) (180/PI)*x
12 #define vp vector<pt>
13
14 const ld DINF = 1e18;
15
16 struct pt{
17     T x, y;
18
19     pt(T x=0, T y=0): x(x), y(y){};
20
21     pt operator+(const pt &o) const{ return {x+o.x, y
+o.y}; }
22     pt operator-(const pt &o) const{ return {x-o.x, y
-o.y}; }
23     pt operator*(T t) const{ return {x*t, y*t};}
24     pt operator/(T t) const{return {x/t, y/t};}
25     T operator*(const pt &o) const{ return x * o.x +
y * o.y; }
26     T operator^(const pt &o) const{ return x * o.y -
y * o.x; }
27
28     bool operator<(const pt &o) const{ if(!eq(x, o.x)
) return x < o.x; return y < o.y; }
29     bool operator==(const pt &o) const{ return eq(x,
o.x) and eq(y, o.y); }
30 };
31
32 //\ PONTO E VETOR /\
33
34 bool nulo(pt p){ return (eq(p.x, 0) && eq(p.y, 0));}
35 // confere se = nulo
36
37 ld dist(pt p, pt q){ return hypot(p.y - q.y, p.x - q.
x); } // distancia
38
39 ld dist2(pt p, pt q){ return sq(p.y - q.y) + sq(p.x -
q.x); } // distancia*distancia
40
41 ld norm(pt p){ return dist(pt(0, 0), p); } // norma
do vetor
42
43 ld sArea(pt p, pt q, pt r) { //
44     return ((q-p)^(r-q))/2;
45 }
46
47 bool col(pt p, pt q, pt r) { // se p, q e r sao colin
.
48     return eq(sArea(p, q, r), 0);
49 }
50
51 ld angle(pt p){ // angle of a vector
52     ld ang = atan2(p.y, p.x);
53     if (ang < 0) ang += 2*PI;
54     return ang;
55 }
56
57 ld angle(pt p, pt q){ // angle between two vectors

```

