

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	4
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



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 // O(alfa(n)) - log n
3 int get(int x) {
4     return p[x] = (p[x] == x ? x : get(p[x]));
5 }
6
7 void unite(int a, int b){
8     a = get(a);
9     b = get(b);
10
11     if(r[a] == r[b]) r[a]++;
12     if(r[a] > r[b]) p[b] = a;
13     else p[a] = b;
14 }
15

```

```

16 // Initializing values in main()
17 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 //      representa(inclusivo)
8 // str: limite superior do intervalo que aquele nodo
9 //      representa(inclusivo)
10 // l : limite inferior do intervalo que queremos
11 //     fazer a consulta
12 // r : limite superior do intervalo que queremos
13 //     fazer a consulta
14 // i : indice do vetor que queremos atualizar
15 // amm: novo valor daquele indice no vetor
16
17 class SegTree{
18     vector<int> st;
19     vector<int> lazy;
20     vector<bool> has;
21     int size;
22
23     int el_neutro = -(1e9 + 7);
24
25     int f(int a, int b){
26         return max(a,b);
27     }
28
29     void propagate(int sti, int stl, int str){
30         if(has[sti]){
31             st[sti] = lazy[sti]*(str-stl+1);
32         }
33     }
34 }

```

```

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

```

```

94 SegTree st(v.size());
95
96 for(int i = 0; i < n; i++){
97     st.update(i, v[i]);
98 }

```

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

```

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 = mid-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 O(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 |          | F0 |    | Fk |
65 */
66 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
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 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]);

```

```

45     return dp[W];
46 }

```

5.2 LCS

```

1 // LCS maior subs comum
2 // a,b = indice maximo do vetor
3 // s1={1,2,3} a = 2
4 #define MAXN 1010
5
6 int s1[MAXN], s2[MAXN], tab[MAXN][MAXN];
7
8 int lcs(int a, int b){
9
10    if(a == 0 || b == 0) return tab[a][b] = 0;
11
12    if(tab[a][b] != -1) return tab[a][b];
13
14    if(s1[a] == s2[b]) return lcs(a-1,b-1)+1;
15
16    return tab[a][b] = max(lcs(a-1, b), lcs(a, b-1));
17 }

```

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
15     return dp[w];

```