Competitive programming Notebook 🎈

Pablo Arruda Araujo

Sumário

1 Ds

1.1 sparse-table

```
// 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

```
// 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;

  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

```
// 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

```
// 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

```
// 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
    vector<int> lazy;
    vector<bool> has;
    int size;

    int el_neutro = -(1e9 + 7);

    int f(int a, int b){
        return max(a,b);
    }

    void propagate(int sti, int stl, int str){
        if(has[sti]){
            st[sti] = lazy[sti]*(str-stl+1);
            if(stl!=str){
                lazy[sti*2+1] = lazy[sti];
                lazy[sti*2+2] = lazy[sti];

                has[sti*2+1] = true;
                has[sti*2+2] = true;
            }
            has[sti] = false;
        }
    }

    int query(int sti, int stl, int str, int l, int r
)){
        if(str < l || stl > r) return el_neutro;

        if(stl >= l && str <= r)
            return st[sti];

        // intervalo parcialmente incluido em l-r
        int mid = (stl+str)/2;

        return f(query(2*sti+1, stl, mid, l, r),
query(2*sti+2, mid+1, str, l, r));
    }

    void update(int sti, int stl, int str, int i, int
 amm){
        if(stl == i && str == i){
            st[sti] += amm;
            return;
        }

        if(stl > i || str < i) return;

        int mid = (stl+str)/2;

        // Processo de atualizacao dos nos filhos
        update(sti*2+1, stl, mid, i, amm);
        update(sti*2+2, mid+1, str, i, amm);

        st[sti] = f(st[sti*2+1], st[sti*2+2]);
    }

    void update_range(int sti, int stl, int str, int
l, int r, int amm){
        if(stl >= l && str <= r){
            lazy[sti] = amm;
            has[sti] = true;
            propagate(sti, stl, str);
            return;
        }

        if(stl > r || str < l) return;

        int mid = (stl+str)/2;
        update_range(sti*2+1, stl, mid, l, r, amm);
        update_range(sti*2+2, mid+1, str, l, r, amm);

        st[sti] = f(st[sti*2+1],st[sti*2+2]);
    }

    public:
        SegTree(int n): st(4*n, 0){size=n;}
        int query(int l, int r){return query(0,0,size
-1,l,r);}
        void update(int i, int amm){update(0,0,size
-1,i,amm);}
        void update_range(int l, int r, int amm){
update_range(0,0,size-1,l,r,amm);}
};

// In main()

SegTree st(v.size());

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

2 Graph

2.1 Dijkstra

```cpp
// Dijkstra
// O(V + E log E)
#define INF 1e9+10
vector<pair<int, int>> adj[MAXN];
vector<int> dist;
vector<bool> visited;
priority_queue<pair<int,int>> q;

void Dijkstra(int n, int start){
    for(int i = 0; i <= n; i++){
        dist.push_back(INF);
        visited.push_back(false);
    }
    dist[start] = 0;
    q.push(make_pair(0, start));
    while(!q.empty()){
        int a = q.top().second; q.pop();
        if(visited[a]) continue;
        visited[a] = true;
        for(auto u : adj[a]){
            int b = u.first, w = u.second;
            if(dist[a]+w < dist[b]){
                dist[b] = dist[a]+w;
                q.push({-dist[b], b});
            }
        }
    }
}
```

2.2 Bipartite

```cpp
// Checking if graph is Bipartite
// O(V+E)

const int MAXN { 100010 };
vector<vector<int> > g(MAXN);
vector<int> color(MAXN);

bool bfs(int s){
    const int NONE=0,B=1,W=2;
    queue<int> q;
    q.push(s);
    color[s]=B;

    while(!q.empty()){
        auto u = q.front(); q.pop();

        for(auto v : g[u]){
            if(color[v] == NONE){
                color[v]=3-color[u];
                q.push(v);
            }else if(color[v]==color[u]){
                return false;
            }
        }

        return true;
    }
}

bool is_bipartite(int n){

    for (int u = 1; u <= n; u++)
        if (color[u] == NONE && !bfs(u))
            return false;

