Relatório EP3 - MAC0121

João Gabriel Basi - N° USP: 9793801

1 O programa

O programa recebe, na linha de comando, um arquivo, um tipo de implemetação de lista ligada e uma ordem de impressão. O programa imprime na saída padrão as todas as palavras do arquivo e suas respectivas ocorrências, tanto em ordem alfabética, ou em ordem de ocorrência, dependendo do que for recebido na linha de comando.

2 As funções

As funções foram separadas em bibliotecas de acordo com a sua utilidade:

2.1 auxfuncs.h

Biblioteca com funções e uma struct que auxiliam o programa a manusear a memória:

- *InsertionResult*: Struct que auxilia no recebimento de dados sobre a inserção de um elemento na tabela de símbolos;
- *emalloc*: Aloca um espaço na memória, mostrando uma mensagem de erro se não houver espaço para a alocação;
- *estrdup*: Dulica uma string, retornando um ponteiro para a cópia e mostrando uma mensagem de erro se não houver espaço para a alocação;

2.2 buffer.h

Biblioteca que cria a struct Buffer e cria funções sobre ela:

- *Buffer*: Struct com um vetor de tamanho dinâmico, que auxilia no recebimento de strings da entrada;
- *BufferCreate*: Cria um buffer;
- *BufferDestroy*: Destroi um buffer;
- *BufferReset*: Reseta um buffer;
- *BufferPush*: Adiciona um caractere no fim do buffer;
- *readLine*: Lê uma linha da entrada e guarda no buffer.

2.3 vectorfuncs.h

Biblioteca com funções comuns entre as tabelas de símbolos implementadas com vetor:

- *Entry*: Struct que guarda uma chave e um valor associado à ela;
- *VectorSTable (VST)*: Struct que guarda uma tabela de símbolos com associações string-int por meio de um vetor de Entries;
- *VTableCreate*: Cria uma tabela de símbolos utilizando vetor;
- *VTableDestroy*: Destroi uma tabela de símbolos implemetada com vetor;
- *VTablePush*: Adiciona uma chave e um valor associado à ela no fim de uma tabela de símbolos feita com vetor;
- *valcompV*: Compara dois valores de uma tabela de símbolos implementda com vetor;
- *strcompV*: Compara duas chaves de uma tabela de símbolos implementda com vetor;
- *mergeSortV*: Organiza uma tabela de símbolos implementada com vetor utilizando a função fornecida.

2.4 linkedlistfuncs.h

Biblioteca com funções comuns entre as tabelas de símbolos implementadas com lista ligada:

- *LLNode*: Nó para lista ligada com uma chave, um valor associado à ela e um ponteiro para o próxomo nó;
- *LinkedListSTable (LLST)*: Cabeça para uma tabela de símbolos com associações string-int feita com uma lista ligada de LLNodes;
- *LLTableCreate*: Cria uma tablea de símbolos utilizando lista ligada;
- *LLTableDestroy*: Destroi uma tabela de símbolos que utiliza lista ligada.

2.5 tabelaSimbolo_VO.h

Biblioteca com funções sobre a tabela de símbolos implementada com vetor ordenado:

- *OVAdd*: Adiciona uma chave à uma tabela de símbolos ordenada e implementada com vetor;
- *OVPritVal*: Imprime os elementos de uma tabela de símbolos, ordenada e implementada com vetor, em ordem decrescente de valor;
- *OVPrintLexi*: Imprime os elementos de uma tabela de símbolos, ordenada e implementada com vetor, em ordem alfabética;

2.6 tabelaSimbolo_VD.h

Biblioteca com funções sobre a tabela de símbolos implementada com vetor desordenado;

- *UVAdd*: Adiciona uma chave à uma tabela de símbolos desordenada e implementada com vetor;
- *UVPrintVal*: Imprime os elementos de uma tabela de símbolos, desordenada e implementada com vetor, em ordem decrescente de valor;
- *UVPrintLexi*: Imprime os elementos de uma tabela de símbolos, desordenada e implementada com vetor, em ordem alfabética;

2.7 tabelaSimbolo_LO.h

Biblioteca com funções sobre a tabela de símbolos implementada com lista ligada ordenada;

- *OLLAdd*: Adiciona uma chave à uma tabela de símbolos ordenada e implementada com lista ligada;
- *OLLPrintVal*: Imprime os elementos de uma tabela de símbolos, ordenada e implementada com lista ligada, em ordem decrescente de valor;
- *OLLPrintLexi*: Imprime os elementos de uma tabela de símbolos, ordenada e implementada com lista ligada, em ordem alfabética;

