# SamuellH12 - ICPC Library

## Contents

# 1 Data Structures

## 1.1 BIT - Binary Indexed Tree

```cpp
#include <bits/stdc++.h>
#define optimize ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL)
#define ALL(x) x.begin(), x.end()
#define endl "\n"
#define ll long long
#define vi vector<int>
#define pii pair<int,int>
#define INF 0x3f3f3f3f
const int MAXN = 1e6 + 5;
using namespace std;

struct BIT {
	int bit[MAXN];

	void update(int pos, int val){
		for(; pos < MAXN; pos += pos&(-pos))
			bit[pos] += val;
	}

	int query(int pos){
		int sum = 0;
		for(; pos > 0; pos -= pos&(-pos))
			sum += bit[pos];
		return sum;
	}

	void init(){
		memset(bit, 0, sizeof bit);
	}

} Bit;


int main(){
	cout << "Binary Indexed Tree - Fenwick Tree" << endl;
	return 0;
}

/*

Syntax:

Bit.init();                          //Seta tudo como 0
Bit.update(i, x);                //Adiciona +x na posi  o i da BIT
Bit.update(2, 5);
cout << Bit.query(i) << endl; //Retorna o somat rio de [0, i]

Query: O(log n)
Update: O(log n)

*/
```

## 1.2 BIT2D

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;

struct BIT2D {
	int bit[MAXN][MAXN];

	void update(int X, int Y, int val){
		for(int x = X; x < MAXN; x += x&(-x))
			for(int y = Y; y < MAXN; y += y&(-y))
				bit[x][y] += val;
	}

	int query(int X, int Y){
		int sum = 0;

		for(int x = X; x > 0; x -= x&(-x))
			for(int y = Y; y > 0; y -= y&(-y))
				sum += bit[x][y];

		return sum;
	}

	void updateArea(int xi, int yi, int xf, int yf, int val){
		update(xi,   yi,    val);
		update(xf+1, yi,   -val);
		update(xi,   yf+1, -val);
		update(xf+1, yf+1,  val);
	}

	int queryArea(int xi, int yi, int xf, int yf){
		return query(xf, yf) - query(xf, yi-1) - query(xi-1, yf) +
		    query(xi-1, yi-1);
	}

	void init(){
		memset(bit, 0, sizeof bit);
	}

} Bit;


int main(){
	cout << "Binary Indexed Tree 2D - Fenwick Tree 2D" << endl;
	return 0;
}

/*
Syntax:

Bit.init();                              //Seta tudo como 0
Bit.update(x, y, v);    //Adiciona +v na posi  o {x, y} da BIT
Bit.query(x, y);                //Retorna o somatorio do retangulo de
    inicio {1, 1} e fim {x, y}
Bit.queryArea(xi, yi, xf, yf);     //Retorna o somatorio do retangulo de
    inicio {xi, yi} e fim {xf, yf}
Bit.updateArea(xi, yi, xf, yf, v); //adiciona +v no retangulo de inicio {xi
    , yi} e fim {xf, yf}

IMPORTANTE! UpdateArea N O atualiza o valor de todas as c lulas no
     ret ngulo!!! Deve ser usado para Color Update
IMPORTANTE! Use query(x, y) Para acessar o valor da posi  o (x, y) quando
     estiver usando UpdateArea
IMPORTANTE! Use queryArea(x, y, x, y) Para acessar o valor da posi  o (x,
     y) quando estiver usando Update Padr o

Query:  O(log NM)
Update: O(log NM)

*Build: O(NM log NM)
Para consultas est ticas, sem update, o melhor   usar uma Prefix Sum 2D

