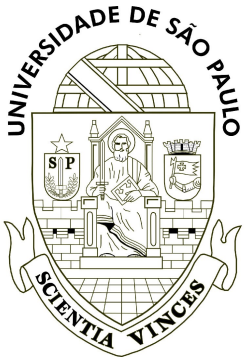
Universidade de São Paulo
Instituto de Matemática e Estatística

MAC0323 - Algoritmos e Estruturas de Dados II
2023



Exercício-programa 2: Tabela de Símbolos

Beatriz Viana Costa

Conteúdo

1. Estrutura de Dados e Algoritmo

As estruturas de dados feitas foram as indicadas no enunciado: vetor dinâmico ordenado, árvore de busca binária, *treap*, árvore rubro-negra e árvore 2-3. Estas foram utilizadas tanto nas funções de busca de uma palavra específica, tanto na função de busca das palavras mais frequentes.

Ademais, para a realização das demais funções de consulta disponíveis, foram utilizadas *structs* contendo um vetor de *strings* e mais um ou dois atributos do tipo *int*, para guardar informações adicionais, como o tamanho das maiores ou menores palavras, quantidade de vogais sem repetição, entre outros.

O algoritmo implementado lê pela linha de comando o nome do arquivo contendo as informações necessárias, a partir disso insere o texto de entrada na estrutura desejada, enquanto isso realiza o pré-processamento de algumas informações, como quais são as palavras mais longas, quais as maiores palavras que não repetem letras e quais as menores palavras com mais vogais sem repetição.

Após terminada a inserção das palavras na estrutura é feita a leitura de quantas e quais são as consultas desejadas.

As consultas de quais são as palavras mais frequentes no texto e quantas vezes ocorre uma palavra, especificada pelo usuário, no texto é feita após a inserção do texto na estrutura.

2. Execução

Para a compilar o programa basta digitar o comando *make* no terminal que então será gerado um executável com o nome ”ep2”.

E para executa-lo, basta escrever ”./ep2 <nome do arquivo de leitura>” na linha de comando. O nome deve ser dado com a extensão *.txt* e o arquivo deve apresentar o seguinte formato:

<estrutura de dados desejada>

<quantidade N de palavras que serão lidas no texto>

<texto com N palavras>

<quantidade n de consultas>

<n linhas contendo cada uma o código da consulta desejada>

A estrutura de dados desejada deve ser especificada da seguinte maneira:

(1) Vetor dinâmico ordenado: VO;

(2) Árvore de busca binária: ABB;

(3) Treap: TR;

(4) Árvore rubro-negra: ARN;

(5) Árvore 2-3: A23.

Já as consultas desejadas devem ser especificadas da seguinte maneira:

(1) Quais as palavras mais frequentes no texto: F;

(2) Quantas vezes uma palavra especifica ocorre no texto: O <palavra>;

(3) Quais as palavras mais longas: L;

(4) Quais as maiores palavras que não repetem letras: SR;

(5) Quais as menores palavras com mais vogais sem repetição: VD.

Exemplos de arquivos de entrada para testes podem ser encontrados na pasta ”arquivos_de_testes”, estes são os mesmos utilizados para os testes que foram relatados neste documento.

A saída do programa é feita no próprio terminal, e os resultados são dados na ordem das consultas.

2.1. **Exemplos de execução.**

Algoritmo 1: No meio do caminho - Carlos Drummond de Andrade

```
1  ABB
2  61
3  No meio do caminho tinha uma pedra
4  tinha uma pedra no meio do caminho
5  tinha uma pedra
6  no meio do caminho tinha uma pedra.
7
8  Nunca me esquecerei desse acontecimento
9  na vida de minhas retinas tão fatigadas.
10 Nunca me esquecerei que no meio do caminho
11 tinha uma pedra
12 tinha uma pedra no meio do caminho
13 no meio do caminho tinha uma pedra.
14 2
15 F
16 SR                                                    // Fim da entrada
17
                                                                 // Saída
18
19 pedra tinha uma                                  // Palavras mais frequentes
20 caminho retinas                  // Maiores palavras que não repetem letras
```

Algoritmo 2: Soneto da fidelidade - Vinicius de Moraes

```
A23
103
De tudo ao meu amor serei atento
Antes, e com tal zelo, e sempre, e tanto
Que mesmo em face do maior encanto
Dele se encante mais meu pensamento.

Quero vivê-lo em cada vão momento
E em seu louvor hei de espalhar meu canto
E rir meu riso e derramar meu pranto
Ao seu pesar ou seu contentamento

E assim, quando mais tarde me procure
Quem sabe a morte, angústia de quem vive
Quem sabe a solidão, fim de quem ama

Eu possa me dizer do amor (que tive):
Que não seja imortal, posto que é chama
Mas que seja infinito enquanto dure.
3
L
VD
F                                                    // Fim da entrada

                                                                 // Saída

contentamento                                      // Palavra mais longa
enquanto                    // Menor palavra com mais vogais sem repetição
meu                                                // Palavra mais frequente
```

3. Testes

Os testes realizados foram em relação ao tempo de inserção em cada estrutura, o processamento de cada função, e também a inserção junto do processamento de todas as funções. Para isto foi utilizada a biblioteca *chrono* do *C++*, e sua função *high_resolution_clock::now()*.

