AED2 - Lista 5
Ordenação digital, busca de palavras, tries

Seguem alguns exercícios relacionados com ordenação digital, busca de palavras em um texto e árvores de busca digital.

1 - [DNA] Escreva uma função que rearranje em ordem crescente um vetor cujos elementos são os caracteres A, C, G e T do código genético. Comece por transformar o conjunto A C G T no intervalo numérico 0 . . 3.

2 - [Vetor de structs] Imagine que o vetor v[0 .. n - 1] representa o cadastro de funcionários de uma empresa. Cada elemento do vetor é uma struct com dois campos: um campo num, que dá o número do funcionário, e um campo nome, que dá o nome do funcionário. Suponha que cada número de funcionário é um inteiro com três dígitos decimais. Adapte o código de countingSort para ordenar v[0..n-1] pelo campo num.

3 - Mostre que a função countingSort é estável.

4 - Na função ordenacaoDigital, diga o que acontece se trocarmos a linha for (d = W - 1; d >= 0; -- d) por for (d = 0; d < W; ++ d).

5 - [Sedgewick-Wayne 5.1.2] Aplique o algoritmo de ordenação digital ao vetor de strings: no is th ti fo al go pe to co to th ai of th pa

6 - [Sedgewick-Wayne 5.1.20] Escreva uma função que receba inteiros n e W e produza um vetor de n strings aleatórias com W caracteres ASCII cada. (Essa função gera dados de teste para ordenacaoDigital.)

7 - [13.2.1] - Dê um exemplo em que o algoritmo básico para busca de palavras em um texto faz o maior número possível de comparações entre elementos de a e b. Descreva o exemplo com precisão.

8 - [13.3.3] - Mostre que é possível eliminar o incômodo “if (k = n) k += 1; else” no código da função BoyerMoore1 com o auxílio de uma sentinela postada em b[n + 1].

9 - [13.3.4] - Mostre que a seguinte variante da função BoyerMoore1 está correta:

```
int ult[256];
for (int i = 0; i < 256; ++i) ult[i] = 0;
for (int i = 1; i < m; ++i) ult[a[i]] = i;
```

```
int ocorrs = 0, k = m;
while (k <= n) {
    r = 0;
    while (r < m && p[m - r] == t[k - r])
        r++;
    if (r >= m) ocorrs++;
    k += ult[t[k]];
}
return ocorrs;
```

10 - [13.4.1] - Considere a função BoyerMoore2. Calcule a tabela alcance[ ] no caso em que p[1] = p[2] = … = p[m]. Calcule a tabela no caso em que os elementos de p[1 .. m] são distintos dois a dois.

11 - [13.4.3] - [preProcGoodSuff Eficiente] Mostre que o código abaixo calcula corretamente a tabela alcance[ ]. Mostre que a execução do código não consome mais que m unidades de tempo (e portanto é bem mais eficiente que o código visto em aula).

```
i = q = m;
do {
    q -= 1; r = 0;
    while (q - r >= 1 && a[m - r] == a[q - r]) r += 1;
    while (i > m - r) alcance[i--] = m - q;
} while (q - r >= 1);
while (i >= 1) alcance[i--] = m - q;
```

12 - De quantas maneiras diferentes pode terminar uma busca (bem- ou malsucedida) numa trie?

13 - As chaves de uma trie precisam ser comparáveis?

14 - (SW 5.2.3) Insira as chaves now is the time for all good people to come to the aid of numa trie inicialmente vazia. Faça um desenho da trie resultante. (Não desenhe os links de valor null.)

Para revisar conceitos sobre ordenação digital, busca de palavras e tries, além de encontrar mais exercícios, acesse:

- https://www.ime.usp.br/~pf/algoritmos/aulas/radix.html
- https://www.ime.usp.br/~pf/algoritmos/aulas/strma.html
- https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/tries.html