2.8 tabelaSimbolo_LD.h

Biblioteca com funções sobre a tabela de símbolos implementada com lista ligada desordenada;

- *ULLAdd*: Adiciona uma chave à uma tabela de símbolos desordenada e implementada com lista ligada;
- *ULLPrintVal*: Imprime os elementos de uma tabela de símbolos, desordenada e implementada com lista ligada, em ordem decrescente de valor;
- *ULLPrintLexi*: Imprime os elementos de uma tabela de símbolos, desordenada e implementada com lista ligada, em ordem alfabética;

2.9 tabelaSimbolo_AB.h

Biblioteca com funções sobre a tabela de símbolos implementada com árvore de busca binária.

- *BTNode*: Nó para árvore binária com uma chave, um valor associado à ela, um ponteiro para o nó direito e um ponteiro para o nó esquerdo;

- *BinaryTreeSTable (BTST)*: Raiz para um tabela de símbolos com associações string-int feita com uma árvore binária de BTNodes;
- *BSTTableCreate*: Cria uma tabela de símbolos utilizando árvore binária;
- *BSTTableDestroy*: Destroi uma tabela de símbolos implemetada com árvore binária;
- *BSTAdd*: Adiciona uma chave à uma tabela de símbolos implementada com árvore de busca binária;
- *BSTPrintVal*: Imprime os elementos de uma tabela de símbolos, implementada com árvore de busca binária, em ordem decrescente de valor;
- *BSTPrintLexi*: Imprime os elementos de uma tabela de símbolos, implementada com árvore de busca binária, em ordem alfabética;

Na main:

- *show_usage*: Mostra uma mensagem de erro na saída de erro de acordo com o número fornecido;
- *executeOV*: Executa o programa utilizando uma tabela de símbolos ordenada e implementada com vetor;
- *executeUV*: Executa o programa utilizando uma tabela de símbolos desordenada e implementada com vetor;
- *executeOLL*: Executa o programa utilizando uma tabela de símbolos ordenada e implementada com lista ligada;
- *executeULL*: Executa o programa utilizando uma tabela de símbolos desordenada e implementada com lista ligada;
- *executeBST*: Executa o programa utilizando uma tabela de símbolos implementada com árvore de busca binária;

3 Análize dos algoritmos

3.1 Vetor desordenado

3.1.1 Inserção

A inserção na tabela foi feita de modo linear, comparando a chave a ser inserida com todas as que estão na tabela. Se for achada uma chave igual, a função para de comparar, caso contrário, o algoritmo insere a nova chave no final da lista.

No pior caso, temos todas as palavras do texto diferentes, sempre inserindo a palavra no final da lista. Com isso, para cada palavra, fazemos m comparações (sendo m o número de palavras na tabela). Para n palavras fazemos:

$$\text{Max} = \sum_{m=1}^{n} m \overset{\text{Soma de P.A.}}{\Longrightarrow} \frac{n(n+1)}{2} \text{ comparações}$$

No caso médio, temos que a média de comparações é para uma inserção:

$$E(x) = \sum_{i=1}^{m} \left(i \cdot \frac{1}{m} \right) = \frac{1}{m} \cdot \sum_{i=1}^{m} i \overset{\text{Soma de P.A.}}{\Longrightarrow} \frac{m(m+1)}{2m} = \frac{m+1}{2} \tag{1}$$

Para n inserções, temos, em média, $\frac{n(m+1)}{2}$ comparações. Como é dificil estimar a média de palavras que a tabela terá, não sabemos o valor exato de m, porém sabemos que $\frac{n(m+1)}{2} < \frac{n(n+1)}{2} \Rightarrow m < n$

Já em se tratando de movimentação de elementos da lista, o algoritmo de inserção não movimenta nenhum elemento, já que os elementos novos são inseridos no final da lista.

3.1.2 Impressão

Para ambas as ordens de impressão, é preciso ordenar a lista. Para isso, o programa usa um merge sort, que faz $2n \log n$ movimentações e $n \log n$ comparações em todos os casos.

3.2 Vetor ordenado

3.2.1 Inserção

O programa faz uma busca binária para achar o lugar certo de inserção, então move todos os elementos, a partir desse lugar, uma posição à frente, para poder inserir a nova chave. Na busca binária, o algoritmo faz $\log n$ comparações, e colocando as chaves maiores para frente, se considerarmos que a probabilidade de uma palavra entrar em uma lugar da tabela é a mesma para qualquer posição, ele faz, em média, $\frac{n+1}{2}$ movimentações (a equação fica igual à equação (1) do item anterior).