```

58     ld ang = p*q / norm(p) / norm(q);
59     return acos(max(min(ang, (ld)1), (ld)-1));
60 }
61
62 int ccw(pt a, pt b, pt e){ // -1=dir; 0=col; 1=esq;
63     esq = AE esta a esquerda de AB
64     T tmp = (b-a)^(e-a);
65     return (tmp > EPS) - (tmp < -EPS);
66 }
67 pt rotccw(pt p, ld a){ // rotacionar ccw
68     // a = PI*a/180; // graus
69     return pt((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.x*sin(a)));
70 }
71
72 pt rot90cw(pt p) { return pt(p.y, -p.x); };
73
74 pt rot90ccw(pt p) { return pt(-p.y, p.x); };
75
76 ld proj(pt a, pt b){ // a sobre b
77     return a*b/norm(b);
78 }
79
80 int paral(pt u, pt v) { // se u e v sao paralelos
81     if (!eq(u^v, 0)) return 0;
82     if ((u.x > EPS) == (v.x > EPS) && (u.y > EPS) == (v.y > EPS))
83         return 1;
84     return -1;
85 }
86
87 pt mirror(pt m1, pt m2, pt p){
88     // mirror pt p around segment m1m2
89     pt seg = m2-m1;
90     ld t0 = ((p-m1)*seg) / (seg*seg);
91     pt ort = m1 + seg*t0;
92     pt pm = ort-(p-ort);
93     return pm;
94 }
95
96 pt center(vp &A){ // center of pts
97     pt c = pt();
98     int len = A.size();
99     for(int i=0; i<len; i++)
100         c=c+A[i];
101     return c/len;
102 }
103
104 bool simetric(vector<pt> &a){ // ordered - check
105     simetric pt
106     int n = a.size(); // . . . . ok / . . . . !ok
107     pt c = center(a);
108     if(n%1) return false;
109     for(int i=0; i<n/2; i++)
110         if(!col(a[i], a[i+n/2], c))
111             return false;
112     return true;
113 }
114 //\ LINE /\
115
116 struct line{ // line or line segment
117     T a, b, c;
118     pt p1, p2; // ax + by + c = 0 -> y = ((-a/b)x - (c/b))
119     line(pt p1, pt p2): p1(p1), p2(p2){
120         a = p1.y-p2.y; b = p2.x-p1.x; c = -(a*p1.x + b*p1.y);
121     }
122     line(T a, T b, T c): a(a), b(b), c(c){
123
124         if(b == 0){ p1 = pt(0, -c/a); p2 = pt(0, -c/a); }
125         else{
126             p1 = pt(1, (-c-a*1)/b);
127             p2 = pt(0, -c/b);
128         }
129     }
130
131     T eval(pt p){ // value of {x,y} on line
132         return a*p.x+b*p.y+c;
133     }
134
135     bool insideLine(pt p){ // check if pt is inside line
136         return eq(eval(p), 0);
137     }
138
139     bool insideSeg(pt p){ // check if pt is inside line seg
140         return (insideLine(p) &&
141             min(p1.x, p2.x)<=p.x && p.x<=max(p1.x, p2.x) &&
142             min(p1.y, p2.y)<=p.y && p.y<=max(p1.y, p2.y));
143     }
144
145     pt normal(){ // normal vector
146         return pt(a, b);
147     }
148 };
149
150 vp intersecLine(line l1, line l2){ // pt of two line
151     intersec
152     ld det = l1.a*l2.b - l1.b*l2.a;
153     if(det==0) return {};
154     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
155     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
156     return {pt(x, y)};
157 }
158
159 vp intersecSeg(line l1, line l2){ // intersec of two line seg
160     vp ans = intersecLine(l1, l2);
161     if(ans.empty() || !l1.insideSeg(ans[0]) || !l2.insideSeg(ans[0]))
162         return {};
163     return ans;
164 }
165
166 ld dSeg(pt p, pt a, pt b){ // distance - pt to line seg
167     if(((p-a)*(b-a)) < EPS) return norm(p-a);
168     if(((p-b)*(a-b)) < EPS) return norm(p-b);
169     return abs((p-a)^(b-a))/norm(b-a);
170 }
171
172 ld dLine(pt p, line l){ // pt - line
173     return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
174 }
175
176 bool paralel(line r, line s) { // se r e s sao paralelas
177     return paral(r.p1 - r.p2, s.p1 - s.p2);
178 }
179
180 line perpendicular(line l, pt p){ // passes through p
181     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
182 }
183
184 line bisector(line l){ // bisector of a line segment
185     pt mid = pt((l.p1.x + l.p2.x)/2, (l.p1.y + l.p2.y)/2);
186

```