-> N: Numero de colunas
-> M: Numero de linhas
```

```
*/
```

## 1.3 Segment Tree

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;
int seg[4*MAXN];


int query(int no, int l, int r, int a, int b){
	if(b <  l || r <  a) return 0;
	if(a <= l && r <= b) return seg[no];

	int m=(l+r)/2, e=no*2, d=no*2+1;

	return query(e, l, m, a, b) + query(d, m+1, r, a, b);
}

void update(int no, int l, int r, int pos, int v){
	if(pos < l || r < pos) return;
	if(l == r){seg[no] = v; return; }

	int m=(l+r)/2, e=no*2, d=no*2+1;

	update(e, l,   m, pos, v);
	update(d, m+1, r, pos, v);

	seg[no] = seg[e] + seg[d];
}

void build(int no, int l, int r, vector<int> &lista){
	if(l == r){ seg[no] = lista[l-1]; return; }

	int m=(l+r)>>1, e=no*2, d=no*2+1;

	build(e, l,   m, lista);
	build(d, m+1, r, lista);

	seg[no] = seg[e] + seg[d];
}


int main()
{
	cout << "Segment Tree" << endl;
	return 0;
}

/*******************************************************

-> Segment Tree com:
	- Query em Range
	- Update em Ponto

-> Chamadas padr o:
	build(1, 1, n, lista);
	query(1, 1, n, a, b);
	update(1, 1, n, i, x);

-> Em que:
|   n    | o tamanho m ximo da lista
| [a, b] | o intervalo da busca
|   i    | a posi  o a ser modificada
|   x    | o novo valor da posi  o i
| lista  | o array de elementos originais


Build:  O(N)
Query:  O(log N)
Update: O(log N)

*******************************************************/
```

## 1.4 Segment Tree Lazy

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;
int seg[4*MAXN];
int lazy[4*MAXN];

void unlazy(int no, int l, int r){
	if(lazy[no] == 0) return;

	int m=(l+r)>>1, e=no*2, d=no*2+1;

	seg[no] += (r-l+1) * lazy[no];

	if(l != r){
		lazy[e] += lazy[no];
		lazy[d] += lazy[no];
	}

	lazy[no] = 0;
}

int query(int no, int l, int r, int a, int b){
	unlazy(no, l, r);
	if(b <  l || r <  a) return 0;
	if(a <= l && r <= b) return seg[no];

	int m=(l+r)>>1, e=no*2, d=no*2+1;

	return query(e, l, m, a, b) + query(d, m+1, r, a, b);
}

void update(int no, int l, int r, int a, int b, int v){
	unlazy(no, l, r);
	if(b <  l || r <  a) return;
	if(a <= l && r <= b)
	{
		lazy[no]+= v;
		unlazy(no, l, r);
		return;
	}

	int m=(l+r)>>1, e=no*2, d=no*2+1;

	update(e, l,   m, a, b, v);
	update(d, m+1, r, a, b, v);

	seg[no] = seg[e] + seg[d];
}


void build(int no, int l, int r, vector<int> &lista){
	if(l == r){ seg[no] = lista[l-1]; return; }

	int m=(l+r)>>1, e=no*2, d=no*2+1;

	build(e, l,   m, lista);
	build(d, m+1, r, lista);

	seg[no] = seg[e] + seg[d];
}


int main()
{
	cout << "Segment Tree - Lazy Propagation" << endl;
```

```
        return 0;
}

/*******************************************************

-> Segment Tree - Lazy Propagation com:
        - Query em Range
        - Update em Range

-> Chamadas padr o:
        build(1, 1, n, lista);
        query(1, 1, n, a, b);
        update(1, 1, n, a, b, x);

-> Em que:
|   n    | o tamanho m ximo da lista
| [a, b] | o intervalo da busca ou update
|   x    | o novo valor a ser somada no intervalo [a, b]
| lista  | o array de elementos originais

Build:  O(N)
Query:  O(log N)
Update: O(log N)
Unlazy: O(1)

