Algoritmos e Estruturas de Dados

2ª Série

(Problema)

Agrupar Palavras

49470 Ana Carolina Pereira
50546 Rafael Nicolau

Licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2022/2023

14/05/2023

Índice

1. Introdução

Na resolução deste problema pretendeu-se desenvolver uma aplicação que recebe como input um ficheiro de texto com uma lista de n palavras e que retorne um ficheiro de texto em que as palavras que contenham os mesmos caracteres estão agrupadas por linhas (separadas por uma vírgula).

2. Implementação do ADT AEDHashMap

A classe AEDHashMapAED implementa a interface AEDMutableMap<K, V>, sendo K o tipo das *keys* e V o tipo dos *values*.

Desta forma, dentro desta classe, foi criada uma outra classe Node<K, V> que representa o nó da tabela. Cada nó é representado por uma chave, um valor e foi criada também uma variável *next* de forma a ter como referência o próximo na mesma posição da tabela.

Foi também criada a propriedade *size* - só pode ser definida internamente mas pode ser acedida publicamente - que indica o nº de elementos do mapa.

2.1 Função put(key: K, value: V): V?

A função put adiciona um par chave-valor ao mapa. Começa por calcular o índice da tabela com base na chave e, se a posição correspondente estiver vazia, cria um novo nó e adiciona-o. Se a posição já tiver um nó, então percorre a *linked list* até encontrar a chave correspondente. Se encontrar atualiza o valor, caso contrário adiciona um novo nó ao final da lista.

2.2 Função remove(key: K)

A função remove, remove um par chave-valor com base na chave passada como parâmetro. Inicialmente calcula o índice da tabela com base na chave e percorre a *linked list* até encontrar o nó com a chave correspondente. Posto isto, o nó é removido e os nós vizinhos são atualizados.

2.3 Função get(key: K): V?

Esta função calcula o índice da tabela e percorre a *linked list* até encontrar o nó com a chave correspondente. Se encontrar, retorna o valor associado, caso contrário retorna null.

2.4 Função iterator(): Iterator<AEDMutableMap.MutableEntry<K, V>>

A função iterator retorna um iterador que percorre os elementos da tabela de dispersão. Para isto foi criada a classe interna MyIterator que implementa as funções hasNext() e next(). A função hasNext() existe para verificar se existem mais elementos a serem iterados enquanto que a função next() retorna o próximo elemento.

2.4.1. Função hasNext(): Boolean

A função hasNext(): Boolean verifica se há mais elementos a serem iterados. Desta forma, se o currentElement não for null, significa que ainda há um elemento a ser retornado e retorna true. Caso contrário, verifica se há mais nós na *linked list*, se houver atualiza o currentElement para o nó atual e move nodeIt para o próximo nó na lista. A função retorna true para indicar que ainda há mais elementos e retorna false se nenhum elemento for encontrado.

2.4.2. Função next(): AEDMutableMap.MutableEntry<K, V>

A função next() retorna o próximo elemento na iteração. Se não houver próximo elemento é lançada uma exceção. Caso contrário, o currentElement é armazenado na variável aux, definido como null e a variável auxiliar é retornada.

2.5 Função containsKey(k: K): Boolean

Esta função tem como objetivo verificar se uma chave existe no mapa. Desta forma, é calculado o índice na tabela de dispersão e se não houver nenhum nó no índice retorna false. Caso contrário, percorre a lista à procura de uma correspondência e se encontrar retorna true, se não encontrar a chave em nenhum nó, retorna false.

3. Resolução do problema

Para a resolução do problema, foram criadas duas funções auxiliares, a função generateAnagramHash e a função writeTableToFile.

3.1 Função generateAnagramHash(word: String): Long?

A função generateAnagramHash tem como objetivo atribuir um valor numérico único a cada palavra em função dos caracteres presentes. Desta forma, foi criado um mapa ***primes*** que associa cada letra do alfabeto a um número primo. Posto isto, a função vai percorrer cada caractere da palavra que está a ser passada como parâmetro e converte-o para um *lowercaseChar* para de seguida, verificar se esse caractere está no mapa ***primes***. Se o caractere estiver no mapa então o valor atual é multiplicado pelo número primo que corresponde a esse caractere - este processo é feito para cada caractere da palavra. No fim, o valor que resulta dessa multiplicação é retornado e se, esse valor for igual a 1 então isso significa que a palavra não possui caracteres que correspondam aos números primos definidos no mapa e desta forma, a função vai retornar null.

3.2 Função writeTableToFile(hashtable: Array<AEDHashMapAED.Node <Long, HashSet<String>>?>, writer: PrintWriter)

A função writeTableToFile tem como objetivo escrever os valores armazenados na hashtable num ficheiro de output. A função itera sobre cada elemento da hashtable. Se um elemento for null então significa que não há nenhum valor associado àquela posição e a iteração continua até ao próximo elemento. Caso o elemento não seja null, itera-se sobre todos os nós que estejam relacionados com esse elemento escrevendo o valor de cada nó no ficheiro de output.

3.3 Função groupingWords() - main

Nesta função é criado um mapa utilizando a classe AEDHashMap previamente implementada. Posto isto é inicializado um loop enquanto a variável line (que representa cada linha lida) não for null. Dentro deste loop, para cada palavra no conjunto de palavras obtidas, gera-se um valor para cada palavra utilizando a função generateAnagramHash e verifica-se se esse valor é diferente de null. Se essa condição for verdadeira, ou seja, se o valor obtido for diferente de null vai-se verificar se o mapa já tem a key anagramHash e adiciona a palavra ao conjunto de palavras correspondente. Caso contrário, cria-se um novo conjunto com essa palavra e adiciona-se ao mapa com a chave anagramHash. Por fim, é chamada a função writeTableToFile - explicada anteriormente.

4. Avaliação Experimental

Nesta secção é possível observar os testes experimentais realizados. É ainda importante referir que os valores atribuídos para a realização da tabela e do gráfico foram obtidos numa máquina com processador AMD Ryzen 5 4500U with Radeon Graphics 2.38 GHz, RAM 16,0 GB e sistema operativo de 64 bits, processador baseado em x64.

Nome ficheiro	Tempo
The Prophet	316,647
Metamorphosis	290,7633
Romeo And Juliet	318,6114
The Enchanted April	572,816
The Republic	747,7105
War And Peace	1543,45
Bible	1213,88

Tabela 1: Resultados do tempo de execução para os ficheiros.

A Figura 1, ilustra em termos comparativos através de um gráfico, os tempos de execução dos ficheiros passados como input.

Figura 1: Comparação dos tempos de execução.

5. Conclusões

Com esta série de exercícios, conseguimos aprofundar o conhecimento sobre HashMaps e Listas Ligadas, aplicando os conhecimentos abordados em aula.

Referências

[1] “Disciplina: Algoritmos e Estruturas de Dados - 2223SV,” Moodle 2022/23. [Online]. Available: https://2223.moodle.isel.pt. [Accessed: 16-03-2023].

[2] Introduction to Algorithms, 3º Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. MIT Press.