```

187     return perpendicular(l, mid);
188 }
189
190
191
192 // \ POLIGONO / \
193
194 ld area(vp &p){ // polygon area (pts sorted)
195     ld ret = 0;
196     for(int i=2; i<(int)p.size(); i++){
197         ret += (p[i]-p[0])^(p[i-1]-p[0]);
198     }
199     return abs(ret/2);
200 }
201
202 int isInside(vector<pt>& v, pt p) { // O(n) - pt
    inside polygon
203     int qt = 0; // 0 outside / 1 inside / 2 border
204     for (int i = 0; i < (int)v.size(); i++) {
205         if (p == v[i]) return 2;
206         int j = (i+1)%v.size();
207         if (eq(p.y, v[i].y) && eq(p.y, v[j].y)) {
208             if ((v[i]-p)*(v[j]-p) < EPS) return 2;
209             continue;
210         }
211         bool baixo = v[i].y+EPS < p.y;
212         if (baixo == (v[j].y+EPS < p.y)) continue;
213         auto t = (p-v[i])^(v[j]-v[i]);
214         if (eq(t, 0)) return 2;
215         if (baixo == (t > EPS)) qt += baixo ? 1 : -1;
216     }
217     return qt != 0;
218 }
219
220 bool isIntersec(vector<pt> v1, vector<pt> v2) { // 2
    polygons intersec- O(n*m)
221     int n = v1.size(), m = v2.size();
222     for (int i = 0; i < n; i++) if (isInside(v2, v1[i]))
223         return 1;
224     for (int i = 0; i < n; i++) if (isInside(v1, v2[i]))
225         return 1;
226     for (int i = 0; i < n; i++) for (int j = 0; j < m; j++)
227         if (intersecSeg(line(v1[i], v1[(i+1)%n]),
228             line(v2[j], v2[(j+1)%m])).size() != 0) return 1;
229     return 0;
230 }
231
232 // ld distPol(vector<pt> v1, vector<pt> v2) { //
    distancia de poligonos
233 //     if (isIntersec(v1, v2)) return 0;
234 //
235 //     ld ret = DINF;
236 //     for (int i = 0; i < v1.size(); i++){
237 //         for (int j = 0; j < v2.size(); j++){
238 //             ret = min(ret, dSeg(line(v1[i], v1[(i+1)%v1.size()]),
239 //                 line(v2[j], v2[(j+1)%v2.size()])));
240 //         }
241 //     }
242 //     return ret;
243 // }
244
245 // \ Circle / \
246
247 struct circle{
248     pt c; T r;
249     circle() : c(0, 0), r(0){}
250     circle(const pt o) : c(o), r(0){}
251     circle(const pt a, const pt b){
252         c = (a+b)/2;
253         r = norm(a-c);
254     }
255     bool inside(const pt &a) const{
256         return norm(a - c) <= r;
257     }
258     pair<pt, pt> getTangent(pt p) {
259         ld d1 = norm(p-c), theta = asin(r/d1);
260         pt p1 = rotccw(c-p, -theta);
261         pt p2 = rotccw(c-p, theta);
262         p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
263         p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
264         return {p1,p2};
265     }
266 };
267
268 circle incircle( pt p1, pt p2, pt p3 ){
269     ld m1=norm(p2-p3);
270     ld m2=norm(p1-p3);
271     ld m3=norm(p1-p2);
272     pt c = (p1*m1+p2*m2+p3*m3)*(1/(m1+m2+m3));
273     ld s = 0.5*(m1+m2+m3);
274     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3))/s;
275     return circle(c, r);
276 }
277
278 circle circumCircle(pt a, pt b, pt c) {
279     circle ans;
280     pt u = pt((b-a).y, -(b-a).x);
281     pt v = pt((c-a).y, -(c-a).x);
282     pt n = (c-b)*0.5;
283     ld t = (u^v)/(v^u);
284     ans.c = ((a+c)*0.5) + (v*t);
285     ans.r = norm(ans.c-a);
286     return ans;
287 }
288
289 vp intersecCircleLine(circle C, line L){
290     pt ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.p1)
291         *(ab)/(ab*ab));
292     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
293         / (ab*ab);
294     if (h2 < 0) return {};
295     if (h2 == 0) return {p};
296     pt h = (ab/norm(ab)) * sqrt(h2);
297     return {p - h, p + h};
298 }
299
300 vp intersecCircles(circle C1, circle C2){
301     if(C1.c == C2.c) { assert(C1.r != C2.r); return
302         {}; }
303     pt vec = C2.c - C1.c;
304     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
305     ;
306     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
307         C1.r*C1.r - p*p*d2;
308     if (sum*sum < d2 or dif*dif > d2) return {};
309     pt mid = C1.c + vec*p, per = pt(-vec.y, vec.x) *
310         sqrt(max((ld)0, h2) / d2);
311     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
312     return {mid + per, mid - per};
313 }
314
315 // circle minCircleCover(vector<pt> v){ // O(n) min
    circle that cover all pts
316 //     random_shuffle(v.begin(), v.end());
317 //     circle ans;
318 //     int n = v.size();
319 //     for(int i=0;i<n;i++) if(!ans.inside(v[i])){
320 //         ans = circle(v[i]);
321 //         for(int j=0;j<i;j++) if(!ans.inside(v[j]))
322 //             {

```