*******************************************************/
```

# 2 Graph

## 2.1 Dinic

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long


struct Aresta {
        int u, v; ll cap;
        Aresta(int u, int v, ll cap) : u(u), v(v), cap(cap) {}
};


struct Dinic {

        int n, source, sink;
        vector<vector<int>> adj;
        vector<Aresta> arestas;
        vector<int> level, ptr; //pointer para a pr xima aresta n o
            saturada de cada v rtice


        Dinic(int n, int source, int sink) : n(n), source(source), sink(
            sink) { adj.resize(n); }


        void addAresta(int u, int v, ll cap)
        {
                adj[u].push_back(arestas.size());
                arestas.emplace_back(u, v, cap);

                adj[v].push_back(arestas.size());
                arestas.emplace_back(v, u, 0);
        }


        ll dfs(int u, ll flow = 1e9){
                if(flow == 0) return 0;
                if(u == sink) return flow;

                for(int &i = ptr[u];  i < adj[u].size();  i++)
                {
                        int atual = adj[u][i];
                        int v = arestas[atual].v;

                        if(level[u] + 1 != level[v]) continue;

                        if(ll got = dfs(v, min(flow, arestas[atual].cap)) )
                        {
                                arestas[atual].cap -= got;
                                arestas[atual^1].cap += got;
                                return got;
                        }
                }

                return 0;
        }


        bool bfs(){
                level = vector<int> (n, n);
                level[source] = 0;

                queue<int> fila;
                fila.push(source);


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

                        for(auto i : adj[u]){
                                int v = arestas[i].v;

                                if(arestas[i].cap == 0 || level[v] <= level
                                    [u] + 1 ) continue;

                                level[v] = level[u] + 1;
                                fila.push(v);
                        }
                }

                return level[sink] < n;
        }


        bool inCut(int u){ return level[u] < n; }


        ll maxFlow(){
                ll ans = 0;

                while( bfs() ){
                        ptr = vector<int> (n+1, 0);

                        while(ll got = dfs(source)) ans += got;
                }

                return ans;
        }

};



int main(){
        cout << "Dinic - Max Flow Min Cut" << endl;
        return 0;
}


/**************************************
Algoritmo de Dinic ou Dinitz para encontrar
o Fluxo M ximo e Corte M nimo em um grafo
```

```
Complexity:
O( V^2 * E )      ->  Para grafos gerais
O( sqrt(V) * E )  ->  Para grafos com capacidade = 1 para todos os
    v rtices:


* Informa es:

        Crie o Dinic:
                Dinic dinic(n, source, sink);

        Adicione as Arestas:
                dinic.addAresta(u, v, capacity);

        Para calcular o Fluxo M ximo:
                dinic.maxFlow()

        Para saber se um v rtice U est  no Corte M nimo:
                dinic.inCut(u)


* Sobre o C digo:

        vector<Aresta> arestas;  -> Guarda todas as arestas do grafo e do
            grafo residual

        vector<vector<int>> adj; -> Guarda em adj[u] os  ndices  de todas
            as arestas saindo de u

        vector<int> ptr;   ->  Pointer para a pr xima aresta ainda  n o
            visitada de cada v rtice

        vector<int> level; -> Dist ncia em v rtices a partir do Source.
            Se igual a N o v rtice  n o  foi visitado.

        A BFS retorna se Sink    alcan avel de Source. Se  n o     porque
            foi atingido o Fluxo M ximo

        A DFS retorna um poss vel aumento do Fluxo


IMPORTANTE! O algoritmo est  0-indexado

*************************************/
```