Os resultados dos testes podem ser vistos a seguir, cada tempo conseguido foi resultado da média de 10 execuções:

(1) Tempo de inserção:

Hamlet - 32.081 palavras	
Estrutura de dados	Tempo de inserção
Vetor dinâmico ordenado	0,2786702s
Árvore de busca binária	0,1778382s
Treap	0,2148898s
Árvore rubro-negra	0,1820314s
Árvore 2-3	0,1786286s

Oliver Twist - 161.005 palavras	
Estrutura de dados	Tempo de inserção
Vetor dinâmico ordenado	1.0783638s
Árvore de busca binária	1.1140711s
Treap	1.0098421s
Árvore rubro-negra	1.0056262s
Árvore 2-3	1.0070222s

Moby Dick - 209.329 palavras	
Estrutura de dados	Tempo de inserção
Vetor dinâmico ordenado	2,9223426s
Árvore de busca binária	1,2457175s
Treap	1,4023748s
Árvore rubro-negra	1,1861106s
Árvore 2-3	1,1751485s

War and Peace - 566.330 palavras	
Estrutura de dados	Tempo de inserção
Vetor dinâmico ordenado	3,180857s
Árvore de busca binária	3,5301011s
Treap	3,7046553s
Árvore rubro-negra	3,3509732s
Árvore 2-3	3,1605127s

Bible - 826.069 palavras	
Estrutura de dados	Tempo de inserção
Vetor dinâmico ordenado	4,8738659s
Árvore de busca binária	4,6283047s
Treap	5,5286305s
Árvore rubro-negra	4,5678301s
Árvore 2-3	4,3802351s

In search of the lost time - 1.356.895 palavras	
Estrutura de dados	Tempo de inserção
Vetor dinâmico ordenado	12,848597s
Árvore de busca binária	7,8654619s
Treap	9,2489512s
Árvore rubro-negra	7,7507902s
Árvore 2-3	7,8350109s

(2) Tempo da função de encontrar as palavras mais frequentes (F):

Hamlet - 32.081 palavras	
Estrutura de dados	Tempo da função F
Vetor dinâmico ordenado	0,0000084s
Árvore de busca binária	0,0000595s
Treap	0,0000577s
Árvore rubro-negra	0,0000573s
Árvore 2-3	0,0000366s

Oliver Twist - 161.005 palavras	
Estrutura de dados	Tempo de inserção
Vetor dinâmico ordenado	0,0000351s
Árvore de busca binária	0,000119s
Treap	0,0001224s
Árvore rubro-negra	0,000114s
Árvore 2-3	0,0001855s

Moby Dick - 209.329 palavras	
Estrutura de dados	Tempo da função F
Vetor dinâmico ordenado	0,0000351s
Árvore de busca binária	0,003559s
Treap	0,0003368s
Árvore rubro-negra	0,0002911s
Árvore 2-3	0,000446s

War and Peace - 566.330 palavras	
Estrutura de dados	Tempo da função F
Vetor dinâmico ordenado	0,0000359s
Árvore de busca binária	0,0004548s
Treap	0,0003854s
Árvore rubro-negra	0,0003924s
Árvore 2-3	0,0004393s

Bible - 826.069 palavras	
Estrutura de dados	Tempo da função F
Vetor dinâmico ordenado	0,0000195s
Árvore de busca binária	0,0002141s
Treap	0,0002117s
Árvore rubro-negra	0,0001869s
Árvore 2-3	0,0002464s

In search of the lost time - 1.356.895 palavras	
Estrutura de dados	Tempo da função F
Vetor dinâmico ordenado	0,0000666s
Árvore de busca binária	0,0008476s
Treap	0,000761s
Árvore rubro-negra	0,0007065s
Árvore 2-3	0,0008244s

(3) Tempo da função de busca da quantidade de ocorrência de uma palavra específica no texto (O):

Foi feita a pesquisa com somente a palavra *"woman"*, a qual estava em todos os textos utilizados nos testes.

