

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	Bipartite	3
2.3	BFS	4
2.4	Kruskal	4
2.5	BellmanFord	4
2.6	DFS	5
2.7	MCBM	5
2.8	Bridge	5
2.9	Warshall	5
2.10	CycleDetection	5
3	Algorithm	6
3.1	merge-sort	6
3.2	bsearch-iterative	6
3.3	counting-inversions	6
3.4	kadane	6
4	Math	7
4.1	floor-log	7
4.2	fast-exponentiation	7
4.3	matrix-exponentiation	7
5	Dp	7
5.1	knapsack	7
5.2	LCS	8
5.3	coin-change	8
5.4	unbouded-knapsack	8
6	String	8
6.1	General	8
6.2	Manacher	8
6.3	Z-function	9
6.4	Trie	9
6.5	AllSubPalindromes	9
6.6	Kmp	9
7	Geometry	9
7.1	2D	9
7.2	ConvexHull	11

1 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(a == b) return;
15

```

```

16     if(r[a] == r[b]){
17         p[a] = b;
18         r[b]++;
19         qtd[b]+=qtd[a];
20     }else if(r[a] > r[b]){
21         p[b] = a;
22         qtd[a]+=qtd[b];
23     }else{
24         p[a] = b;
25         qtd[b]+=qtd[a];
26     }
27 }
28
29 // Initializing values in main()
30 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
53 void update(int sti, int stl, int str, int i, int
54             amm){
55     if(stl == i && str == i){
56         st[sti] += amm;
57         return;
58     }
59
60     if(stl > i || str < i) return;
61
62     int mid = (stl+str)/2;
63
64     // Processo de atualizacao dos nos filhos
65     update(sti*2+1, stl, mid, i, amm);
66     update(sti*2+2, mid+1, str, i, amm);
67
68     st[sti] = f(st[sti*2+1], st[sti*2+2]);
69 }
70
71 void update_range(int sti, int stl, int str, int
72                  l, int r, int amm){
73     if(stl >= l && str <= r){
74         lazy[sti] = amm;
75         has[sti] = true;
76         propagate(sti, stl, str);
77         return;
78     }
79
80     if(stl > r || str < l) return;
81
82     int mid = (stl+str)/2;
83     update_range(sti*2+1, stl, mid, l, r, amm);
84     update_range(sti*2+2, mid+1, str, l, r, amm);
85
86     st[sti] = f(st[sti*2+1], st[sti*2+2]);
87 }

```

```

84
85 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
98 for(int i = 0; i < n; i++){
99     st.update(i, v[i]);
100 }

```

2 Graph

2.1 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 }

```

2.2 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 }

```

2.3 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 }

```

2.4 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 /* ----- main -----*/
29
30 vector<edge> edges, result;
31 int total_weight;
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 }

```

2.5 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<
10     edge>& edges){
11     const int oo { 1000000010 };
12
13     vector<int> dist(N + 1, oo);
14     dist[s] = 0;
15
16     for (int i = 1; i <= N - 1; i++)
17         for (auto [u, v, w] : edges)
18             if (dist[u] < oo and dist[v] > dist[u] +
19                 w){
20                 dist[v] = dist[u] + w;
21                 // pred[v]=u to find path
22             }
23
24     return dist;
25 }
26
27 // Identifying negative Cycle
28 bool has_negative_cycle(int s, int N, const vector<
29     edge>& edges){
30     const int oo { 1000000010 };
31
32     vector<int> dist(N + 1, oo);
33     dist[s] = 0;
34
35     for (int i = 1; i <= N - 1; i++)
36         for (auto [u, v, w] : edges)
37             if (dist[u] < oo and dist[v] > dist[u] +
38                 w)
39                 dist[v] = dist[u] + w;
40
41     // If after all rounds, exists a better answer -
42     // Negative cycle found
43     for (auto [u, v, w] : edges)
44         if (dist[u] < oo and dist[v] > dist[u] + w)
45             return true;

```

```

41         return false;
42     }
43 }

```

2.6 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 }

```

2.7 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 }

```

2.8 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]) {
16             dfs_bridge(v, u, next, bridges);
17

```