    return true;
}
```

2.3 BFS

```
// BFS
// O(V+E)

const int MAXN { 100010 };

vector<vector<int> > g(MAXN);
vector<bool> visited(MAXN);
vector<int> dist(MAXN, oo);
queue<int> q;

void bfs(int s){
    q.push(s);
    dist[s] = 0;
    visited[s] = true;

    while(!q.empty()){
        int u = q.front(); q.pop();

        for(auto v : g[u]){
            if(not visited[v]){
                dist[v] = dist[u]+1;
                visited[v] = true;
                q.push(v);
            }
        }
    }
}
```

2.4 BellmanFord

```
// Bellman Ford - Min distance

// O(V*E)
// Min dist from a start node
// Can be aplied to negative weights

using edge = tuple<int, int, int>;

vector<int> bellman_ford(int s, int N, const vector<
    edge>& edges){
    const int oo { 1000000010 };

    vector<int> dist(N + 1, oo);
    dist[s] = 0;

    for (int i = 1; i <= N - 1; i++)
        for (auto [u, v, w] : edges)
            if (dist[u] < oo and dist[v] > dist[u] +
    w){
                dist[v] = dist[u] + w;
                // pred[v]=u to find path
            }

    return dist;
}

// Identifying negative Cycle
bool has_negative_cycle(int s, int N, const vector<
    edge>& edges){
    const int oo { 1000000010 };

    vector<int> dist(N + 1, oo);
    dist[s] = 0;

    for (int i = 1; i <= N - 1; i++)
        for (auto [u, v, w] : edges)
            if (dist[u] < oo and dist[v] > dist[u] +
    w)
                dist[v] = dist[u] + w;

    // If after all rounds, exists a better answer -
    Negative cycle found
    for (auto [u, v, w] : edges)
        if (dist[u] < oo and dist[v] > dist[u] + w)
            return true;

    return false;
}
```

2.5 DFS

```
// 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);
    }
}
```

2.6 MCBM

```
// 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;
```

```cpp
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);
}
```

2.7 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);

            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;
}
```

2.8 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]);
            }
        }
    }
}
```

2.9 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;
}

bool has_cycle(int n){
    visited.reset();

    for(int u = 1; u <= n; u++)
        if(!visited[u] && dfs(u))
            return true;

    return false;
}

// 2. Better when only detect cycle is important
// Only for undirected graphs
// When E>=V, a cycle exists

void dfs(int u, function<void(int)> process){
    if (visited[u])
        return;

    visited[u] = true;

    process(u);

    for (auto v : adj[u])
        dfs(v, process);
}

bool has_cycle(int N) {
    visited.reset();

    for (int u = 1; u <= N; ++u)
        if (not visited[u])
        {
            vector<int> cs;
            size_t edges = 0;

            dfs(u, [&](int u) {
                cs.push_back(u);

                for (const auto& v : adj[u])
                    edges += (visited[v] ? 0 : 1);
            });

            if (edges >= cs.size()) return true;
        }

    return false;
}
```

2.10 MST

```
// Minimum Spanning tree
// w/ DSU structure

typedef struct{
    int a, b;
    int w;
} edge;

/* ----- DSU Structure ------*/
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(r[a] == r[b]) r[a]++;
  if(r[a] > r[b]) p[b] = a;
  else p[a] = b;
}

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

/* -------------------------*/

vector<edge> edges;
int total_weight;

void mst(){
    // sort edges
    for(auto e : edges){
        if(get(e.a) != get(e.b)){
            unite(e.a, e.b);
            total_weight+=e.w;
        }
    }
}
```

3 Algorithm

3.1 merge-sort

```
// Merge Sort
// O(n log n)
void merge_sort(vector<int>& v){
    if(v.size() == 1) return;

    vector<int> l, r;

    for(int i = 0; i < v.size()/2; i++)
        l.push_back(v[i]);
    for(int i = v.size()/2; i < v.size(); i++)
        r.push_back(v[i]);

    merge_sort(l);
    merge_sort(r);

    l.push_back(INF);
    r.push_back(INF);

    int inil = 0, inir = 0;

    for(int i = 0; i < v.size(); i++){
        if(l[inil] < r[inir]) v[i] = l[inil++];
        else v[i] = r[inir++];
    }

    return;
}
```

