Competitive programming Notebook 🎈

Pablo Arruda Araujo

Sumário

1 Ds

1.1 sparse-table

```cpp
// Sparse-Table
// O(log n)
const int logn = 22; // max log

int logv[MAX];
// Pre comp log values
void make_log(){
    logv[1] = 0;
    for(int i = 2; i <= MAX; i++)
        logv[i] = logv[i/2]+1;
}

struct Sparse {
    vector<vector<int> > st;

    Sparse(vector<int>& v) {
        int n = v.size();
        st.assign(n, vector<int>(logn, 0));
        // Unitary values st[i][0] = v[i, i+2^0] = v[i]
        for(int i = 0; i < n; i++){
            st[i][0] = v[i];
        }
        // Constructing Sparse Table in O(log n)
        for(int k = 1; k < logn; k++){
            for(int i = 0; i < n; i++){
                if(i + (1 << k)-1 >= n)
                    continue;
                int prox = i + (1 << (k-1));
                st[i][k] = min(st[i][k-1], st[prox][k-1]);
            }
        }
    }

    int f(int a, int b){
        // Can be: min, max, gcd
        // f must have idempotent property
        return min(a, b);
    }
    // Queries in O(1)
    int query(int l, int r){
        int size = r-l+1;
        int k = logv[size];
        // cat jump for queries in O(1)
        int res = f(st[l][k], st[r - ((1 << k)-1)][k]);
        return res;
    }
};
```

1.2 DSU

```cpp
// Disjoint union set
// Operation ~ O(1)
int r[MAXN];
vector<int> qtd(MAXN, 1);

int get(int x) {
  return p[x] = (p[x] == x ? x : get(p[x]));
}

void unite(int a, int b){
  a = get(a);
  b = get(b);

  if(a == b) return;
```

```cpp
  if(r[a] == r[b]){
    p[a] = b;
    r[b]++;
    qtd[b]+=qtd[a];
  }else if(r[a] > r[b]){
    p[b] = a;
    qtd[a]+=qtd[b];
  }else{
    p[a] = b;
    qtd[b]+=qtd[a];
  }
}

// Initializing values in main()
for(int i = 1; i <= n; i++) p[i]=i;
```

1.3 prefix-sum-array

```cpp
// Preffix sum 1D
// O(n)
int v[MAXN];
int psum[MAXN];

int create_psum(){
    int acc = 0;
    for(int i = 0; i < v.size(); i++){
        acc+=v[i];
        psum[i] = acc;
    }
}

int query(int l, int r){
    return l == 0 ? psum[r] : psum[r]-psum[l-1];
}
```

1.4 delta-encoding

```cpp
// Delta encoding
// O (n)

for(int i = 0; i < queries; i++){
    int l, r, x;
    cin >> l >> r >> x;
    delta[l]+=x;
    delta[r+1]-=x;
}
int acc = 0;
for(int i = 0; i < v.size(); i++){
    acc+=delta[i];
    v[i]+=acc;
}
```

