ALGORITMOS E ESTRUTURAS DE DADOS

Second written report

Licenciatura em Engenharia Informática

Pedro Miguel Bastos Almeida – 93150 – 33,3%

Eduardo Henrique Ferreira Santos – 93107 – 33,3%

José Vaz – 88903 – 33,3%

Índice

[Introdução 3](#_Toc28699958)

[Hash Tables 3](#_Toc28699959)

[Separate Chaining 3](#_Toc28699960)

[Código implementado 4](#_Toc28699961)

[Conclusão 8](#_Toc28699962)

# Introdução

Com este projeto nós pretendemos aprofundar os conhecimentos na linguagem C, aperfeiçoando também a prática em programar utilizando novos métodos necessários para a resolução deste problema.

Ajudará também a um melhor entendimento das hash tables e linked lists, assim como o método separate chaining para a sua implementação.

# Hash Tables

A hash table é uma estrutura de dados que suporta a sua criação e destruição, inserção de um par key-value, a procura e a remoção de dados. A informação é procurada através da chave (key). Com um tamanho apropriado, estas operações têm uma complexidade computacional de O(1).

A implementação da hash table pode ser feita de duas maneiras. Com ***Open Addressing****,* é usado um array de chaves e respetivos valores. Já com ***Separate chaining,*** é usado um array com ponteiros para as heads de ***linked lists*** ou ***binary trees***. Neste trabalho, iremos usar o método de separate chaining, com ponteiros para linked lists.

# Separate Chaining

Quando se usa este método, é então usado um array com ponteiros para as linked lists. Cada chave (key), passa pela ***hash function*** que dita qual das linked lists iremos operar (procura, inserção ou remoção). O principal objetivo com o chaining é distribuir as Keys o melhor possível pelas posições do array. Quanto melhor for a distribuição, menos colisões vão existir, e mais rápida será a procura de elementos.

# Código implementado

Em termos de estruturas, temos a ‘file\_data\_t’ e a ‘hash\_data’. A primeira servirá para trabalhar com o ficheiro que vamos trabalhar, e a segunda será a estrutura que define um nó (node) das linked lists usadas na hash table.

A close up of text on a black background

Description automatically generated

Usámos também o código já fornecido pelo docente para trabalhar com o ficheiro, visto que facilita a abertura ( open\_text\_file() ) e o fecho do ficheiro ( close\_text\_file() ). A função read\_word() irá ler cada palavra do ficheiro, o que nos será bastante útil.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Já na parte da hash table, temos a função ‘hash\_function()’, retirada dos slides das Lecture notes, que calcula um hash value dada uma string.

A screen shot of a social media post

Description automatically generated

Seguidamente, temos a função ‘find\_data()’, também adaptada da contida nas Lecture notes, que passando como argumentos a key, a hash table e o seu tamanho retorna o node correspondente, se já estiver contido na hash table. Basicamente, calcula o index do array através da hash function e, depois de encontrar a linked list, percorre-a até ao último elemento.

A screenshot of a cell phone

Description automatically generated

Temos também a função new\_hash\_data() que cria um novo nó com os parâmetros necessários. Há uma pre-alocação da memória necessária para o nó. Depois, basta inicializar os valores.

A screenshot of a cell phone

Description automatically generated

Para nós já existentes, temos a função update() que atualiza os valores de um nó já existente. Para isso, é passado como argumento o nó existente e a posição da palavra repetida. Depois basta calcular a distância entre a posição dada e a última vez que a palavra apareceu e verificar se é maior que ‘maxD’ ou menor que ´minD´ e, se for, substituir. Incrementa-se um no contador, soma-se à distância total (totalD) e atualiza-se a posição da última palavra para a passada como argumento.

A screenshot of a cell phone

Description automatically generated

Por fim, temos a função que aumenta o tamanho da hash table: ‘resize()’. Esta recebe como argumentos o tamanho atual, a hash table e o novo tamanho pretendido. Aloca-se a memória equivalente ao novo tamanho da hash table e coloca-se tudo a NULL. Em seguida, para cada posição do array de ponteiros (primeiro for loop), percorre-se a linked list correspondente (segundo for loop), sendo o primeiro nó da linked list ‘hashtable[i]’. Para cada nó, calcula-se o novo hash value (muda pois o tamanho da hash table também muda). Depois basta manipular os nós de forma a que se coloque na nova hash table. Retorna a nova hash table já com o tamanho pretendido.

A screenshot of a cell phone

Description automatically generated

Passando agora para a função ‘main()’, aloca-se memória e abre-se o ficheiro pretendido. Definimos um tamanho inicial para a hash table e aloca-se memória para a mesma. Em seguida, coloca-se todos os seus valores a NULL. Lemos o ficheiro palavra a palavra usando a função read\_word (que retorna -1 quando acabam as palavras). Verificamos se o número de palavras excede os 60% do tamanho da hash table e, se sim, redimensionamos a hash table para o dobro do tamanho.

Agora temos duas situações possíveis. Se a palavra não existir na hash table (o find\_data vai ser NULL), temos de criar um nó novo. Ou seja, calcular o hash value para achar o index pretendido, depois percorrer essa linked list até ao último valor e adicionar um nó com a palavra pretendida. Por outro lado, se já existir na hash table, temos de chamar a função ‘update()’, já mencionada anteriormente, para atualizar os dados. Está assim então preenchida a hash table.

Por fim, apenas precisamos de imprimir no terminal toda a informação de cada palavra. Percorre-se a hash table. Para cada index verifica-se se contém algum nó e, se sim, é porque tem uma linked list nesse index. Assim, percorre-se essa linked list e, para cada nó, imprime--se no terminal as informações da palavra: primeira e última posição, número de vezes que aparece e a menor, maior e média distância de ocorrências consecutivas da palavra.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

# Conclusão

Em suma, este trabalho ajudou-nos a uma melhor percepção de métodos de criação de código e estruturas de dados, bem como a implementação das hash tables com separate chaining. Enriquecemos também os nossos conhecimentos da linguagem de programação C e na gestão de tempo e de trabalho entre grupo.