3.2 bsearch-iterative

```
// Binary search in iterative questions
// O(log n)
bool query(int mid, int x){
    cout << mid << endl;
    cout.flush();

    int ans;
    cin >> ans;
    return ans == x;
}

int solve(int x){
    int l = 1, r = n;
    int res = -1;

    while(l <= r){
        int mid = (l+r)/2;
        if(query(mid, x)){
            res = mid;
            l = mid+1;
        }else{
            r = m-1;
        }
    }

    return res;
}
```

3.3 counting-inversions

```
// Counting inversions in Array
// O(n log n)
int merge_sort(vector<int>& v){
    if(v.size() == 1) return 0;

    vector<int> l, r;

    for(int i = 0; i < v.size()/2; i++)
        l.push_back(v[i]);
    for(int i = v.size()/2; i < v.size(); i++)
        r.push_back(v[i]);
    int ans = 0;
    ans += merge_sort(l);
    ans += merge_sort(r);

    l.push_back(1e9);
    r.push_back(1e9);

    int inil = 0, inir = 0;

    for(int i = 0; i < v.size(); i++){
        if(l[inil] <= r[inir]) v[i] = l[inil++];
        else{
            v[i] = r[inir++];
            ans+=l.size()-inil-1;
        }
    }

    return ans;
}
```

3.4 kadane

```
// Maximum possible sum in Array
// O(n)
int array[MAXN];

```

```cpp
int kadane(){
    int sum = 0, best = 0;
    for(int i = 0; i < n; i++){
        sum = max(array[i], sum+array[i]);
        best = max(sum, best);
    }

    return best;
}
```

4 Math

4.1 floor-log

```cpp
// Find floor(log(x))
// O(n)
int logv[MAXN];
void make_log(){
    logv[1] = 0;
    for(int i = 2; i <= MAXN; i++)
        logv[i] = logv[i/2]+1;
}
```

4.2 fast-exponentiation

```cpp
// Fast Exponentiation
// O(log n)
ll fexp(ll b, ll e){
    if(e == 0){
        return 1;
    }
    ll resp = fexp(b, e/2)%MOD;
    resp = (resp*resp)%MOD;
    if(e%2) resp = (b*resp)%MOD;

    return resp;
}
```

4.3 matrix-exponentiation

```cpp
// Matrix Exponentiation
// O(log n)
#define ll long long int
#define vl vector<ll>
struct Matrix {
    vector<vl> m;
    int r, c;

    Matrix(vector<vl> mat) {
        m = mat;
        r = mat.size();
        c = mat[0].size();
    }

    Matrix(int row, int col, bool ident=false) {
        r = row; c = col;
        m = vector<vl>(r, vl(c, 0));
        if(ident)
            for(int i = 0; i < min(r, c); i++)
                m[i][i] = 1;
    }

    Matrix operator*(const Matrix &o) const {
        assert(c == o.r); // garantir que da pra multiplicar
        vector<vl> res(r, vl(o.c, 0));

        for(int i = 0; i < r; i++)
            for(int j = 0; j < o.c; j++)
                for(int k = 0; k < c; k++)
                    res[i][j] = (res[i][j] + m[i][k]*o.m[k][j]) % 1000000007;

        return Matrix(res);
    }

    void printMatrix(){
        for(int i = 0; i < r; i++)
            for(int j = 0; j < c; j++)
                cout << m[i][j] << " \n"[j == (c-1)];
    }
};

Matrix fexp(Matrix b, ll e, int n) {
    if(e == 0) return Matrix(n, n, true); // identidade
    Matrix res = fexp(b, e/2LL, n);
    res = (res * res);
    if(e%2) res = (res * b);

    return res;
}

// Fibonacci Example O(log n)
/*  Fibonacci
    |1 1|*|Fn  | = |Fn+1|
    |1 0| |Fn-1|   |Fn  |

    Generic
    |a1 a2 ... an| ** K *  |Fn-1| = |Fk+n-1|
    |1 0   ...  0|         |Fn-2|   |Fk+n-2|
    |0 1 0 ...  0|         |Fn-3|   |Fk+n-3|
           ...              ...      ...
    |0 0 0 ...1 0|         |F0  |   |Fk    |
*/

int main() {
    ll n;
    cin >> n; // Fibonacci(n)

    if(n == 0) {
        cout << 0 << endl;
        return 0;
    }

    vector<vl> m = {{1LL, 1LL}, {1LL, 0LL}};
    vector<vl> b = {{1LL}, {0LL}};

    Matrix mat = Matrix(m);
    Matrix base = Matrix(b);

    mat = fexp(mat, n-1, 2);
    mat = mat*base;

    cout << mat.m[0][0] << endl;


    return 0;
}
```

