

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	8
5.1	knapsack	8
5.2	LCS	8
5.3	kadane-dp	8
5.4	coin-change	8
5.5	unbounded-knapsack	9
6	String	9
6.1	General	9
6.2	Manacher	9
6.3	Z-function	9
6.4	AllSubPalindromes	9
7	Geometry	10
7.1	2D	10
7.2	ConvexHull	12

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 struct DSU {
4     int n = 0;
5     vector<int> p;
6     vector<int> r;
7
8     DSU(int nn){
9         n = nn;
10        r.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(r[a] < r[b]) swap(a, b);
23        p[b] = a;
24        r[a] += r[b];
25    }
26 };
27
28 // Initializing values in main()
29 // DSU(n+1)

```

1.3 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 }

```

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 }

```

```

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

```

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<(const edge& 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 }

```

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;

```

```

42     return false;
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>
  & bridges){
10
11     dfs_low[u] = dfs_num[u] = next++;
12
13     for (auto v : adj[u])
14         if (not dfs_num[v]) {
15
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
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 // }

```

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         }
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 Matrix fexp(Matrix b, ll e, int n) {
46     if(e == 0) return Matrix(n, n, true); //
47     identidade
48     Matrix res = fexp(b, e/2LL, n);
49     res = (res * res);
50     if(e%2) res = (res * b);
51     return res;
52 }
53
54 // Fibonacci Example 0 (log n)
55 /*
56     | 1 1 | * | Fn | = | Fn+1 |
57     | 1 0 | * | Fn-1 | = | Fn |
58
59     Generic
60     | a1 a2 ... an | ** K * | Fn-1 | = | Fk+n-1 |
61     | 1 0 ... 0 |          | Fn-2 |   | Fk+n-2 |
62     | 0 1 0 ... 0 |          | Fn-3 |   | Fk+n-3 |
63     ...
64     | 0 0 0 ... 1 0 |        | F0 |   | Fk |
65 */
66
67 int main() {
68     ll n;
69     cin >> n; // Fibonacci(n)
70
71     if(n == 0) {
72         cout << 0 << endl;
73         return 0;
74     }
75
76     vector<vl> m = {{1LL, 1LL}, {1LL, 0LL}};
77     vector<vl> b = {{1LL}, {0LL}};
78
79     Matrix mat = Matrix(m);
80     Matrix base = Matrix(b);
81
82     mat = fexp(mat, n-1, 2);
83     mat = mat*base;
84
85     cout << mat.m[0][0] << endl;

```

```

85     return 0;
86 }

```

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
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 }

```

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 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 }

```

5.4 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 }

```

5.5 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 }

```

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
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 - i + 1);
12        while(0 <= i - k && i + k < n && s[i - k] == 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 }

```

```

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 - 1] == s[i + k]) {
25         k++;
26     }
27     d2[i] = k--;
28     if(i + k > r) {
29         l = i - k - 1;
30         r = i + k;
31     }
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+1];
38 return res;
39 }
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 }

```

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
8 // Pattern Matching = z-func(s1$s2) acha s1 em s2.
9
10 vector<ll> z_algo(const string &s){
11     ll n = s.size();
12     ll L = 0, R = 0;
13     vector<ll> z(n, 0);
14     for(ll i = 1; i < n; i++){
15         if(i <= R)
16             z[i] = min(z[i-L], R - i + 1);
17         while(z[i]+i < n and s[z[i]+i] == s[z[i]])
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;
26 }