```

316 //          ans = circle(v[i], v[j]);          36      return L;
317 //          for(int k=0;k<j;k++) if(!ans.inside(v[ 37 }
           k])){
318 //          ans = circle(v[i], v[j], v[k]);
319 //          }
320 //          }
321 //          }
322 //          return ans;
323 // }
324
325 //\ EXTRA C++ complex library /\
326
327 typedef double T;
328 typedef complex<T> pt;
329 #define x real()
330 #define y imag()
331
332 pt p{3,-4};
333 cout << p.x << " " << p.y << "\n"; // 3 -4
334 cout << p << "\n"; // (3,-4)
335
336 pt p{-3,2};
337 // p.x = 1; // doesnt compile
338 p = {1,2}; // correct
339
340 pt a{3,1}, b{1,-2};
341 a += 2.0*b; // a = (5,-3)
342 cout << a*b << " " << a/-b << "\n"; // (-1,-13)
           (-2.2,-1.4)// typedef int T;
343 // bool eq(T a, T b){ return (a==b); }
344 typedef ld T; // or int
345 bool eq(T a, T b){ return abs(a - b) <= EPS; }

```

4.2 ConvexHull

```

1 // Convex Hull
2 // Algorithm: Monotone Chain
3 // Complexity: O(n) + ordenacao O(nlogn)
4
5 #define vp vector<pt>
6 typedef int T;
7
8 int ccw(pt a, pt b, pt e){ // -1=dir; 0=col; 1=esq;
           esq = AE esta a esquerda de AB
9           T tmp = (b-a)^(e-a);
10          return (tmp > EPS) - (tmp < -EPS);
11 }
12
13 vector<point> convex_hull(vector<point> p) {
14     sort(p.begin(), p.end());
15
16     vector<point> L, U;
17
18     // Lower Hull
19     for(auto pp : p){
20         while(L.size() >= 2 && esq(L[L.size()-2], L.
           back(), pp) == -1)
21             L.pop_back();
22         L.pb(pp);
23     }
24
25     reverse(all(p));
26     // Upper Hull
27     for(auto pp : p){
28         while(U.size() >= 2 && esq(U[U.size()-2], U.
           back(), pp) == -1)
29             U.pop_back();
30         U.pb(pp);
31     }
32
33     L.pop_back();
34     L.insert(L.end(), U.begin(), U.end()-1);
35

```

5 Graph

5.1 BFS

```

1 // BFS
2 // O(V+E)
3
4 const int MAXN { 100010 };
5
6 vector<vector<int> > g(MAXN);
7 vector<bool> visited(MAXN);
8 vector<int> dist(MAXN, oo);
9 queue<int> q;
10
11 void bfs(int s){
12     q.push(s);
13     dist[s] = 0;
14     visited[s] = true;
15
16     while(!q.empty()){
17         int u = q.front(); q.pop();
18
19         for(auto v : g[u]){
20             if(not visited[v]){
21                 dist[v] = dist[u]+1;
22                 visited[v] = true;
23                 q.push(v);
24             }
25         }
26     }
27 }

```

5.2 BellmanFord

```

1 // Bellman Ford - Min distance
2
3 // O(V*E)
4 // Min dist from a start node
5 // Can be applied to negative weights
6
7 using edge = tuple<int, int, int>;
8
9 vector<int> bellman_ford(int s, int N, const vector<
           edge>& edges){
10     const int oo { 1000000010 };
11
12     vector<int> dist(N + 1, oo);
13     dist[s] = 0;
14
15     for (int i = 1; i <= N - 1; i++){
16         for (auto [u, v, w] : edges)
17             if (dist[u] < oo and dist[v] > dist[u] +
           w){
18                 dist[v] = dist[u] + w;
19                 // pred[v]=u to find path
20             }
21
22     return dist;
23 }
24
25 // Identifying negative Cycle
26 bool has_negative_cycle(int s, int N, const vector<
           edge>& edges){
27     const int oo { 1000000010 };
28
29     vector<int> dist(N + 1, oo);
30     dist[s] = 0;
31

```

```

32     for (int i = 1; i <= N - 1; i++)
33         for (auto [u, v, w] : edges)
34             if (dist[u] < oo and dist[v] > dist[u] +
35                 w)
36                 dist[v] = dist[u] + w;
37
38     // If after all rounds, exists a better answer -
39     // Negative cycle found
40     for (auto [u, v, w] : edges)
41         if (dist[u] < oo and dist[v] > dist[u] + w)
42             return true;
43
44     return false;
45 }

```

5.3 Bipartite