1.5 Segtree

```cpp
// Segtree MAX
// O(log n) operations

// DESCRIPTION:
// sti: id do nodo que estamos na segment tree
// stl: limite inferior do intervalo que aquele nodo
    representa(inclusivo)
// str: limite superior do intervalo que aquele nodo
    representa(inclusivo)
// l : limite inferior do intervalo que queremos
    fazer a consulta
// r : limite superior do intervalo que queremos
    fazer a consulta
// i : indice do vetor que queremos atualizar
// amm: novo valor daquele indice no vetor

class SegTree{
    vector<int> st;
```

```cpp
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          if(str < l || stl > r) return el_neutro;
41
42          if(stl >= l && str <= r)
43              return st[sti];
44
45          // intervalo parcialmente incluido em l-r
46          int mid = (stl+str)/2;
47
48          return f(query(2*sti+1, stl, mid, l, r),
        query(2*sti+2, mid+1, str, l, r));
49      }
50
51      void update(int sti, int stl, int str, int i, int
         amm){
52          if(stl == i && str == i){
53              st[sti] += amm;
54              return;
55          }
56
57          if(stl > i || str < i) return;
58
59          int mid = (stl+str)/2;
60
61          // Processo de atualizacao dos nos filhos
62          update(sti*2+1, stl, mid, i, amm);
63          update(sti*2+2, mid+1, str, i, amm);
64
65          st[sti] = f(st[sti*2+1], st[sti*2+2]);
66      }
67
68      void update_range(int sti, int stl, int str, int
        l, int r, int amm){
69          if(stl >= l && str <= r){
70              lazy[sti] = amm;
71              has[sti] = true;
72              propagate(sti, stl, str);
73              return;
74          }
75
76          if(stl > r || str < l) return;
77
78          int mid = (stl+str)/2;
79          update_range(sti*2+1, stl, mid, l, r, amm);
80          update_range(sti*2+2, mid+1, str, l, r, amm);
81
82          st[sti] = f(st[sti*2+1],st[sti*2+2]);
83      }
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
        -1,l,r);}
88          void update(int i, int amm){update(0,0,size
        -1,i,amm);}
89          void update_range(int l, int r, int amm){
        update_range(0,0,size-1,l,r,amm);}
90  };
91
92  // In main()
93
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

```cpp
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

```cpp
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 /* ------------------------*/
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 aplied to negative weights
6
7 using edge = tuple<int, int, int>;
8
9 vector<int> bellman_ford(int s, int N, const vector<
        edge>& edges){
10     const int oo { 1000000010 };
11
12     vector<int> dist(N + 1, oo);
13     dist[s] = 0;
14
15     for (int i = 1; i <= N - 1; i++)
16         for (auto [u, v, w] : edges)
17             if (dist[u] < oo and dist[v] > dist[u] +
        w){
18                 dist[v] = dist[u] + w;
19                 // pred[v]=u to find path
20             }
21
22     return dist;
23 }
24
25 // Identifying negative Cycle
26 bool has_negative_cycle(int s, int N, const vector<
        edge>& edges){
27     const int oo { 1000000010 };
28
29     vector<int> dist(N + 1, oo);
30     dist[s] = 0;
31
32     for (int i = 1; i <= N - 1; i++)
33         for (auto [u, v, w] : edges)
34             if (dist[u] < oo and dist[v] > dist[u] +
        w)
35                 dist[v] = dist[u] + w;
36
37     // If after all rounds, exists a better answer -
        Negative cycle found
38     for (auto [u, v, w] : edges)
39         if (dist[u] < oo and dist[v] > dist[u] + w)
40             return true;
```

2.6 DFS

```cpp
// DFS
// O(n+m)
vector<vector<int> > graph(MAX_NODES);
vector<bool> visited(MAX_NODES);

void dfs(int s){
    if(visited[s]) return;
    visited[s] = true;
    for(auto v : graph[s]){
        dfs(v);
    }
}
```

2.7 MCBM

```cpp
// Augmenting Path Algorithm for Max Cardinality
    Bipartite Matching
// O(V*E)

// Algorithm to find maximum matches between to set
// of nodes (bipartite graph)

vector<int> match, visited;

int aug(int u){
    if(visited[u]) return 0;
    visited[u]=1;

    for(auto v : g[u]){
        if(match[v]==-1||aug(match[v])){
            match[v]=u;
            return 1;
        }
    }
    return 0;
}

// Inside Main()
// Good to try - left v: [0,n-1], right: [n, m-1]
int MCBM=0;
match.assign(V, -1); // V = all vertices(left+right)
for(int i = 0; i < n; i++){ // n = size of left set
    visited.assign(n, 0);
    MCBM+=aug(i);
}
```

