

Competitive programming Notebook

Pablo Arruda Araujo

Sumário

1	Ds	2
1.1	sparse-table	2
1.2	DSU	2
1.3	prefix-sum-array	2
1.4	delta-encoding	2
1.5	Segtree	2
2	Graph	3
2.1	Dijkstra	3
2.2	DSU-MST	3
2.3	BFS	4
2.4	DFS	4
2.5	Warshall	4
3	Algorithm	4
3.1	merge-sort	4
3.2	bsearch-iterative	4
3.3	counting-inversions	5
3.4	kadane	5
4	Math	5
4.1	floor-log	5
4.2	fast-exponentiation	5
4.3	matrix-exponentiation	5
5	Dp	6
5.1	knapsack	6
5.2	LCS	6
5.3	coin-change	6
5.4	unbounded-knapsack	6
6	Geometry	7
6.1	2D	7
6.2	ConvexHull	9

Ds

1.1 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
16     Sparse(vector<int>& v) {
17         int n = v.size();
18         st.assign(n, vector<int>(logn, 0));
19         // Unitary values st[i][0] = v[i, i+2^0] = v[i]
20         for(int i = 0; i < n; i++){
21             st[i][0] = v[i];
22         }
23         // Constructing Sparse Table in O(log n)
24         for(int k = 1; k < logn; k++){
25             for(int i = 0; i < n; i++){
26                 if(i + (1 << k)-1 >= n)
27                     continue;
28                 int prox = i + (1 << (k-1));
29                 st[i][k] = min(st[i][k-1], st[prox][k-1]);
30             }
31         }
32     }
33
34     int f(int a, int b){
35         // Can be: min, max, gcd
36         // f must have idempotent property
37         return min(a, b);
38     }
39     // Queries in O(1)
40     int query(int l, int r){
41         int size = r-l+1;
42         int k = logv[size];
43         // cat jump for queries in O(1)
44         int res = f(st[l][k], st[r - ((1 << k)-1)][k]);
45         return res;
46     }
47 };

```

1.2 DSU

```

1 // Disjoint union set
2 // Operation ~ O(1)
3 int r[MAXN];
4 vector<int> qtd(MAXN, 1);
5
6 int get(int x) {
7     return p[x] = (p[x] == x ? x : get(p[x]));
8 }
9
10 void unite(int a, int b){
11     a = get(a);
12     b = get(b);
13
14     if(r[a] == r[b]){
15         p[a] = b;

```

```

16         r[b]++;
17         qtd[b]+=qtd[a];
18     }else if(r[a] > r[b]){
19         p[b] = a;
20         qtd[a]+=qtd[b];
21     }else{
22         p[a] = b;
23         qtd[b]+=qtd[a];
24     }
25 }
26
27 // Initializing values in main()
28 for(int i = 1; i <= n; i++) p[i]=i;

```

1.3 prefix-sum-array

```

1 // Preffix 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 }

```

1.4 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 }

```

1.5 Segtree

```

1 // Segtree MAX
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
7 // str: limite superior do intervalo que aquele nodo
8 // l : limite inferior do intervalo que queremos
9 // r : limite superior do intervalo que queremos
10 // i : indice do vetor que queremos atualizar
11 // amm: novo valor daquele indice no vetor
12
13 class SegTree{
14     vector<int> st;
15     vector<int> lazy;
16     vector<bool> has;

```

```

17     int size;
18
19     int el_neutro = -(1e9 + 7);
20
21     int f(int a, int b){
22         return max(a,b);
23     }
24
25     void propagate(int sti, int stl, int str){
26         if(has[sti]){
27             st[sti] = lazy[sti]*(str-stl+1);
28             if(stl!=str){
29                 lazy[sti*2+1] = lazy[sti];
30                 lazy[sti*2+2] = lazy[sti];
31
32                 has[sti*2+1] = true;
33                 has[sti*2+2] = true;
34             }
35             has[sti] = false;
36         }
37     }
38
39     int query(int sti, int stl, int str, int l, int r
40 ){
41         if(str < l || stl > r) return el_neutro;
42
43         if(stl >= l && str <= r)
44             return st[sti];
45
46         // intervalo parcialmente incluído em l-r
47         int mid = (stl+str)/2;
48
49         return f(query(2*sti+1, stl, mid, l, r),
50 query(2*sti+2, mid+1, str, l, r));
51
52     void update(int sti, int stl, int str, int i, int
53 amm){
54         if(stl == i && str == i){
55             st[sti] += amm;
56             return;
57         }
58
59         if(stl > i || str < i) return;
60
61         int mid = (stl+str)/2;
62
63         // Processo de atualizacao dos nos filhos
64         update(sti*2+1, stl, mid, i, amm);
65         update(sti*2+2, mid+1, str, i, amm);
66
67         st[sti] = f(st[sti*2+1], st[sti*2+2]);
68     }
69
70     void update_range(int sti, int stl, int str, int
71 l, int r, int amm){
72         if(stl >= l && str <= r){
73             lazy[sti] = amm;
74             has[sti] = true;
75             propagate(sti, stl, str);
76             return;
77         }
78
79         if(stl > r || str < l) return;
80
81         int mid = (stl+str)/2;
82         update_range(sti*2+1, stl, mid, l, r, amm);
83         update_range(sti*2+2, mid+1, str, l, r, amm);
84
85         st[sti] = f(st[sti*2+1], st[sti*2+2]);
86     }
87
88     public:

```

```

86     SegTree(int n): st(4*n, 0){size=n;}
87     int query(int l, int r){return query(0,0,size
88 -1,l,r);}
89     void update(int i, int amm){update(0,0,size
90 -1,i,amm);}
91     void update_range(int l, int r, int amm){
92         update_range(0,0,size-1,l,r,amm);}
93 }
94
95 // In main()
96 SegTree st(v.size());
97 for(int i = 0; i < n; i++){
98     st.update(i, v[i]);
99 }

```

2 Graph

2.1 Dijkstra

```

1 // Dijkstra
2 // O(n + m log m)
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 }

```

2.2 DSU-MST

```

1 // Minimum Spanning tree
2 // w/ DSU structure
3
4 typedef struct{
5     int a, b;
6     int w;
7 } edge;
8
9 /* ----- DSU Structure -----*/
10 int get(int x) {
11     return p[x] = (p[x] == x ? x : get(p[x]));
12 }
13
14 void unite(int a, int b){
15     a = get(a);
16     b = get(b);
17
18     if(r[a] == r[b]) r[a]++;
19     if(r[a] > r[b]) p[b] = a;

```

```

20     else p[a] = b;
21 }
22
23 // Initializing values in main()
24 for(int i = 1; i <= n; i++) p[i]=i;
25
26 /* ----- */
27
28 vector<edge> edges;
29 int total_weight;
30
31 void mst(){
32     // sort edges
33     for(auto e : edges){
34         if(get(e.a) != get(e.b)){
35             unite(e.a, e.b);
36             total_weight+=e.w;
37         }
38     }
39 }

```

2.3 BFS

```

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

```

2.4 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 }

```

2.5 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]
20                                     , distances[i][z] + distances[z][j]);
21             }
22         }
23     }
24 }

```

3 Algorithm

3.1 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 }

```

3.2 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 }

```

3.3 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 }

```

3.4 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 }

```

4 Math

4.1 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 }

```

4.2 fast-exponentiation

```

1 // Fast Exponentiation
2 // O(log n)
3 ll fexp(ll b, ll e){
4     if(e == 0){
5         return 1;
6     }
7     ll resp = fexp(b, e/2)%MOD;
8     resp = (resp*resp)%MOD;
9     if(e%2) resp = (b*resp)%MOD;
10
11    return resp;
12 }

```

4.3 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
25         multiplicar
26         vector<vl> res(r, vl(o.c, 0));
27
28         for(int i = 0; i < r; i++)
29             for(int j = 0; j < o.c; j++)
30                 for(int k = 0; k < c; k++)
31                     res[i][j] = (res[i][j] + m[i][k]*
32                     o.m[k][j]) % 1000000007;
33
34         return Matrix(res);
35     }
36
37     void printMatrix(){
38         for(int i = 0; i < r; i++)
39             for(int j = 0; j < c; j++)
40                 cout << m[i][j] << " \n"[j == (c-1)];
41     }
42 };
43
44 Matrix fexp(Matrix b, ll e, int n) {
45     if(e == 0) return Matrix(n, n, true); //
46     identidade
47     Matrix res = fexp(b, e/2LL, n);
48     res = (res * res);
49     if(e%2) res = (res * b);
50
51    return res;
52 }
53
54 // Fibonacci Example 0 (log n)
55 /* Fibonacci
56 |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 }

```

5 Dp

5.1 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

```

```

34         dp[i][j] = dp[i-1][j];
35     }
36 }
37 return dp[n][w];
38 }
39
40 int val[MAX], wt[MAX], dp[MAX]; // Optimization for
    space
41 int solve(int n, int W){
42     for(int i=0; i < n; i++)
43         for(int j=W; j>=wt[i]; j--)
44             dp[j] = max(dp[j],dp[j-wt[i]]+val[i]);
45     return dp[W];
46 }

```

5.2 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 }

```

5.3 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 }

```

5.4 unbouded-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 }