```

1 // Checking if graph is Bipartite
2 // O(V+E)
3
4 const int MAXN { 100010 };
5 vector<vector<int>> g(MAXN);
6 vector<int> color(MAXN);
7
8 bool bfs(int s){
9     const int NONE=0,B=1,W=2;
10    queue<int> q;
11    q.push(s);
12    color[s]=B;
13
14    while(!q.empty()){
15        auto u = q.front(); q.pop();
16
17        for(auto v : g[u]){
18            if(color[v] == NONE){
19                color[v]=3-color[u];
20                q.push(v);
21            }else if(color[v]==color[u]){
22                return false;
23            }
24        }
25
26        return true;
27    }
28 }
29
30 bool is_bipartite(int n){
31
32     for (int u = 1; u <= n; u++)
33         if (color[u] == NONE && !bfs(u))
34             return false;
35
36     return true;
37 }

```

5.4 Bridge

```

1 // Algorithm to get bridges in a graph
2
3 using edge = pair<int, int>;
4
5 const int MAX { 100010 };
6 int dfs_num[MAX], dfs_low[MAX];
7 vector<vector<int>> adj;
8
9 void dfs_bridge(int u, int p, int& next, vector<edge>
10 >& bridges){
11
12     dfs_low[u] = dfs_num[u] = next++;
13
14     for (auto v : adj[u])
15         if (not dfs_num[v]) {

```

```

15         dfs_bridge(v, u, next, bridges);
16
17         if (dfs_low[v] > dfs_num[u])
18             bridges.emplace_back(u, v);
19
20         dfs_low[u] = min(dfs_low[u], dfs_low[v]);
21     } else if (v != p)
22         dfs_low[u] = min(dfs_low[u], dfs_num[v]);
23 }
24
25 vector<edge> bridges(int n){
26
27     memset(dfs_num, 0, (n + 1)*sizeof(int));
28     memset(dfs_low, 0, (n + 1)*sizeof(int));
29
30     vector<edge> bridges;
31
32     for (int u = 1, next = 1; u <= n; ++u)
33         if (not dfs_num[u])
34             dfs_bridge(u, u, next, bridges);
35
36     return bridges;
37 }
38 }

```

5.5 CycleDetection

```

1 // Existency of Cycle in a Graph
2
3 // 1. Better to use when path is important
4 // O(V+E)
5 const int MAXN { 100010 };
6 vector<int> visited(MAXN, 0);
7 vector<vector<int>> g(MAXN);
8
9 bool dfs_cycle(int u){
10     if(visited[u]) return false;
11
12     visited[u] = true;
13
14     for(auto v : g[u]){
15         if(visited[v] && v != u) return true;
16         if(dfs_cycle(v)) return true;
17     }
18     return false;
19 }
20
21 bool has_cycle(int n){
22     visited.reset();
23
24     for(int u = 1; u <= n; u++)
25         if(!visited[u] && dfs(u))
26             return true;
27
28     return false;
29 }
30
31 // 2. Better when only detect cycle is important
32 // Only for undirected graphs
33 // When E>V, a cycle exists
34
35 void dfs(int u, function<void(int)> process){
36     if (visited[u])
37         return;
38
39     visited[u] = true;
40
41     process(u);
42
43     for (auto v : adj[u])
44         dfs(v, process);
45 }
46

```

```

47 bool has_cycle(int N) {
48     visited.reset();
49
50     for (int u = 1; u <= N; ++u)
51         if (not visited[u])
52             {
53                 vector<int> cs;
54                 size_t edges = 0;
55
56                 dfs(u, [&](int u) {
57                     cs.push_back(u);
58
59                     for (const auto& v : adj[u])
60                         edges += (visited[v] ? 0 : 1);
61                 });
62
63                 if (edges >= cs.size()) return true;
64             }
65
66     return false;
67 }

```

5.6 DFS

```

1 // DFS
2 // O(n+m)
3 vector<vector<int>> graph(MAX_NODES);
4 vector<bool> visited(MAX_NODES);
5
6 void dfs(int s){
7     if(visited[s]) return;
8     visited[s] = true;
9     for(auto v : graph[s]){
10         dfs(v);
11     }
12 }

```

5.7 Dijkstra