## 2.2 Dijkstra

```cpp
#include <bits/stdc++.h>
using namespace std;

#define INF 0x3f3f3f3f
#define vi vector<int>
#define pii pair<int,int>

const int MAXN = 1e6 + 5;

vector<pii> grafo [MAXN];

vi dijkstra(int s){
        vi dist (MAXN, INF);

        priority_queue<pii, vector<pii>, greater<pii>> fila;
        fila.push({0, s});
        dist[s] = 0;

        while(!fila.empty())
        {
                auto [d, u] = fila.top();
                fila.pop();

                if(d > dist[u]) continue;

                for(auto [v, c] : grafo[u])
                        if( dist[v] > dist[u] + c )
                        {
                                dist[v] = dist[u] + c;
                                fila.push({dist[v], v});
                        }
        }

        return dist;
}


int main(){
        cout << "Dijkstra - Shortest Paths from Source" << endl;
        return 0;
}

/*************************************
Algoritmo para encontrar o caminho
minimo de um vertice u para todos os
outros vertices de um grafo qualquer

Complexity:
O(N Log N)

dijkstra(s)                              ->  s : Source, Origem. As
    distancias serao calculadas com base no vertice s
grafo[u] = {v, c};       ->  u : Vertice inicial, v : Vertice final, c :
    Custo da aresta
priority_queue<pii, vector<pii>, greater<pii>> ->  Ordena pelo menor custo
    -> {d, v} -> d : Distancia, v : Vertice
*************************************/
```

## 2.3 Tarjan - Pontes

```cpp
#include <bits/stdc++.h>
#define ll lowng lowng
#define pii pair<int,int>
#define INF 0x3f3f3f3f
using namespace std;

const int MAXN = 1e6 + 5;

vector<int> grafo [MAXN];
int pre[MAXN], low[MAXN], clk=0;

vector<pair<int, int>> pontes;
vector<int> cut;


void tarjan(int u, int p = -1){
        pre[u] = low[u] = clk++;

        bool any = false;
        int chd = 0;

        for(auto v : grafo[u]){
                if(v == p) continue;

                if(pre[v] == -1)
                {
                        tarjan(v, u);

                        low[u] = min(low[v], low[u]);

                        if(low[v] >  pre[u]) pontes.emplace_back(u, v);
```

```cpp
                        if(low[v] >= pre[u]) any = true;

                        chd++;
                }
                else
                        low[u] = min(low[u], pre[v]);
        }

        if(p == -1 && chd >= 2) cut.push_back(u);
        if(p != -1 && any)      cut.push_back(u);
}


int main(){
        memset(pre, -1, sizeof pre);

        cout << "Tarjan - Pontes e Pontos de Articula  o" << endl;

        return 0;
}


/***************************************
Algoritmo para encontrar todas as pontes
e v rtices de articula  o, ou v rtice de
corte, de um grafo.

Complexity:
O(V + E)

*** Vari veis e explica  es ***

pre[u] = "Altura", ou, x- simo  elemento visitado na DFS. Usado para saber
    a posi  o de um v rtice na  rvore  de DFS
low[u] = Low Link de U, ou a menor aresta de retorno (mais pr xima da raiz
    ) que U alcan a entre seus filhos

chd = Children. Quantidade de componentes filhos de U. Usado para saber se
    a Raiz    Ponto de Articula  o.
any = Marca se alguma aresta de retorno em qualquer dos componentes filhos
    de U n o ultrapassa U. Se isso for verdade, U    Ponto de
    Articula  o.

if(low[v] >  pre[u]) pontes.emplace_back(u, v); -> se a mais alta aresta de
     retorno de V (ou o menor low) estiver abaixo de U, ent o U-V
    ponte
if(low[v] >= pre[u]) any = true;                              -> se a
    mais alta aresta de retorno de V (ou o menor low) estiver abaixo de U
    ou igual a U, ent o U    Ponto de Articula  o

****************************************/
```

## 2.4 LCA

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e4 + 5;
const int MAXLG = 16;

vector<int> grafo[MAXN];

int bl[MAXLG][MAXN], lvl[MAXN];
int N;


void dfs(int u, int p, int l=0){
        lvl[u] = l;
        bl[0][u] = p;

        for(auto v : grafo[u])
                if(v != p)
                        dfs(v, u, l+1);
}


void buildBL(){
        for(int i=1; i<MAXLG; i++)
                for(int u=0; u<N; u++)
                        bl[i][u] = bl[i-1][bl[i-1][u]];
}


int lca(int u, int v){
        if(lvl[u] < lvl[v]) swap(u, v);


        for(int i=MAXLG-1; i>=0; i--)
                if(lvl[u] - (1<<i) >= lvl[v])
                        u = bl[i][u];


        if(u == v) return u;


        for(int i=MAXLG-1; i>=0; i--)
                if(bl[i][u] != bl[i][v])
                        u = bl[i][u],
                        v = bl[i][v];


        return bl[0][u];
}



int main(){
        cout << "LCA - Lowest Common Ancestor - Binary Lifting" << endl;
        return 0;
}