```

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

```

2.9 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
13                 [i][j];
14             else distances[i][j] = INF;
15         }
16     }
17     for (int z = 1; z <= n; z++) {
18         for (int i = 1; i <= n; i++) {
19             for (int j = 1; j <= n; j++) {
20                 distances[i][j] = min(distances[i][j]
21                     , distances[i][z] + distances[z][j]);
22             }
23         }
24     }
25 }

```

2.10 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 }

```

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
13    int ans = 0;
14    ans += merge_sort(l);
15    ans += merge_sort(r);
16
17    l.push_back(1e9);
18    r.push_back(1e9);
19
20    int inil = 0, inir = 0;
21
22    for(int i = 0; i < v.size(); i++){
23        if(l[inil] <= r[inir]) v[i] = l[inil++];
24        else{
25            v[i] = r[inir++];
26            ans += l.size() - inil - 1;
27        }
28    }
29
30    return ans;
31 }

```

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         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]*
31                     o.m[k][j]) % 1000000007;
32                 }
33             }
34         }
35         return Matrix(res);
36     }
37
38     void printMatrix(){
39         for(int i = 0; i < r; i++){
40             for(int j = 0; j < c; j++){
41                 cout << m[i][j] << " \n"[j == (c-1)];
42             }
43         }
44     };
45
46     Matrix fexp(Matrix b, ll e, int n) {
47         if(e == 0) return Matrix(n, n, true); //
48         identidade
49         Matrix res = fexp(b, e/2LL, n);
50         res = (res * res);
51         if(e%2) res = (res * b);
52         return res;
53     }
54
55 // Fibonacci Example 0 (log n)
56 /* Fibonacci
57 | 1 1|*|Fn | = |Fn+1|
58 | 1 0| |Fn-1| |Fn |
59
60 Generic
61 |a1 a2 ... an| ** K * |Fn-1| = |Fk+n-1|
62 |1 0 ... 0| |Fn-2| |Fk+n-2|
63 |0 1 0 ... 0| |Fn-3| |Fk+n-3|
64 ...
65 |0 0 0 ... 1 0| |F0 | |Fk |
66 */
67
68 int main() {
69     ll n;
70     cin >> n; // Fibonacci(n)
71
72     if(n == 0) {
73         cout << 0 << endl;
74         return 0;
75     }
76
77     vector<vl> m = {{1LL, 1LL}, {1LL, 0LL}};
78     vector<vl> b = {{1LL}, {0LL}};
79
80     Matrix mat = Matrix(m);
81     Matrix base = Matrix(b);
82
83     mat = fexp(mat, n-1, 2);
84     mat = mat*base;
85
86     cout << mat.m[0][0] << endl;
87
88     return 0;
89 }

```

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
35             dp[i][j] = dp[i-1][j];
36     }
37 }
38 return dp[n][w];
39 }
40 int val[MAX], wt[MAX], dp[MAX]; // Optimization for
41 // space
42 int solve(int n, int W){
43     for(int i=0; i < n; i++)
44         for(int j=W; j>=wt[i]; j--)
45             dp[j] = max(dp[j],dp[j-wt[i]]+val[i]);
46     return dp[W];
47 }

```

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 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 }

```

6 String

6.1 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')

```

6.2 Manacher

```

1 // Manacher Algorithm
2 // O(n)
3
4 // Find all sub palindromes in a string
5 // d1 = Odd palin, d2 = Even palin

```




```

6 vector<int> manacher(string &s, vector<int> &d1,
7 vector<int> &d2) {
8     int n = s.size();
9     for(int i = 0, l = 0, r = -1; i < n; i++) {
10         int k = (i > r) ? 1 : min(d1[l + r - i], r -
11 i + 1);
12         while(0 <= i - k && i + k < n && s[i - k] ==
13 s[i + k]) {
14             k++;
15         }
16         d1[i] = k--;
17         if(i + k > r) {
18             l = i - k;
19             r = i + k;
20         }
21     }
22     for(int i = 0, l = 0, r = -1; i < n; i++) {
23         int k = (i > r) ? 0 : min(d2[l + r - i + 1],
24 r - i + 1);
25         while(0 <= i - k - 1 && i + k < n && s[i - k
26 - 1] == s[i + k]) {
27             k++;
28         }
29         d2[i] = k--;
30         if(i + k > r) {
31             l = i - k - 1;
32             r = i + k;
33         }
34     }
35     // special vector to construct query by interval
36     vector<int> res(2*n-1);
37     for (int i = 0; i < n; i++) res[2*i] = 2*d1[i]-1;
38     for (int i = 0; i < n-1; i++) res[2*i+1] = 2*d2[i
39 +1];
40     return res;
41 }
42
43 struct palindrome {
44     vector<int> res;
45
46     palindrome(const& s): res(manacher(s)){}
47
48     // Query if [i..j] is palindrome
49     bool is_palindrome(int i, int j){
50         return res[i+j] >= j-i+1;
51     }
52 }