```

1 // Dijkstra
2 // O(V + E log E)
3 #define INF 1e9+10
4 vector<pair<int, int>> adj[MAXN];
5 vector<int> dist;
6 vector<bool> visited;
7 priority_queue<pair<int, int>> q;
8
9 void Dijkstra(int n, int start){
10     for(int i = 0; i <= n; i++){
11         dist.push_back(INF);
12         visited.push_back(false);
13     }
14     dist[start] = 0;
15     q.push(make_pair(0, start));
16     while(!q.empty()){
17         int a = q.top().second; q.pop();
18         if(visited[a]) continue;
19         visited[a] = true;
20         for(auto u : adj[a]){
21             int b = u.first, w = u.second;
22             if(dist[a]+w < dist[b]){
23                 dist[b] = dist[a]+w;
24                 q.push({-dist[b], b});
25             }
26         }
27     }
28 }

```

5.8 Kruskal

```

1 // Minimum Spanning tree
2 // w/ DSU structure

```

```

3
4 struct edge{
5     int a, b, w;
6     bool operator<(edge const& other) {
7         return w < other.w;
8     }
9 };
10
11 /* ----- DSU Structure -----*/
12 int get(int x) {
13     return p[x] = (p[x] == x ? x : get(p[x]));
14 }
15
16 void unite(int a, int b){
17     a = get(a);
18     b = get(b);
19
20     if(r[a] == r[b]) r[a]++;
21     if(r[a] > r[b]) p[b] = a;
22     else p[a] = b;
23 }
24
25 // Initializing values in main()
26 for(int i = 1; i <= n; i++) p[i]=i;
27
28 /* -----*/
29
30 vector<edge> edges, result;
31 int total_weight=0;
32
33 void mst(){
34
35     sort(edges.begin(), edges.end());
36
37     for(auto e : edges){
38         if(get(e.a) != get(e.b)){
39             unite(e.a, e.b);
40             result.pb(e);
41             total_weight+=e.w;
42         }
43     }
44 }

```

5.9 MCBM

```

1 // Augmenting Path Algorithm for Max Cardinality
   Bipartite Matching
2 // O(V*E)
3
4 // Algorithm to find maximum matches between to set
5 // of nodes (bipartite graph)
6
7 vector<int> match, visited;
8
9 int aug(int u){
10     if(visited[u]) return 0;
11     visited[u]=1;
12
13     for(auto v : g[u]){
14         if(match[v]==-1 || aug(match[v])){
15             match[v]=u;
16             return 1;
17         }
18     }
19     return 0;
20 }
21
22 // Inside Main()
23 // Good to try - left v: [0,n-1], right: [n, m-1]
24 int MCBM=0;
25 match.assign(V, -1); // V = all vertices(left+right)
26 for(int i = 0; i < n; i++){ // n = size of left set
27     visited.assign(n, 0);

```

```

28     MCBM+=aug(i);
29 }

```

5.10 Warshall

```

1 // Floyd - Warshall
2 // O(n^3)
3 #define INF 1e9+10
4
5 int adj[MAXN][MAXN];
6 int distances[MAXN][MAXN];
7
8 void Warshall(int n, int start){
9     for (int i = 1; i <= n; i++) {
10         for (int j = 1; j <= n; j++) {
11             if (i == j) distances[i][j] = 0;
12             else if (adj[i][j]) distances[i][j] = adj
[i][j];
13             else distances[i][j] = INF;
14         }
15     }
16     for (int z = 1; z <= n; z++) {
17         for (int i = 1; i <= n; i++) {
18             for (int j = 1; j <= n; j++) {
19                 distances[i][j] = min(distances[i][j]
, distances[i][z] + distances[z][j]);
20             }
21         }
22     }
23 }

```

6 Math

6.1 fast-exponentiation