/***************************************
Algoritmo para encontrar o menor ancestral
comum entre dois v rtices U e V em uma  rvore
enraizada

Complexity:
dfs()      ->  O(V+E)
buildBL() ->  O(N Log N)
lca()      ->  O(Log N)


* Informa  es
        -> Monte o grafo na lista de adjac ncias
        -> chame dfs(root, root) para calcular o pai e a altura de cada
            v rtice
        -> chame buildBL() para criar a matriz do Binary Lifting
        -> chame lca(u, v) para encontrar o menor ancestral comum

        bl[i][u] -> Binary Lifting com o (2^i)- simo  pai de u
        lvl[u]   -> Altura ou level de U na  rvore

* Em LCA o primeiro FOR iguala a altura de U e V
* E o segundo anda at   o primeiro v rtice de U que n o    ancestral de V
* A resposta    o pai desse v rtice


IMPORTANTE! O algoritmo est   0-indexado

****************************************/
```

## 2.5 DSU Persistente

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;

int pai[MAXN], sz[MAXN], tim[MAXN], t=1;

inline int find(int u, int q = INT_MAX){
        if( pai[u] == u || pai[u] == -1 || q < tim[u] ) return u;
        return find(pai[u], q);
}

inline void join(int u, int v){
        u = find(u);
        v = find(v);

        if(u == v) return;
        if(sz[v] > sz[u]) swap(u, v);

        pai[v] = u;
        tim[v] = t++;
        sz[u] += sz[v];
}

inline void resetDSU(){
        memset(pai, -1, sizeof pai);
        for(int i=0; i<MAXN; i++) sz[i] = 1;
        memset(tim, 0, sizeof tim);
}


int main(){
        resetDSU();
        cout << "Persistent Disjoint Set Union - Persistent Union Find" <<
            endl;
        return 0;
}

/*************************************************
-> Complexity:

 - Find: O( Log N )

  find(u, q) -> Retorna o representante do conjunto de U no tempo Q

 * N o    poss vel utilizar Path Compression
 * tim -> tempo em que o pai de U foi alterado
*************************************************/
```

# 3 Dynamic Programming

## 3.1 Longest Common Subsequence

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 5*1e3 + 5;
int memo[MAXN][MAXN];

string s, t;

inline int LCS(int i, int j)
{
        if(i == s.size() || j == t.size()) return 0;
        if(memo[i][j] != -1) return memo[i][j];

        if(s[i] == t[j]) return memo[i][j] = 1 + LCS(i+1, j+1);

        return memo[i][j] = max(LCS(i+1, j), LCS(i, j+1));
}


inline int LCS_It()
{

        for(int i=s.size()-1; i>=0; i--)
                for(int j=t.size()-1; j>=0; j--)
                {
                        if(s[i] == t[j])
                                memo[i][j] = 1 + memo[i+1][j+1];
                        else
                                memo[i][j] = max( memo[i+1][j], memo[i][j
                                    +1] );
                }

        return memo[0][0];
}


inline string RecoverLCS(int i, int j)
{
        if(i == s.size() || j == t.size()) return "";

        if(s[i] == t[j]) return s[i] + RecoverLCS(i+1, j+1);

        if(memo[i+1][j] > memo[i][j+1]) return RecoverLCS(i+1, j);

        return RecoverLCS(i, j+1);
}


int main(){
        cin >> s >> t;

        cerr << "Max size: " << LCS_It() << endl;

        cout << RecoverLCS(0, 0) << endl;

        return 0;
}

/****************************

LCS - Longest Common Subsequence

Complexity: O(N^2)

* Recursive:
memset(memo, -1, sizeof memo);
LCS(0, 0);

* Iterative:
LCS_It();


* RecoverLCS
  Complexity: O(N)
  Recover one of all the possible longest
  common subsequence.
  Return a String.