```

6.4 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
+1][i+k-1]);

```

7 Geometry

7.1 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
68 pt rotccw(pt p, ld a){ // rotacionar ccw
69     // a = PI*a/180; // graus
70     return pt((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.
x*sin(a)));
71 }
72
73 pt rot90cw(pt p) { return pt(p.y, -p.x); };
74
75 pt rot90ccw(pt p) { return pt(-p.y, p.x); };
76
77 ld proj(pt a, pt b){ // a sobre b
78     return a*b/norm(b);
79 }
80
81 int paral(pt u, pt v) { // se u e v sao paralelos
82     if (!eq(u^v, 0)) return 0;
83     if ((u.x > EPS) == (v.x > EPS) && (u.y > EPS) ==
(v.y > EPS))
84         return 1;
85     return -1;
86 }
87
88 pt mirror(pt m1, pt m2, pt p){
89     // mirror pt p around segment m1m2
90     pt seg = m2-m1;
91     ld t0 = ((p-m1)*seg) / (seg*seg);
92     pt ort = m1 + seg*t0;
93     pt pm = ort-(p-ort);
94     return pm;
95 }
96
97 pt center(vp &A){ // center of pts
98     pt c = pt();
99     int len = A.size();
100     for(int i=0;i<len;i++)
101         c=c+A[i];
102     return c/len;
103 }
104
105 bool simetric(vector<pt> &a){ // ordered - check
106     simetric pt
107     int n = a.size(); // . . . . ok / . . . !ok
108     pt c = center(a);
109     if(n&1) return false;
110     for(int i=0;i<n/2;i++)
111         if(!col(a[i], a[i+n/2], c))
112             return false;
113     return true;
114 }
115
116 //\ LINE /\
117
118 struct line{ // line or line segment

```

```

118 T a, b, c;
119 pt p1, p2; // ax + by + c = 0 -> y = ((-a/b)x - (c/b))
120 line(pt p1, pt p2): p1(p1), p2(p2){
121     a = p1.y-p2.y; b = p2.x-p1.x; c = -(a*p1.x + b*p1.y);
122 }
123
124 line(T a, T b, T c): a(a), b(b), c(c){
125     if(b == 0){ p1 = pt(0, -c/a); p2 = pt(0, -c/a); }
126     else{
127         p1 = pt(1, (-c-a*1)/b);
128         p2 = pt(0, -c/b);
129     }
130 }
131
132 T eval(pt p){ // value of {x,y} on line
133     return a*p.x+b*p.y+c;
134 }
135
136 bool insideLine(pt p){ // check if pt is inside line
137     return eq(eval(p), 0);
138 }
139
140 bool insideSeg(pt p){ // check if pt is inside line seg
141     return (insideLine(p) &&
142             min(p1.x, p2.x)<=p.x && p.x<=max(p1.x, p2.x) &&
143             min(p1.y, p2.y)<=p.y && p.y<=max(p1.y, p2.y));
144 }
145
146 pt normal(){ // normal vector
147     return pt(a, b);
148 }
149 };
150
151
152 vp intersecLine(line l1, line l2){ // pt of two line
153     intersec
154     ld det = l1.a*l2.b - l1.b*l2.a;
155     if(det==0) return {};
156     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
157     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
158     return {pt(x, y)};
159 }
160
161 vp intersecSeg(line l1, line l2){ // intersec of two line seg
162     vp ans = intersecLine(l1, l2);
163     if(ans.empty() || !l1.insideSeg(ans[0]) || !l2.insideSeg(ans[0]))
164         return {};
165     return ans;
166 }
167
168 ld dSeg(pt p, pt a, pt b){ // distance - pt to line seg
169     if(((p-a)*(b-a)) < EPS) return norm(p-a);
170     if(((p-b)*(a-b)) < EPS) return norm(p-b);
171     return abs((p-a)^(b-a))/norm(b-a);
172 }
173
174 ld dLine(pt p, line l){ // pt - line
175     return abs(l.eval(p))/sqrt(1.a*1.a + 1.b*1.b);
176 }
177
178 bool paraleline(line r, line s) { // se r e s sao paralelas
179     return paral(r.p1 - r.p2, s.p1 - s.p2);
180 }
181
182 line perpendicular(line l, pt p){ // passes through p
183     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
184 }
185
186 line bisector(line l){ // bisector of a line segment
187     pt mid = pt((l.p1.x + l.p2.x)/2, (l.p1.y + l.p2.y)/2);
188     return perpendicular(l, mid);
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) { // 0(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- 0(n*m)
221     int n = v1.size(), m = v2.size();
222     for (int i = 0; i < n; i++) if (isInside(v2, v1[i])) return 1;
223     for (int i = 0; i < n; i++) if (isInside(v1, v2[i])) return 1;
224     for (int i = 0; i < n; i++) for (int j = 0; j < m; j++)
225         if (intersecSeg(line(v1[i], v1[(i+1)%n]), line(v2[j], v2[(j+1)%m])).size() != 0) return 1;
226     return 0;
227 }
228
229 // ld distPol(vector<pt> v1, vector<pt> v2) { // distancia de poligonos
230 //     if (isIntersec(v1, v2)) return 0;
231 //     ld ret = DINF;
232 //     for (int i = 0; i < v1.size(); i++){
233 //         for (int j = 0; j < v2.size(); j++){
234 //             ret = min(ret, dSeg(line(v1[i], v1[(i+1)%v1.size()]), line(v2[j], v2[(j+1)%v2.size()])));
235 //         }
236 //     }
237 //     return ret;
238 // }
239 // }
240 // }
241