```

1 // Fast Exponentiation
2 // O(log n)
3
4 ll fexp(ll b, ll e, ll mod) {
5     ll res = 1;
6     b %= mod;
7     while(e){
8         if(e & 1LL)
9             res = (res * b) % mod;
10        e = e >> 1LL;
11        b = (b * b) % mod;
12    }
13    return res;
14 }
15
16 // ll fexp(ll b, ll e){
17 //     if(e == 0){
18 //         return 1;
19 //     }
20 //     ll resp = fexp(b, e/2)%MOD;
21 //     resp = (resp*resp)%MOD;
22 //     if(e%2) resp = (b*resp)%MOD;
23
24 //     return resp;
25 // }

```

6.2 floor-log

```

1 // Find floor(log(x))
2 // O(n)
3 int logv[MAXN];
4 void make_log(){
5     logv[1] = 0;
6     for(int i = 2; i <= MAXN; i++)
7         logv[i] = logv[i/2]+1;
8 }

```

6.3 matrix-exponentiation

```

1 // Matrix Exponentiation
2 // O(log n)
3 #define ll long long int
4 #define vl vector<ll>
5 struct Matrix {
6     vector<vl> m;
7     int r, c;
8
9     Matrix(vector<vl> mat) {
10         m = mat;
11         r = mat.size();
12         c = mat[0].size();
13     }
14
15     Matrix(int row, int col, bool ident=false) {
16         r = row; c = col;
17         m = vector<vl>(r, vl(c, 0));
18         if(ident)
19             for(int i = 0; i < min(r, c); i++)
20                 m[i][i] = 1;
21     }
22
23     Matrix operator*(const Matrix &o) const {
24         assert(c == o.r); // garantir que da pra
multiplicar
25         vector<vl> res(r, vl(o.c, 0));
26
27         for(int i = 0; i < r; i++)
28             for(int j = 0; j < o.c; j++)
29                 for(int k = 0; k < c; k++)
30                     res[i][j] = (res[i][j] + m[i][k]*
o.m[k][j]) % 1000000007;
31
32         return Matrix(res);
33     }
34
35     void printMatrix(){
36         for(int i = 0; i < r; i++)
37             for(int j = 0; j < c; j++)
38                 cout << m[i][j] << " \n"[j == (c-1)];
39     }
40 };
41
42 Matrix fexp(Matrix b, ll e, int n) {
43     if(e == 0) return Matrix(n, n, true); //
identidade
44     Matrix res = fexp(b, e/2LL, n);
45     res = (res * res);
46     if(e%2) res = (res * b);
47
48     return res;
49 }
50
51 // Fibonacci Example O(log n)
52 /* Fibonacci
53    | 1 1 | * | Fn | = | Fn+1 |
54    | 1 0 | * | Fn-1 | = | Fn |
55
56    Generic
57    | a1 a2 ... an | ** K * | Fn-1 | = | Fk+n-1 |
58    | 1 0 ... 0 |          | Fn-2 |   | Fk+n-2 |
59    | 0 1 0 ... 0 |          | Fn-3 |   | Fk+n-3 |
60    ...
61    | 0 0 0 ... 1 0 |          | F0 |   | Fk |
62 */
63
64 int main() {
65     ll n;
66     cin >> n; // Fibonacci(n)
67
68     if(n == 0) {

```

```

69         cout << 0 << endl;
70         return 0;
71     }
72
73     vector<vl> m = {{1LL, 1LL}, {1LL, 0LL}};
74     vector<vl> b = {{1LL}, {0LL}};
75
76     Matrix mat = Matrix(m);
77     Matrix base = Matrix(b);
78
79     mat = fexp(mat, n-1, 2);
80     mat = mat*base;
81
82     cout << mat.m[0][0] << endl;
83
84     return 0;
85 }

```

7 String

7.1 AllSubPalindromes

```

1 // Function to find all Sub palindromes
2 // O(n*n)
3
4 string s; // n = s.size();
5 vector<vector<bool>> is_pal(n, vector<bool>(n, true)
6     );
7 // formando todos os subpalindromos
8 forne(k, 1, n-1)
9     forne(i, 0, n-k-1)
10         is_pal[i][i+k] = (s[i]==s[i+k] && is_pal[i
11             +1][i+k-1]);

```

7.2 General