****************************/
```

## 3.2 Longest Increasing Subsequence

```
#include <bits/stdc++.h>
```

```cpp
using namespace std;

int LIS(vector<int>& nums)
{
	vector<int> lis;

	for(auto x : nums)
	{
		auto it = lower_bound(lis.begin(), lis.end(), x);

		if(it == lis.end()) lis.push_back(x);
		else *it = x;
	}

	return (int) lis.size();
}


int main(){
	//sequence reading
	int n; cin >> n;
	vector<int> num (n);
	for(auto &x : num) cin >> x;


	cout << LIS(num) << endl;


	return 0;
}

/****************************

LIS - Longest Increasing Subsequence

Complexity: O(N Log N)

* For ICREASING sequence, use lower_bound()
* For NON DECREASING sequence, use upper_bound()

To recover the answer, add an array that holds
the index of the added element. (Replace and add
indexes the same way you do with the LIS)

****************************/
```

# 4 String

## 4.1 Hash

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;


const int MAXN = 1e6 + 5;

const ll MOD = 1200000961; //WA? Muda o MOD e a base
const ll base = 153;

ll expBase[MAXN];


void precalc(){
	expBase[0] = 1;
	for(int i=1; i<MAXN; i++)
		expBase[i] = (expBase[i-1]*base)%MOD;
}

struct StringHash{
	vector<ll> hsh;
	int size;

	StringHash(string &_s){
		hsh = vector<ll>(_s.size()+1, 0);
		size = _s.length();
		for(int i=0; i<_s.size(); i++)
			hsh[i+1] = ((hsh[i]*base) % MOD +_s[i]) % MOD;
	}

	ll gethash(int l, int r){
		return (MOD + hsh[r+1] - (hsh[l]*expBase[r-l+1]) % MOD ) %
		     MOD;
	}
};


int main(){
	cout << "String Hash" << endl;
	return 0;
}

/*******************************************************
String Hash

Complexidade:
precalc()     -> O(N)
StringHash() -> O(|S|)
gethash()     -> O(1)

StringHash hash(s);      -> Cria uma struct de StringHash para a string s
hash.gethash(l, r); -> Retorna o hash do intervalo L R da string (0-
    Indexado)

IMPORTANTE! Chamar precalc() no in cio do c digo

const ll MOD = 12'501'968'177; -> Big Prime Number
const ll base = 127;               -> Random number larger than the
    Alphabet

*******************************************************/


/************************
Some Big Prime Numbers:
127
157
1201
37139213
127065427
131807699
************************/
```

## 4.2 Double Hash

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;


const int MAXN = 1e6 + 5;

const ll MOD1 = 1200000961;
const ll MOD2 = 1227090031;
const ll base = 157;

ll expBase1[MAXN];
ll expBase2[MAXN];

void precalc(){
```

```
    expBase1[0]=1;
    expBase2[0]=1;

        for (int i=1;i<MAXN;i++)
        {
        expBase1[i] = ( expBase1[i-1]*base ) % MOD1;
        expBase2[i] = ( expBase2[i-1]*base ) % MOD2;
    }
}

struct StringHash{
    vector<pair<ll,ll>> hsh;
        int size;