5 Dp

5.1 knapsack

```cpp
// Knapsack problem
// O(n.w)
int valor[MAXN], peso[MAXN], memo[MAXN];

ll solve(int i, int w){ // Recursive version
    if(i <= 0 || w <= 0) return 0;
    if(memo[i][w] != -1) return memo[i][w];
    ll pegar=-1e9;

    if(peso[i] <= w){
        pegar = solve(i-1,w-peso[i])+valor[i];
    }

    ll naopegar = solve(i-1,w);

    memo[i][w] = max(pegar,naopegar);

    return memo[i][w];
}

int dp[MAXN][MAXN], valor[MAXN], peso[MAXN];
int solve(int n, w){  // Iterative version
// n objects | max weight
    for(int i = 0; i <= n; i++)
        for(int j=0; j <= w;j++)
            dp[i][j] = 0;

    for(int i = 0; i <= n; i++){
        for(int j = 0; j <= w; j++){
            if(i == 0 || j == 0) return dp[i][j];
            else if(peso[i-1] <= j)
                dp[i][j] = max(dp[i-1][j-peso[i-1]]+
    valor[i-1],dp[i-1][j]);
            else
                dp[i][j] = dp[i-1][j];
        }
    }
    return dp[n][w];
}

int val[MAX], wt[MAX], dp[MAX]; // Optimization for
    space
int solve(int n, int W){
    for(int i=0; i < n; i++)
        for(int j=W; j>=wt[i]; j--)
            dp[j] = max(dp[j],dp[j-wt[i]]+val[i]);
    return dp[W];
}
```

5.2 LCS

```
// LCS  maior subs comum
// ** usar s[1 - n]
#define MAXN 1010

int s1[MAXN], s2[MAXN], tab[MAXN][MAXN];

int lcs(int a, int b){

    if(a == 0 || b == 0) return tab[a][b] = 0;

    if(tab[a][b] != -1) return tab[a][b];

    if(s1[a] == s2[b]) return lcs(a-1,b-1)+1;

    return tab[a][b] = max(lcs(a-1, b), lcs(a, b-1));
}
```

5.3 coin-change

```
// You have n coins {c1, ..., cn}
// Find min quantity of coins to sum K
// O(n.c)
int dp(int acc){ // Recursive version
    if(acc < 0) return oo;
    if(acc == 0) return 0;

    if(memo[acc] != -1) return memo[acc];

    int best = oo;

    for(auto c : coins){
        best = min(best, dp(acc-c)+1);
    }

    return memo[acc] = best;
}

int dp(){ // Iterative version
    memo[0] = 0
    for(int i = 1; i <= n; i++){
        memo[i] = oo;
        for(auto c : coins){
            if(i-c >= 0)
                memo[i] = min(memo[i], memo[i-c]+1);
        }
    }
}
```

5.4 unbouded-knapsack

```
// Knapsack (unlimited objects)
// O(n.w)

int w, n;
int c[MAXN], v[MAXN], dp[MAXN];

int unbounded_knapsack(){

    for(int i=0;i<=w;i++)
        for(int j=0;j<n;j++)
            if(c[j] <= i)
                dp[i] = max(dp[i], dp[i-c[j]] + v[j])
    ;

    return dp[w];
}
```