2.8 Bridge

```cpp
// Algorithm to get bridges in a graph

using edge = pair<int, int>;

const int MAX { 100010 };
int dfs_num[MAX], dfs_low[MAX];
vector<vector<int> > adj;

void dfs_bridge(int u, int p, int& next, vector<edge
    >& bridges){

    dfs_low[u] = dfs_num[u] = next++;

    for (auto v : adj[u])
        if (not dfs_num[v]) {

            dfs_bridge(v, u, next, bridges);
```

```cpp
            if (dfs_low[v] > dfs_num[u])
                bridges.emplace_back(u, v);

            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
        } else if (v != p)
            dfs_low[u] = min(dfs_low[u], dfs_num[v]);
}

vector<edge> bridges(int n){

    memset(dfs_num, 0, (n + 1)*sizeof(int));
    memset(dfs_low, 0, (n + 1)*sizeof(int));

    vector<edge> bridges;

    for (int u = 1, next = 1; u <= n; ++u)
        if (not dfs_num[u])
            dfs_bridge(u, u, next, bridges);

    return bridges;
}
```

2.9 Warshall

```cpp
// Floyd - Warshall
// O(n^3)
#define INF 1e9+10

int adj[MAXN][MAXN];
int distances[MAXN][MAXN];

void Warshall(int n, int start){
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == j) distances[i][j] = 0;
            else if (adj[i][j]) distances[i][j] = adj
    [i][j];
            else distances[i][j] = INF;
        }
    }
    for (int z = 1; z <= n; z++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                distances[i][j] = min(distances[i][j
    ],distances[i][z] + distances[z][j]);
            }
        }
    }
}
```

2.10 CycleDetection

```cpp
// Existency of Cycle in a Graph

// 1. Better to use when path is important
// O(V+E)
const int MAXN { 100010 };
vector<int> visited(MAXN, 0);
vector<vector<int> > g(MAXN);

bool dfs_cycle(int u){
    if(visited[u]) return false;

    visited[u] = true;

    for(auto v : g[u]){
        if(visited[v] && v != u) return true;
        if(dfs_cycle(v)) return true;
    }
    return false;
}
```

```
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    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
    multiplicar
25         vector<vl> res(r, vl(o.c, 0));
26
27         for(int i = 0; i < r; i++)
28             for(int j = 0; j < o.c; j++)
29                 for(int k = 0; k < c; k++)
30                     res[i][j] = (res[i][j] + m[i][k]*
    o.m[k][j]) % 1000000007;
31
32         return Matrix(res);
33     }
34
35     void printMatrix(){
36         for(int i = 0; i < r; i++)
37             for(int j = 0; j < c; j++)
38                 cout << m[i][j] << " \n"[j == (c-1)];
39     }
40 };
41
42 Matrix fexp(Matrix b, ll e, int n) {
43     if(e == 0) return Matrix(n, n, true); //
    identidade
44     Matrix res = fexp(b, e/2LL, n);
45     res = (res * res);
46     if(e%2) res = (res * b);
47
48     return res;
49 }
50
51 // Fibonacci Example 0 (log n)
52 /*  Fibonacci
53     |1 1|*|Fn   | = |Fn+1|
54     |1 0| |Fn-1|   |Fn  |
55
56     Generic
57     |a1 a2 ... an| ** K *  |Fn-1| = |Fk+n-1|
58     |1 0   ...  0|         |Fn-2|   |Fk+n-2|
59     |0 1 0 ...  0|         |Fn-3|   |Fk+n-3|
60         ...                 ...       ...
61     |0 0 0 ...1 0|         |F0  |   |Fk     |
62 */
63
64 int main() {
65     ll n;
66     cin >> n; // Fibonacci(n)
67
68     if(n == 0) {
69         cout << 0 << endl;
70         return 0;
71     }
72
73     vector<vl> m = {{1LL, 1LL}, {1LL, 0LL}};
74     vector<vl> b = {{1LL}, {0LL}};
75
76     Matrix mat = Matrix(m);
77     Matrix base = Matrix(b);
78
79     mat = fexp(mat, n-1, 2);
80     mat = mat*base;
81
82     cout << mat.m[0][0] << endl;
83
84
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]]+
   valor[i-1],dp[i-1][j]);
33             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   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,
       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 -
    i + 1);
11         while(0 <= i - k && i + k < n && s[i - k] ==
    s[i + k]) {
12             k++;
13         }
14         d1[i] = k--;
15         if(i + k > r) {
16             l = i - k;
17             r = i + k;
18         }
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],
    r - i + 1);
23         while(0 <= i - k - 1 && i + k < n && s[i - k
    - 1] == s[i + k]) {
24             k++;
25         }
26         d2[i] = k--;
27         if(i + k > r) {
28             l = i - k - 1;
29             r = i + k ;
30         }
31     }
32
33     // special vector to construct query by interval
34     vector<int> res(2*n-1);
35     for (int i = 0; i < n; i++) res[2*i] = 2*d1[i]-1;
36     for (int i = 0; i < n-1; i++) res[2*i+1] = 2*d2[i
    +1];
37     return res;
38
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
       string s.
6
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             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
    +1][i+k-1]);
```