    StringHash(string& _s)
        {
        hsh = vector<pair<ll,ll>> (_s.size()+1, {0,0});
                size = _s.size();

        for (int i=0;i<_s.length();i++)
                {
            hsh[i+1].first  = ( (hsh[i].first *base) % MOD1 + _s[i] ) %
                MOD1;
            hsh[i+1].second = ( (hsh[i].second*base) % MOD2 + _s[i] ) %
                MOD2;
        }
    }

    pair<ll,ll> getKey(int a,int b)
        {
        auto h1 = (MOD1 + hsh[b+1].first  - ( hsh[a].first *expBase1[b-a+1]
             ) % MOD1) % MOD2;
        auto h2 = (MOD2 + hsh[b+1].second - ( hsh[a].second*expBase2[b-a+1]
             ) % MOD2) % MOD2;
        return {h1,  h2};
    }

};


int main(){
        cout << "String Hash - Double Hash" << endl;
        return 0;
}

/*******************************************************
String Hash

Complexidade:
precalc()    -> O(N)
StringHash() -> O(|S|)
gethash()    -> O(1)

StringHash hash(s);     -> Cria uma struct de StringHash para a string s
hash.gethash(l, r); -> Retorna um pair com os dois hashs do intervalo L R
     da string (0-Indexado)

IMPORTANTE! Chamar precalc() no in cio do c digo

const ll MOD1 = 12'501'968'177; -> Big Prime Number for hash 1
const ll MOD1 =  1'227'090'031; -> Big Prime Number for hash 2
const ll base = 127;                -> Random number larger than the
     Alphabet

*******************************************************/


/***********************
Some Big Prime Numbers:
127
157
1201
37139213
127065427
131807699
************************/
```

## 4.3 Z-Function

```
#include <bits/stdc++.h>
using namespace std;

#define vi vector<int>

vi Zfunction(string &s)
{
        int n = s.size();
        vi z (n, 0);

        for(int i=1, l=0, r=0; i<n;  i++)
        {
                if(i <= r) z[i] = min(z[i-l], r-i+1);

                while(z[i] + i < n && s[z[i]] == s[i+z[i]]) z[i]++;

                if(r < i+z[i]-1) l = i, r = i+z[i]-1;
        }

        return z;
}


int main(){
        cout << "Z-Function" << endl;
        return 0;
}

/*****************

Complexidade: O(N)

******************/
```

## 4.4 Manacher

```
#include <bits/stdc++.h>
using namespace std;

#define vi vector<int>

vi manacher(string &st)
{
        string s = "$_";
        for(char c : st){ s += c; s += "_"; }
        s += "#";

        int n = s.size()-2;

        vi p(n+2, 0);
        int l=1, r=1;

        for(int i=1, j; i<=n; i++)
        {
                p[i] = max(0, min(r-i, p[l+r-i]) );     //atualizo o valor
                    atual para o valor do palindromo espelho na string ou
                    para o total que est  contido

                while( s[i-p[i]] == s[i+p[i]] ) p[i]++;

                if( i+p[i] > r ) l = i-p[i], r = i+p[i];
        }
```

```
        for(auto &x : p) x--; //o valor de p[i]    igual ao tamanho do
            palindromo + 1

        return p;
}

int main(){
        cout << "Manacher" << endl;
        return 0;
}

/*****************

Complexidade: O(N)

*****************/
```

## 4.5 KMP

```
#include <bits/stdc++.h>
using namespace std;

vector<int> pi(string &t){
        vector<int> p(t.size(), 0);

        for(int i=1, j=0; i<t.size(); i++)
        {
                while(j > 0  && t[j] != t[i]) j = p[j-1];

                if(t[j] == t[i]) j++;

                p[i] = j;
        }

        return p;
}

vector<int> kmp(string &s, string &t){
        vector<int> p = pi(t), occ;

        for(int i=0, j=0; i<s.size(); i++)
        {
                while( j > 0 && s[i] != t[j]) j = p[j-1];

                if(s[i]==t[j]) j++;

                if(j == t.size()) occ.push_back(i-j+1), j = p[j-1];
        }

        return occ;
}


int main()
{
        cout << "KMP - Pattern Searching" << endl;

        return 0;
}

/*************************
 K n u t h MorrisPratt  Algorithm / KMP

Complexity: O(|S|+|T|)