Então concluímos que, para n elementos, o algoritmo faz $n \log n$ comparações e, em média, $\frac{n(n+1)}{2}$ movimentações.

3.2.2 Impressão

Como o vetor já está ordenado em ordem alfabética, para a impressão em ordem alfabética não são necessários comparações ou movimentações.

Já para a impressão em ordem de frequência, é preciso ordenar o vetor. Para isso foi utilizado um merge sort, que faz $2n \log n$ movimentações e $n \log n$ comparações.

3.3 Lista ligada desordenada

3.3.1 Inserção

O programa faz uma busca linear, que faz, em média, $\frac{n+1}{2}$ comparações (assim como calculado na equação (1) da inserção do vetor desordenado) se a chave estiver na lista, e faz n comparações se ela não estiver; porém não faz nenhuma movimentação, já que o novo nó pode ser inserido trocando o ponteiro do nó anterior.

Se o algoritmo não achar uma chave igual na tabela, ele a insere no final da lista.

Com isso, concluímos que, no caso médio, o algoritmo faz $\frac{n(n+1)}{2}$ comparações, no pior caso faz n^2 comparações e não faz movimentações.

3.3.2 Impressão

Como a lista está desordenada, em ambos os casos programa passa as chaves da lista ligada para um vetor, fazendo n movimentações e n comparações, e ordena o vetor com um merge sort, que faz $2n \log n$ movimentações e $n \log n$ comparações, fazendo no total $n(1 + 2 \log n)$ movimentações e $n(1 + \log n)$ comparações.

3.4 Lista ligada ordenada

3.4.1 Inserção

O programa faz uma busca linear, que faz, em média, $\frac{n+1}{2}$ comparações (assim como calculado na equação (1) da inserção do vetor desordenado), porém, ele para ao achar uma chave maior ou igual à que será inserida e a insere nessa posição.

Com isso, o caso médio do algoritmo faz $\frac{n(n+1)}{2}$ comparações, seu pior caso faz n^2 comaprações e ele não faz nenhuma movimentação.

3.4.2 Impressão

Para a impressão em ordem alfabética, o programa só percorre a lista e imprime seus elementos, fazendo n comparações, já que ela já está ordenada por ordem alfabética. Já para a ordem de ocorrência, o programa faz o mesmo procedimento da impressão da lista ligada desordenada, resultando em $n(1 + 2 \log n)$ movimentações e $n(1 + \log n)$ comparações.

3.5 Árvore de busca binária

3.5.1 Inserção

O programa percorre a árvore fazendo, no melhor caso, $\log n$ comparações e, no pior caso, n comparações, porém a inserção nesse local é feita sem comparações ou movimentações, já que só é preciso mudar o ponteiro da folha em que será inserida o novo elemento.

Com isso, para n elementos, temos que o caso médio é de $n \log n$ comparações, e o pior caso é de n^2 comparações.

3.5.2 Impressão

Para a impressão em ordem alfabética, o programa utiliza um algoritmo recursivo que faz n comparações e nenhuma movimentação. Já para a impressão em ordem de ocorrência, o mesmo método da lista ligada desordenada é usado, resultando em $n(1 + 2\log n)$ movimentações e $n(1 + \log n)$ comparações.

4 Testes

Os testes foram feitos com uma versão do dicionário inglês de 1913 [1], em que os caracteres que não estão na tabela ASCII foram retirados.

Inserção						
Implementação da tabela	Comparações		Movimentações		Tempos	
	médio	pior	médio	pior	Dicionário	Bíblia
Vetor desordenado	$\frac{n(m+1)}{2}$	$\frac{n(n+1)}{2}$	0	0	273,37s	2,7s
Vetor ordenado	$\frac{n(m+1)}{2}$	$\frac{n(n+1)}{2}$	$n(n-m)$	nm	8,72s	0,2s
Lista ligada desordenada	$\frac{n(n+1)}{2}$	n^2	0	0	427,5s	3,3s
Lista ligada ordenada	$\frac{n(n+1)}{2}$	n^2	0	0	10900s (3h)	48s
Árvore de busca binária	$n \log n$	n^2	0	0	1,78s	0,2

Referências

[1] Link para o dicionário `http://www.gutenberg.org/ebooks/29765`