6.6   Kmp

7   Geometry

7.1   2D

```
1  // 2D structures template
2
3  // Code from - Github: Tiagosf00/Competitive-
       Programming !!
4  // Writer: Tiago de Souza Fernandes
5
6  #define EPS 1e-6
7  #define PI acos(-1)
8  #define vp vector<point>
9
10 // typedef int cod;
11 // bool eq(cod a, cod b){ return (a==b); }
12 typedef ld cod;
13 bool eq(cod a, cod b){ return abs(a - b) <= EPS; }
14
15 struct point{
16     cod x, y;
17     int id;
18     point(cod x=0, cod y=0): x(x), y(y){}
19
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
49 ld norm(point a){ // Modulo
50     return sqrt(a*a);
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=
       collinear; 1=esq;
57     cod tmp = (b-a)^(e-a); // from a to b
58     return (tmp > EPS) - (tmp < -EPS);
59     // if int: tira comentario
60     // if(tmp==0) return 0;
61     // if(tmp>0) return 1;
62     // return -1;
63 }
64 point rotccw(point p, ld a){
65     // a = PI*a/180; // graus
66     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
       +p.x*sin(a)));
67 }
68 point rot90cw(point a) { return point(a.y, -a.x); };
69 point rot90ccw(point a) { return point(-a.y, a.x); };
70
71 ld proj(point a, point b){ // a sobre b
72     return a*b/norm(b);
73 }
74 ld angle(point a, point b){ // em radianos
75     ld ang = a*b / norm(a) / norm(b);
76     return acos(max(min(ang, (ld)1), (ld)-1));
77 }
78 ld angle_vec(point v){
79     // return 180/PI*atan2(v.x, v.y); // graus
80     return atan2(v.x, v.y);
81 }
82 ld order_angle(point a, point b){ // from a to b ccw
       (a in front of b)
83     ld aux = angle(a,b)*180/PI;
84     return ((a^b)<=0 ? aux:360-aux);
85 }
86 bool angle_less(point a1, point b1, point a2, point
       b2){ // ang(a1,b1) <= ang(a2,b2)
87     point p1((a1*b1), abs((a1^b1)));
88     point p2((a2*b2), abs((a2^b2)));
89     return (p1^p2) <= 0;
90 }
91
92 ld area(vp &p){ // (points sorted)
93     ld ret = 0;
94     for(int i=2;i<(int)p.size();i++)
95         ret += (p[i]-p[0])^(p[i-1]-p[0]);
96     return abs(ret/2);
97 }
98 ld areaT(point &a, point &b, point &c){
99     return abs((b-a)^(c-a))/2.0;
100 }
101
102 point center(vp &A){
103     point c = point();
104     int len = A.size();
105     for(int i=0;i<len;i++)
106         c=c+A[i];
```

```
107     return c/len;
108 }
109
110 point forca_mod(point p, ld m){
111     ld cm = norm(p);
112     if(cm<EPS) return point();
113     return point(p.x*m/cm,p.y*m/cm);
114 }
115
116
117 ////////////
118 // Line //
119 ////////////
120
121 struct line{
122     point p1, p2;
123     cod a, b, c; // ax+by+c = 0;
124     // 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         // Gera os pontos p1 p2 dados os coeficientes
132         // isso aqui eh horrivel mas quebra um galho
       kkkkkk
133         if(b==0){
134             p1 = point(1, -c/a);
135             p1 = point(0, -c/a);
136         }else{
137             p1 = point(1, (-c-a*1)/b);
138             p2 = point(0, -c/b);
139         }
140     }
141
142     cod eval(point p){
143         return a*p.x+b*p.y+c;
144     }
145     bool inside(point p){
146         return eq(eval(p), 0);
147     }
148     point normal(){
149         return point(a, b);
150     }
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.
       x, p2.x) and
155                 min(p1.y, p2.y)<=p.y and p.y<=max(p1.