```

1 // General functions to manipulate strings
2
3 // find function
4 int i = str.find("aa");
5 i = pos ou -1
6
7 // find multiples strings
8 while(i!=string::npos){
9     i = str.find("aa", i);
10 }
11
12 // replace function
13 str.replace(index, (int)size_of_erased, "content");
14 "paablo".replace(1, 2, "a"); // = Pablo
15
16 // string concatenation
17 string a = "pabl"
18 a+="o" or a+='o' or a.pb('o')

```

7.3 Manacher

```

1 // Manacher Algorithm
2 // O(n)
3
4 // Find all sub palindromes in a string
5 // d1 = Odd palin, d2 = Even palin
6
7 vector<int> manacher(string &s, vector<int> &d1,
8     vector<int> &d2) {
9     int n = s.size();
10     for(int i = 0, l = 0, r = -1; i < n; i++) {
11         int k = (i > r) ? 1 : min(d1[l + r - i], r -
12             i + 1);

```

```

11         while(0 <= i - k && i + k < n && s[i - k] ==
12             s[i + k]) {
13             k++;
14         }
15         d1[i] = k--;
16         if(i + k > r) {
17             l = i - k;
18             r = i + k;
19         }
20     }
21     for(int i = 0, l = 0, r = -1; i < n; i++) {
22         int k = (i > r) ? 0 : min(d2[l + r - i + 1],
23             r - i + 1);
24         while(0 <= i - k - 1 && i + k < n && s[i - k
25             - 1] == s[i + k]) {
26             k++;
27         }
28         d2[i] = k--;
29         if(i + k > r) {
30             l = i - k - 1;
31             r = i + k;
32         }
33     }
34     // special vector to construct query by interval
35     vector<int> res(2*n-1);
36     for (int i = 0; i < n; i++) res[2*i] = 2*d1[i]-1;
37     for (int i = 0; i < n-1; i++) res[2*i+1] = 2*d2[i
38         +1];
39     return res;
40 }
41 struct palindrome {
42     vector<int> res;
43
44     palindrome(const& s): res(manacher(s)){
45
46         // Query if [i..j] is palindrome
47         bool is_palindrome(int i, int j){
48             return res[i+j] >= j-i+1;
49         }
50 }

```

7.4 Z-function

```

1 // Z-function
2 // O(n)
3
4 // Return array z(n) that each value z[i] tells the
5 // longest subsequence from i that is prefix of
6 // string s.
7 // Pattern Matching = z-func(s1$s2) acha s1 em s2.
8
9 vector<ll> z_algo(const string &s){
10     ll n = s.size();
11     ll L = 0, R = 0;
12     vector<ll> z(n, 0);
13     for(ll i = 1; i < n; i++){
14         if(i <= R)
15             z[i] = min(z[i-L], R - i + 1);
16         while(z[i]+i < n and s[z[i]+i] == s[z[i]
17             ])
18             z[i]++;
19         if(i+z[i]-1 > R){
20             L = i;
21             R = i + z[i] - 1;
22         }
23     }
24     z[0]=n;
25     return z;

```

```
25 }
```

8 Templates

8.1 template

```
1 #include <bits/stdc++.h>
2 #define ff first
3 #define ss second
4 #define pii pair<int, int>
5 #define vi vector<int>
6 #define ll long long int
7 #define ld long double
8 #define pb push_back
9 #define sws ios_base::sync_with_stdio(false); cin.tie(
  NULL); cout.tie(NULL)
10 #define forn(i, n) for(int i = 0; i < (int)n; i++)
11 #define forne(i, a, b) for(int i = a; i <= b; i++)
12 #define all(x) x.begin(), x.end()
```

```
13 #define teto(a, b) ((a+b-1)/(b))
14 #define dbg(msg, var) cerr << msg << " " << var <<
  endl;
15
16 using namespace std;
17
18 const int INF = 0x3f3f3f3f;
19 const ll LLINF = 0x3f3f3f3f3f3f3f3f;
20 const int MOD = 1000000007;
21 const ld EPS = 1e-8;
22 const ld PI = acos(-1);
23
24 // End Template //
25
26 int main(){
27     sws;
28
29     return 0;
30 }
```