```

6.3 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]++;

```

```

18         if(i+z[i]-1 > R){
19             L = i;
20             R = i + z[i] - 1;
21         }
22     }
23     z[0]=n;
24     return z;
25 }

```

6.4 Trie

6.5 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]);

```

6.6 Kmp

7 Geometry

7.1 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
66 point rotccw(point p, ld a){
67     // a = PI*a/180; // graus
68     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+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
83 ld angle_vec(point v){
84     // return 180/PI*atan2(v.x, v.y); // graus
85     return atan2(v.x, v.y);
86 }
87
88 ld order_angle(point a, point b){ // from a to b ccw
89     (a in front of b)
90     ld aux = angle(a,b)*180/PI;
91     return ((a^b)<=0 ? aux:360-aux);
92 }
93
94 bool angle_less(point a1, point b1, point a2, point
95     b2){ // ang(a1,b1) <= ang(a2,b2)
96     point p1((a1*b1), abs((a1^b1)));
97     point p2((a2*b2), abs((a2^b2)));
98     return (p1^p2) <= 0;
99 }
100
101 ld area(vp &p){ // (points sorted)
102     ld ret = 0;
103     for(int i=2;i<(int)p.size();i++)
104         ret += (p[i]-p[0])^(p[i-1]-p[0]);
105     return abs(ret/2);
106 }
107
108 ld areaT(point &a, point &b, point &c){
109     return abs((b-a)^(c-a))/2.0;
110 }
111
112 point center(vp &A){
113     point c = point();
114     int len = A.size();
115     for(int i=0;i<len;i++)
116         c=c+A[i];
117
118     return c/len;
119 }
120
121 point forca_mod(point p, ld m){
122     ld cm = norm(p);
123     if(cm<EPS) return point();
124     return point(p.x*m/cm,p.y*m/cm);
125 }
126
127 ///////////////
128 // Line //
129 ///////////////
130
131 struct line{
132     point p1, p2;
133     cod a, b, c; // ax+by+c = 0;
134     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
135     line(point p1=0, point p2=0): p1(p1), p2(p2){
136         a = p1.y-p2.y;
137         b = p2.x-p1.x;
138         c = -(a*p1.x + b*p1.y);
139     }
140     line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
141     {
142         // Gera os pontos p1 p2 dados os coeficientes
143         // isso aqui eh horrivel mas quebra um galho
144         kkkkkk
145         if(b==0){
146             p1 = point(1, -c/a);
147             p2 = point(0, -c/a);
148         }else{
149             p1 = point(1, (-c-a*1)/b);
150             p2 = point(0, -c/b);
151         }
152     }
153
154     cod eval(point p){
155         return a*p.x+b*p.y+c;
156     }
157     bool inside(point p){
158         return eq(eval(p), 0);
159     }
160     point normal(){
161         return point(a, b);
162     }
163 }
164
165 bool inside_seg(point p){
166     return (inside(p) and
167         min(p1.x, p2.x)<=p.x and p.x<=max(p1.
168         x, p2.x) and
169         min(p1.y, p2.y)<=p.y and p.y<=max(p1.
170         y, p2.y));
171 }
172
173 vp inter_line(line l1, line l2){
174     ld det = l1.a*l2.b - l1.b*l2.a;
175     if(det==0) return {};
176     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
177     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
178     return {point(x, y)};
179 }
180
181 point inter_seg(line l1, line l2){
182     point ans = inter_line(l1, l2);
183     if(ans.x==INF or !l1.inside_seg(ans) or !l2.
184     inside_seg(ans))
185         return point(INF, INF);
186     return ans;
187 }

```