```

```

242 //\ Circle /\
243
244
245 struct circle{
246     pt c; T r;
247     circle() : c(0, 0), r(0){}
248     circle(const pt o) : c(o), r(0){}
249     circle(const pt a, const pt b){
250         c = (a+b)/2;
251         r = norm(a-c);
252     }
253     bool inside(const pt &a) const{
254         return norm(a - c) <= r;
255     }
256     pair<pt, pt> getTangent(pt p) {
257         ld d1 = norm(p-c), theta = asin(r/d1);
258         pt p1 = rotccw(c-p, -theta);
259         pt p2 = rotccw(c-p, theta);
260         p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
261         p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
262         return {p1,p2};
263     }
264 };
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

```

```

309 // circle minCircleCover(vector<pt> v){ // 0(n) min
310     circle that cover all pts
311     // random_shuffle(v.begin(), v.end());
312     circle ans;
313     int n = v.size();
314     for(int i=0;i<n;i++) if(!ans.inside(v[i])){
315         ans = circle(v[i]);
316         for(int j=0;j<i;j++) if(!ans.inside(v[j]))
317             {
318                 ans = circle(v[i], v[j]);
319                 for(int k=0;k<j;k++) if(!ans.inside(v[
320                     k])){
321                     ans = circle(v[i], v[j], v[k]);
322                 }
323             }
324     }
325     return ans;
326 }
327
328 //\ EXTRA C++ complex library /\
329
330 typedef double T;
331 typedef complex<T> pt;
332 #define x real()
333 #define y imag()
334
335 pt p{3,-4};
336 cout << p.x << " " << p.y << "\n"; // 3 -4
337 cout << p << "\n"; // (3,-4)
338
339 pt p{-3,2};
340 // p.x = 1; // doesnt compile
341 p = {1,2}; // correct
342
343 pt a{3,1}, b{1,-2};
344 a += 2.0*b; // a = (5,-3)
345 cout << a*b << " " << a/-b << "\n"; // (-1,-13)
346 (-2.2,-1.4)// typedef int T;
347 // bool eq(T a, T b){ return (a==b); }
348 typedef ld T; // or int
349 bool eq(T a, T b){ return abs(a - b) <= EPS; }

```

7.2 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;
9     esq = AE esta a esquerda de AB
10     T tmp = (b-a)^(e-a);
11     return (tmp > EPS) - (tmp < -EPS);
12 }
13
14 vector<point> convex_hull(vector<point> p) {
15     sort(p.begin(), p.end());
16
17     vector<point> L, U;
18
19     // Lower Hull
20     for(auto pp : p){
21         while(L.size() >= 2 && esq(L[L.size()-2], L.
22             back(), pp) == -1)
23             L.pop_back();
24         L.pb(pp);
25     }
26
27     reverse(all(p));
28     // Upper Hull
29     for(auto pp : p){

```

```
28         while(U.size() >= 2 && esq(U[U.size()-2], U. 33         L.pop_back();
    back(), pp) == -1) 34         L.insert(L.end(), U.begin(), U.end()-1);
29         U.pop_back(); 35
30         U.pb(pp); 36         return L;
31     } 37 }
```