6 String

6.1 General

```
// General functions to manipulate strings

// find function
int i = str.find("aa");
i = pos ou -1

// find multiples strings
while(i!=string::npos){
    i = str.find("aa", i);
}

// replace function
str.replace(index, (int)size_of_erased, "content");
"paablo".replace(1, 2, "a"); // = Pablo

// string concatenation
string a = "pabl"
a+="o" or a+='o' or a.pb('o')
```

6.2 Manacher

```
// Manacher Algorithm
// O(n)

// Find all sub palindromes in a string
// d1 = Odd palin, d2 = Even palin

vector<int> manacher(string &s, vector<int> &d1,
    vector<int> &d2) {
```

```
    int n = s.size();
    for(int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
        while(0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
            k++;
        }
        d1[i] = k--;
        if(i + k > r) {
            l = i - k;
            r = i + k;
        }
    }

    for(int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while(0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
            k++;
        }
        d2[i] = k--;
        if(i + k > r) {
            l = i - k - 1;
            r = i + k ;
        }
    }

    // special vector to construct query by interval
    vector<int> res(2*n-1);
    for (int i = 0; i < n; i++) res[2*i] = 2*d1[i]-1;
    for (int i = 0; i < n-1; i++) res[2*i+1] = 2*d2[i+1];
    return res;

}

struct palindrome {
    vector<int> res;

    palindrome(const& s): res(manacher(s)){}

    // Query if [i..j] is palindrome
    bool is_palindrome(int i, int j){
        return res[i+j] >= j-i+1;
    }
}
```

6.3 Z-function

```
// Z-function
// O(n)

// Return array z(n) that each value z[i] tells the
// longest subsequence from i that is prefix of
    string s.

// Pattern Matching = z-func(s1$s2) acha s1 em s2.

vector<ll> z_algo(const string &s){
    ll n = s.size();
    ll L = 0, R = 0;
    vector<ll> z(n, 0);
    for(ll i = 1; i < n; i++){
        if(i <= R)
            z[i] = min(z[i-L], R - i + 1);
        while(z[i]+i < n and s[ z[i]+i ] == s[ z[i] ])
            z[i]++;
        if(i+z[i]-1 > R){
            L = i;
            R = i + z[i] - 1;
        }
    }
    z[0]=n;
    return z;
}
```

6.4 Trie

6.5 AllSubPalindromes

```
// Function to find all Sub palindromes
// O(n*n)

string s; // n = s.size();
vector<vector<bool> > is_pal(n, vector<bool>(n, true)
    );

// formando todos os subpalindromos
forne(k, 1, n-1)
    forne(i, 0, n-k-1)
        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

```
// 2D structures template

// Code from - Github: Tiagosf00/Competitive-
    Programming !!
// Writer: Tiago de Souza Fernandes

#define EPS 1e-6
#define PI acos(-1)
#define vp vector<point>

// typedef int cod;
// bool eq(cod a, cod b){ return (a==b); }
typedef ld cod;
bool eq(cod a, cod b){ return abs(a - b) <= EPS; }

struct point{
    cod x, y;
    int id;
    point(cod x=0, cod y=0): x(x), y(y){}


    point operator+(const point &o) const{
        return {x+o.x, y+o.y};
    }
    point operator-(const point &o) const{
        return {x-o.x, y-o.y};
    }
    point operator*(cod t) const{
        return {x*t, y*t};
    }
    point operator/(cod t) const{
        return {x/t, y/t};
    }
    cod operator*(const point &o) const{ // dot
        return x * o.x + y * o.y;
    }
    cod operator^(const point &o) const{ // cross
        return x * o.y - y * o.x;
    }
    bool operator<(const point &o) const{
        if(!eq(x, o.x)) return x < o.x;
```

```cpp
        return y < o.y;
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y);
    }