```

6 Geometry

6.1 2D

```

1 // 2D structures template
2
3 // Code from - Github: Tiagosf00/Competitive-
4 // Programming !!
5 // Writer: Tiago de Souza Fernandes
6
7 #define EPS 1e-6
8 #define PI acos(-1)
9 #define vp vector<point>
10
11 // typedef int cod;
12 // bool eq(cod a, cod b){ return (a==b); }
13 typedef ld cod;
14 bool eq(cod a, cod b){ return abs(a - b) <= EPS; }
15
16 struct point{
17     cod x, y;
18     int id;
19     point(cod x=0, cod y=0): x(x), y(y){}
20
21     point operator+(const point &o) const{
22         return {x+o.x, y+o.y};
23     }
24     point operator-(const point &o) const{
25         return {x-o.x, y-o.y};
26     }
27     point operator*(cod t) const{
28         return {x*t, y*t};
29     }
30     point operator/(cod t) const{
31         return {x/t, y/t};
32     }
33     cod operator*(const point &o) const{ //dot
34         return x * o.x + y * o.y;
35     }
36     cod operator^(const point &o) const{ // cross
37         return x * o.y - y * o.x;
38     }
39     bool operator<(const point &o) const{
40         if(!eq(x, o.x)) return x < o.x;
41         return y < o.y;
42     }
43     bool operator==(const point &o) const{
44         return eq(x, o.x) and eq(y, o.y);
45     }
46 };
47
48 ld norm(point a){ // Modulo
49     return sqrt(a*a);
50 }
51
52 bool nulo(point a){
53     return (eq(a.x, 0) and eq(a.y, 0));
54 }
55

```

```

56 int ccw(point a, point b, point e){ //-1=dir; 0=
57     collinear; 1=esq;
58     cod tmp = (b-a)^(e-a); // from a to b
59     return (tmp > EPS) - (tmp < -EPS);
60     // if int: tira comentario
61     // if(tmp==0) return 0;
62     // if(tmp>0) return 1;
63     // return -1;
64 }
65 point rotccw(point p, ld a){
66     // a = PI*a/180; // graus
67     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
68     +p.x*sin(a)));
69 }
70
71 point rot90cw(point a) { return point(a.y, -a.x); };
72 point rot90ccw(point a) { return point(-a.y, a.x); };
73
74 ld proj(point a, point b){ // a sobre b
75     return a*b/norm(b);
76 }
77
78 ld angle(point a, point b){ // em radianos
79     ld ang = a*b / norm(a) / norm(b);
80     return acos(max(min(ang, (ld)1), (ld)-1));
81 }
82 ld angle_vec(point v){
83     // return 180/PI*atan2(v.x, v.y); // graus
84     return atan2(v.x, v.y);
85 }
86 ld order_angle(point a, point b){ // from a to b ccw
87     (a in front of b)
88     ld aux = angle(a,b)*180/PI;
89     return ((a^b)<=0 ? aux:360-aux);
90 }
91
92 bool angle_less(point a1, point b1, point a2, point
93     b2){ // ang(a1,b1) <= ang(a2,b2)
94     point p1((a1*b1), abs((a1^b1)));
95     point p2((a2*b2), abs((a2^b2)));
96     return (p1^p2) <= 0;
97 }
98
99 ld area(vp &p){ // (points sorted)
100     ld ret = 0;
101     for(int i=2;i<(int)p.size();i++)
102         ret += (p[i]-p[0])^(p[i-1]-p[0]);
103     return abs(ret/2);
104 }
105
106 ld areaT(point &a, point &b, point &c){
107     return abs((b-a)^(c-a))/2.0;
108 }
109
110 point center(vp &A){
111     point c = point();
112     int len = A.size();
113     for(int i=0;i<len;i++)
114         c=c+A[i];
115     return c/len;
116 }
117
118 point forca_mod(point p, ld m){
119     ld cm = norm(p);
120     if(cm<EPS) return point();
121     return point(p.x*m/cm, p.y*m/cm);
122 }
123
124 ///////////////
125 // Line //
126 ///////////////
127
128 struct line{
129     point p1, p2;
130     cod a, b, c; // ax+by+c = 0;
131     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)

```

```

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

```



```

262
263 vp inter_circle_line(circle C, line L){
264     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
265         p1)*(ab) / (ab*ab));
266     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
267         / (ab*ab);
268     if (h2 < 0) return {};
269     if (h2 == 0) return {p};
270     point h = (ab/norm(ab)) * sqrt(h2);
271     return {p - h, p + h};
272 }
273
274 vp inter_circle(circle C1, circle C2){
275     if(C1.c == C2.c) { assert(C1.r != C2.r); return
276         {};}
277     point vec = C2.c - C1.c;
278     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
279     ;
280     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
281         C1.r*C1.r - p*p*d2;
282     if (sum*sum < d2 or dif*dif > d2) return {};
283     point mid = C1.c + vec*p, per = point(-vec.y, vec
284         .x) * sqrt(max((ld)0, h2) / d2);
285     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
286     return {mid + per, mid - per};
287 }
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

6.2 ConvexHull

```

1 // Convex Hull
2 // Algorithm: Monotone Chain
3 // Complexity: O(n) + ordenacao O(nlogn)
4
5 // Regra mao direita p2->p1 (dedao p cima ? esq : dir
6     || colinear)

```