       y, p2.y));
156     }
157
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
168 point inter_seg(line l1, line l2){
169     point ans = inter_line(l1, l2);
170     if(ans.x==INF or !l1.inside_seg(ans) or !l2.
       inside_seg(ans))
171         return point(INF, INF);
172     return ans;
173 }
174
```

```
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(l.a*l.a + l.b*l.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
         through p
191      return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
192  }
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
         cc){
208          c = inter_line(mediatrix(a, b), mediatrix(b,
             cc));
209          r = norm(a-c);
210      }
211      bool inside(const point &a) const{
212          return norm(a - c) <= r;
213      }
214      pair<point, point> getTangentPoint(point p) {
215          ld d1 = norm(p-c), theta = asin(r/d1);
216          point p1 = rotccw(c-p,-theta);
217          point p2 = rotccw(c-p,theta);
218          p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
219          p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
220          return {p1,p2};
221      }
222  };
223
224  // minimum circle cover O(n) amortizado
225  circle min_circle_cover(vector<point> v){
226      random_shuffle(v.begin(), v.end());
227      circle ans;
228      int n = v.size();
229      for(int i=0;i<n;i++) if(!ans.inside(v[i])){
230          ans = circle(v[i]);
231          for(int j=0;j<i;j++) if(!ans.inside(v[j])){
232              ans = circle(v[i], v[j]);
233              for(int k=0;k<j;k++) if(!ans.inside(v[k])
                 ){
234                  ans = circle(v[i], v[j], v[k]);
235              }
236          }
237      }
238      return ans;
239  }
240
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^n)/(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.
         p1)*(ab) / (ab*ab));
265      ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
          / (ab*ab);
266      if (h2 < 0) return {};
267      if (h2 == 0) return {p};
268      point h = (ab/norm(ab)) * sqrt(h2);
269      return {p - h, p + h};
270  }
271
272  vp inter_circle(circle C1, circle C2){
273      if(C1.c == C2.c) { assert(C1.r != C2.r); return
         {}; }
274      point vec = C2.c - C1.c;
275      ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
         ;
276      ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
         C1.r*C1.r - p*p*d2;
277      if (sum*sum < d2 or dif*dif > d2) return {};
278      point mid = C1.c + vec*p, per = point(-vec.y, vec
         .x) * sqrt(max((ld)0, h2) / d2);
279      if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
280      return {mid + per, mid - per};
281  }
```

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
        || colinear)
6
7   int esq(point p1, point p2, point p3){
8       cod cross = (p2-p1)^(p3-p1);
9       if(cross == 0) return 0;
10      else if(cross > 0) return 1;
11      return -1;
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.
             back(), pp) == -1)
22              L.pop_back();
23          L.pb(pp);
24      }
```

```
25
26        reverse(all(p));
27        // Upper Hull
28        for(auto pp : p){
29            while(U.size() >= 2 && esq(U[U.size()-2], U.
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 }
```