};

ld norm(point a){ // Modulo
    return sqrt(a*a);
}
bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0));
}

int ccw(point a, point b, point e){ //-1=dir; 0=
    collinear; 1=esq;
    cod tmp = (b-a)^(e-a); // from a to b
    return (tmp > EPS) - (tmp < -EPS);
    // if int: tira comentario
    // if(tmp==0) return 0;
    // if(tmp>0) return 1;
    // return -1;
}
point rotccw(point p, ld a){
    // a = PI*a/180; // graus
    return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
    +p.x*sin(a)));
}
point rot90cw(point a) { return point(a.y, -a.x); };
point rot90ccw(point a) { return point(-a.y, a.x); };

ld proj(point a, point b){ // a sobre b
    return a*b/norm(b);
}
ld angle(point a, point b){ // em radianos
    ld ang = a*b / norm(a) / norm(b);
    return acos(max(min(ang, (ld)1), (ld)-1));
}
ld angle_vec(point v){
    // return 180/PI*atan2(v.x, v.y); // graus
    return atan2(v.x, v.y);
}
ld order_angle(point a, point b){ // from a to b ccw
    (a in front of b)
    ld aux = angle(a,b)*180/PI;
    return ((a^b)<=0 ? aux:360-aux);
}
bool angle_less(point a1, point b1, point a2, point
    b2){ // ang(a1,b1) <= ang(a2,b2)
    point p1((a1*b1), abs((a1^b1)));
    point p2((a2*b2), abs((a2^b2)));
    return (p1^p2) <= 0;
}

ld area(vp &p){ // (points sorted)
    ld ret = 0;
    for(int i=2;i<(int)p.size();i++)
        ret += (p[i]-p[0])^(p[i-1]-p[0]);
    return abs(ret/2);
}
ld areaT(point &a, point &b, point &c){
    return abs((b-a)^(c-a))/2.0;
}

point center(vp &A){
    point c = point();
    int len = A.size();
    for(int i=0;i<len;i++)
        c=c+A[i];
    return c/len;
}


point forca_mod(point p, ld m){
    ld cm = norm(p);
    if(cm<EPS) return point();
    return point(p.x*m/cm,p.y*m/cm);
}


///////////
//  Line  //
///////////

struct line{
    point p1, p2;
    cod a, b, c; // ax+by+c = 0;
    // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
    line(point p1=0, point p2=0): p1(p1), p2(p2){
        a = p1.y-p2.y;
        b = p2.x-p1.x;
        c = -(a*p1.x + b*p1.y);
    }
    line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
    {
        // Gera os pontos p1 p2 dados os coeficientes
        // isso aqui eh horrivel mas quebra um galho
    kkkkkk
        if(b==0){
            p1 = point(1, -c/a);
            p1 = point(0, -c/a);
        }else{
            p1 = point(1, (-c-a*1)/b);
            p2 = point(0, -c/b);
        }
    }

    cod eval(point p){
        return a*p.x+b*p.y+c;
    }
    bool inside(point p){
        return eq(eval(p), 0);
    }
    point normal(){
        return point(a, b);
    }

    bool inside_seg(point p){
        return (inside(p) and
                min(p1.x, p2.x)<=p.x and p.x<=max(p1.
    x, p2.x) and
                min(p1.y, p2.y)<=p.y and p.y<=max(p1.
    y, p2.y));
    }

};

vp inter_line(line l1, line l2){
    ld det = l1.a*l2.b - l1.b*l2.a;
    if(det==0) return {};
    ld x = (l1.b*l2.c - l1.c*l2.b)/det;
    ld y = (l1.c*l2.a - l1.a*l2.c)/det;
    return {point(x, y)};
}

point inter_seg(line l1, line l2){
    point ans = inter_line(l1, l2);
    if(ans.x==INF or !l1.inside_seg(ans) or !l2.
    inside_seg(ans))
        return point(INF, INF);
    return ans;
}

ld dseg(point p, point a, point b){ // point - seg
    if(((p-a)*(b-a)) < EPS) return norm(p-a);
    if(((p-b)*(a-b)) < EPS) return norm(p-b);
    return abs((p-a)^(b-a))/norm(b-a);
}

ld dline(point p, line l){ // point - line
    return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
}

line mediatrix(point a, point b){
    point d = (b-a)*2;
    return line(d.x, d.y, a*a - b*b);
}

line perpendicular(line l, point p){ // passes through p
    return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
}