Hamlet - 32.081 palavras	
Estrutura de dados	Tempo da função O
Vetor dinâmico ordenado	0,0000009s
Árvore de busca binária	0,0000007s
Treap	0,0000008s
Árvore rubro-negra	0.0000015s
Árvore 2-3	0,000001s

Oliver Twist - 161.005 palavras	
Estrutura de dados	Tempo da função O
Vetor dinâmico ordenado	0,0000030s
Árvore de busca binária	0,0000017s
Treap	0,000002s
Árvore rubro-negra	0,0000018s
Árvore 2-3	0,0000015s

Moby Dick - 209.329 palavras	
Estrutura de dados	Tempo da função O
Vetor dinâmico ordenado	0,0000020s
Árvore de busca binária	0,0000017s
Treap	0,0000007s
Árvore rubro-negra	0,0000012s
Árvore 2-3	0,000001s

War and Peace - 566.330 palavras	
Estrutura de dados	Tempo da função O
Vetor dinâmico ordenado	0,0000031s
Árvore de busca binária	0,000001s
Treap	0,0000010s
Árvore rubro-negra	0,0000010s
Árvore 2-3	0,000001s

Bible - 826.069 palavras	
Estrutura de dados	Tempo da função O
Vetor dinâmico ordenado	0,0000017s
Árvore de busca binária	0,000001s
Treap	0,000001s
Árvore rubro-negra	0,0000011s
Árvore 2-3	0,0000011s

In search of the lost time - 1.356.895 palavras	
Estrutura de dados	Tempo da função O
Vetor dinâmico ordenado	0,0000036s
Árvore de busca binária	0,0000016s
Treap	0,0000016s
Árvore rubro-negra	0,0000012s
Árvore 2-3	0,0000018s

(4) Tempo da função de busca das palavras mais longas do texto (L):

Tempo de execução da função L	
Arquivo testado	Tempo de execução
Hamlet - 32.081 palavras	0,03506140s
Oliver Twist - 161.005 palavras	0,1504957s
Moby Dick - 209.329 palavras	0,2200819s
War and Peace - 566.330 palavras	0,3809997s
Bible - 826.069 palavras	0,3840021s
In search of the lost time - 1.356.895 palavras	0,7230467s

Tabela 1. Como esta função depende de uma estrutura auxiliar e não das árvores e vetor ordenado, o tempo de execução é o mesmo para todas essas estruturas de dados desenvolvidas.

As demais funções - quais as maiores palavras que não repetem letras (SR) e quais as menores palavras com mais vogais sem repetição (VD) - não foram testadas individualmente, pois seguem o mesmo princípio da função de encontrar as palavras mais longas: é feito um pré-processamento no qual cada palavra que é lida tem seus atributos (tamanho e quantidade de vogais sem repetição) comparadas, caso se encaixem no que é pedido, são adicionadas à *struct* auxiliar, de forma que não dependem das demais estruturas de dados desenvolvidas no Exercício-Programa (vetor dinâmico ordenado e as árvores).

Logo, o tempo de execução destas funções é similar ao tempo de execução da função de encontrar as palavras mais longas (L).

4. Conclusões finais

Os tempos de execução de cada função foram próximas do esperado. A função de inserção foi a que mais demorou, e o tempo foi proporcional à quantidade de palavras do arquivo testado, como esperado o vetor dinâmico ordenado foi o que mais demorou, pois mesmo que a inserção seja feita por busca binária, para uma quantidade muito grande de palavras são feitas muitas inserções e também muitos *shifts* (mover as palavras para frente para poder inserir uma outra no meio do vetor); logo em seguida, com o segundo maiore tempo de inserção, veio o *treap*, o que pode ser explicado por conta da quantidade de comparações e rotações feitas para manter a propriedade de *MAX HEAP*.

Na função de busca das palavras mais frequentes ocorreu o que não era esperado, o vetor dinâmico obteve o menor tempo de execução, mas isto pode ser explicado pelo fato de que a função F percorre toda a estrutura de dados realizando comparações do atributo *quantidade de ocorrências*. Como o vetor é uma estrutura de dados linear, isto pode ter contribuído para o menor tempo necessário para percorre-lo.

O tempo de execução da função de busca de uma palavra específica teve resultados que foram esperados. O vetor ordenado foi o que mais demorou em todos os casos (mesmo que a diferença de tempo para as demais estruturas não tenha sido muito significativa), já as demais estruturas de árvores obtiveram resultados muito próximos.

Já a função de encontrar as palavras mais longas (e as demais funções que utilizam a mesma *struct* que esta) obteve tempo de execução maior que as demais funções testadas anteriormente, o que era imaginado, uma vez que é realiza comparações toda vez que uma palavra nova é lida, e toda vez que esta palavra é lida e é uma candidata a ser uma das palavras mais longas, deve ser verificado se esta já não esta em nossa estrutura, dessa forma percorremos todo o *vector* realizando comparações.