S -> String
T -> Pattern
*************************/
```

## 4.6 Trie

```
#include <bits/stdc++.h>
using namespace std;

const int MAXS = 1e5 + 10;
const int sigma = 26;

int trie[MAXS][sigma], terminal[MAXS], z = 1;


void insert(string &p)
{
        int cur = 0;

        for(int i=0; i<p.size(); i++){

                int id = p[i] - 'a';

                if(trie[cur][id] == -1 ){
                        memset(trie[z], -1, sizeof trie[z]);
                        trie[cur][id] = z++;
                }

                cur = trie[cur][id];
        }

        terminal[cur]++;
}


int count(string &p)
{
        int cur = 0;

        for(int i=0; i<p.size(); i++)
        {
                int id = (p[i] - 'a');

                if(trie[cur][id] == -1) return 0;

                cur = trie[cur][id];
        }

        return terminal[cur];
}



void init(){
        memset(trie[0], -1, sizeof trie[0]);
        z = 1;
}



int main(){
        cout << "Trie -  rvore  de Prefixos" << endl;
        return 0;
}

/*****************

Complexidade:

insert(P) - O(|P|)
count(P)  - O(|P|)

MAXS  - Soma do tamanho de todas as Strings
sigma - Tamanho do alfabeto
```

```
*****************/
```

# 5 Outros

## 5.1 Hungaro

```
#include <bits/stdc++.h>
using namespace std;

typedef int TP;

const int MAXN = 1e3 + 5;
const TP INF = 0x3f3f3f3f;

TP matrix[MAXN][MAXN];
TP row[MAXN], col[MAXN];
int match[MAXN], way[MAXN];

TP hungarian(int n, int m){
        memset(row, 0, sizeof row);
        memset(col, 0, sizeof col);
        memset(match, 0, sizeof match);

        for(int i=1; i<=n; i++)
        {
                match[0] = i;
                int j0 = 0, j1, i0;
                TP delta;

                vector<TP> minv (m+1, INF);
                vector<bool> used (m+1, false);

                do {
                        used[j0] = true;
                        i0 = match[j0];
                        j1 = -1;
                        delta = INF;

                        for(int j=1; j<=m; j++)
                                if(!used[j]){
                                        TP cur = matrix[i0][j] - row[i0] -
                                             col[j];

                                        if( cur < minv[j] ) minv[j] = cur,
                                             way[j] = j0;
                                        if(minv[j] < delta) delta = minv[j
                                             ], j1 = j;
                                }

                        for(int j=0; j<=m; j++)
                                if(used[j]){
                                        row[match[j]] += delta,
                                        col[j] -= delta;
                                }else
                                        minv[j] -= delta;

                        j0 = j1;
                } while(match[j0]);


                do {
                        j1 = way[j0];
                        match[j0] = match[j1];
                        j0 = j1;
                } while(j0);
        }

        return -col[0];
}


vector<pair<int, int>> getAssignment(int m){
        vector<pair<int, int>> ans;

        for(int i=1; i<=m; i++)
                ans.push_back(make_pair(match[i], i));

        return ans;
}



int main(){
        cout << "Hungarian Algorithm - Assignment Problem" << endl;
        return 0;
}

/*************************************
Algoritmo para o problema de atribui  o m nima.

Complexity:
O(N^2 * M)

hungarian(int n, int m);  ->  Retorna o valor do custo m nimo
getAssignment(int m)      ->  Retorna a lista de pares <linha, Coluna> do
     Minimum Assignment

n -> N mero de Linhas
m -> N mero de Colunas

IMPORTANTE! O algoritmo    1-indexado
IMPORTANTE! O tipo padr o est  como int, para mudar para outro tipo
     altere | typedef <TIPO> TP; |

Extra: Para o problema da atribui  o m xima, apenas multiplique os
     elementos da matriz por -1
*************************************/
```