////////////
// Circle //
////////////

struct circle{
    point c; cod r;
    circle() : c(0, 0), r(0){}
    circle(const point o) : c(o), r(0){}
    circle(const point a, const point b){
        c = (a+b)/2;
        r = norm(a-c);
    }
    circle(const point a, const point b, const point cc){
        c = inter_line(mediatrix(a, b), mediatrix(b, cc));
        r = norm(a-c);
    }
    bool inside(const point &a) const{
        return norm(a - c) <= r;
    }
    pair<point, point> getTangentPoint(point p) {
        ld d1 = norm(p-c), theta = asin(r/d1);
        point p1 = rotccw(c-p,-theta);
        point p2 = rotccw(c-p,theta);
        p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
        p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
        return {p1,p2};
    }
};

// minimum circle cover O(n) amortizado
circle min_circle_cover(vector<point> v){
    random_shuffle(v.begin(), v.end());
    circle ans;
    int n = v.size();
    for(int i=0;i<n;i++) if(!ans.inside(v[i])){
        ans = circle(v[i]);
        for(int j=0;j<i;j++) if(!ans.inside(v[j])){
            ans = circle(v[i], v[j]);
            for(int k=0;k<j;k++) if(!ans.inside(v[k])){
                ans = circle(v[i], v[j], v[k]);
            }
        }
    }
    return ans;
}


circle incircle( point p1, point p2, point p3 ){
    ld m1=norm(p2-p3);
    ld m2=norm(p1-p3);
    ld m3=norm(p1-p2);
    point c = (p1*m1+p2*m2+p3*m3)*(1/(m1+m2+m3));
    ld s = 0.5*(m1+m2+m3);
    ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3))/s;
    return circle(c, r);
}

circle circumcircle(point a, point b, point c) {
    circle ans;
    point u = point((b-a).y, -(b-a).x);
    point v = point((c-a).y, -(c-a).x);
    point n = (c-b)*0.5;
    ld t = (u^n)/(v^u);
    ans.c = ((a+c)*0.5) + (v*t);
    ans.r = norm(ans.c-a);
    return ans;
}

vp inter_circle_line(circle C, line L){
    point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.p1)*(ab) / (ab*ab));
    ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s / (ab*ab);
    if (h2 < 0) return {};
    if (h2 == 0) return {p};
    point h = (ab/norm(ab)) * sqrt(h2);
    return {p - h, p + h};
}

vp inter_circle(circle C1, circle C2){
    if(C1.c == C2.c) { assert(C1.r != C2.r); return {}; }
    point vec = C2.c - C1.c;
    ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r;
    ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 = C1.r*C1.r - p*p*d2;
    if (sum*sum < d2 or dif*dif > d2) return {};
    point mid = C1.c + vec*p, per = point(-vec.y, vec.x) * sqrt(max((ld)0, h2) / d2);
    if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
    return {mid + per, mid - per};
}
```

7.2 ConvexHull

```
// Convex Hull
// Algorithm: Monotone Chain
// Complexity: O(n) + ordenacao O(nlogn)

// Regra mao direita p2->p1 (dedao p cima ? esq : dir || colinear)

int esq(point p1, point p2, point p3){
    cod cross = (p2-p1)^(p3-p1);
    if(cross == 0) return 0;
    else if(cross > 0) return 1;
    return -1;
}

vector<point> convex_hull(vector<point> p) {
    sort(p.begin(), p.end());

    vector<point> L, U;

    // Lower Hull
    for(auto pp : p){
        while(L.size() >= 2 && esq(L[L.size()-2], L.back(), pp) == -1)
            L.pop_back();
        L.pb(pp);
    }

    reverse(all(p));
    // Upper Hull
    for(auto pp : p){
        while(U.size() >= 2 && esq(U[U.size()-2], U.back(), pp) == -1)
            U.pop_back();
        U.pb(pp);
    }

    L.pop_back();
    L.insert(L.end(), U.begin(), U.end()-1);

    return L;
}
```