```

175 ld dseg(point p, point a, point b){ // point - seg
176     if(((p-a)*(b-a)) < EPS) return norm(p-a);
177     if(((p-b)*(a-b)) < EPS) return norm(p-b);
178     return abs((p-a)^(b-a))/norm(b-a);
179 }
180
181 ld dline(point p, line l){ // point - line
182     return abs(l.eval(p))/sqrt(1.a*1.a + 1.b*1.b);
183 }
184
185 line mediatrix(point a, point b){
186     point d = (b-a)*2;
187     return line(d.x, d.y, a*a - b*b);
188 }
189
190 line perpendicular(line l, point p){ // passes
191     through p
192     return line(1.b, -1.a, -1.b*p.x + 1.a*p.y);
193 }
194
195 ///////////////
196 // Circle //
197 ///////////////
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     bool inside(const point &a) const{
214         return norm(a - c) <= r;
215     }
216     pair<point, point> getTangentPoint(point p) {
217         ld d1 = norm(p-c), theta = asin(r/d1);
218         point p1 = rotccw(c-p, -theta);
219         point p2 = rotccw(c-p, theta);
220         p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
221         p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
222         return {p1,p2};
223     }
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++){
232         if(!ans.inside(v[i])){
233             ans = circle(v[i]);
234             for(int j=0; j<i; j++){
235                 if(!ans.inside(v[j])){
236                     ans = circle(v[i], v[j]);
237                     for(int k=0; k<j; k++){
238                         if(!ans.inside(v[k])){
239                             ans = circle(v[i], v[j], v[k]);
240                         }
241                     }
242                 }
243             }
244         }
245     }
246     return ans;
247 }
248
249 circle incircle( point p1, point p2, point p3 ){
250     ld m1=norm(p2-p3);
251     ld m2=norm(p1-p3);
252     ld m3=norm(p1-p2);
253     point c = (p1*m1+p2*m2+p3*m3)*(1/(m1+m2+m3));
254     ld s = 0.5*(m1+m2+m3);
255     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3))/s;
256     return circle(c, r);
257 }
258
259 circle circumcircle(point a, point b, point c) {
260     circle ans;
261     point u = point((b-a).y, -(b-a).x);
262     point v = point((c-a).y, -(c-a).x);
263     point n = (c-b)*0.5;
264     ld t = (u^n)/(v^u);
265     ans.c = ((a+c)*0.5) + (v*t);
266     ans.r = norm(ans.c-a);
267     return ans;
268 }
269
270 vp inter_circle_line(circle C, line L){
271     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
272     p1)*(ab) / (ab*ab));
273     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
274     / (ab*ab);
275     if (h2 < 0) return {};
276     if (h2 == 0) return {p};
277     point h = (ab/norm(ab)) * sqrt(h2);
278     return {p - h, p + h};
279 }
280
281 vp inter_circle(circle C1, circle C2){
282     if(C1.c == C2.c) { assert(C1.r != C2.r); return
283     {}; }
284     point vec = C2.c - C1.c;
285     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
286     ;
287     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
288     C1.r*C1.r - p*p*d2;
289     if (sum*sum < d2 or dif*dif > d2) return {};
290     point mid = C1.c + vec*p, per = point(-vec.y, vec
291     .x) * sqrt(max((ld)0, h2) / d2);
292     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
293     return {mid + per, mid - per};
294 }

```

7.2 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)
7
8 int esq(point p1, point p2, point p3){
9     cod cross = (p2-p1)^(p3-p1);
10    if(cross == 0) return 0;
11    else if(cross > 0) return 1;
12    return -1;
13 }
14
15 vector<point> convex_hull(vector<point> p) {
16     sort(p.begin(), p.end());
17
18     vector<point> L, U;
19
20     // Lower Hull
21     for(auto pp : p){
22         while(L.size() >= 2 && esq(L[L.size()-2], L.
23         back(), pp) == -1)
24             L.pop_back();
25         L.pb(pp);
26     }

```

```
25         reverse(all(p));
26         // Upper Hull
27         for(auto pp : p){
28             while(U.size() >= 2 && esq(U[U.size()-2], U.
29 back(), pp) == -1)
30                 U.pop_back();
31             U.pb(pp);
32         }
33
34         L.pop_back();
35         L.insert(L.end(), U.begin(), U.end()-1);
36
37         return L;
38     }
```