Atividade Prática II - Estrutura de Dados II

Giulia de Araujo Freulon[1]

[1] Ciência da Computação – Universidade Federal do Maranhão (UFMA)
São Luís – MA – Brasil

giulia.freulon@discente.ufma.br

1. Análise Assintótica

1.1. 1ª questão

Nessa questão foi a escolhida como estrutura de dados auxiliar ao multimapa uma tabela hash com tratamento de colisões como encadeamento de listas ligadas, tendo em vista sua fácil implementação para o problema proposto. Além disso, foi utilizado uma entidade "Empregado" para ser utilizado como um banco de dados teste. Sendo que os atributos do empregado são: id = chave; salário e nome = valores associados à chave. Esses valores são gerados aleatoriamente para testes com 1.000, 100.000 e 1.000.000 de entidades.

O multimapa contém um método *put(chave, valor)* que insere um par chave-valor na tabela hash por meio do uso de uma função hash. Caso a chave ainda não exista no multimapa, é criada uma lista ligada para ela e seu valor é adicionado. Caso a chave do par inserido já exista no multimapa(com valores diferentes) seu valor é adicionado à lista ligada daquela chave. Como o problema de colisão é resolvido com listas ligadas e a posição de cada chave é encontrada com uma função hash, o método *put* tem complexidade de O(1).

Outro método que encontramos no multimapa é o *findAll(chave)* que retorna todos os valores associados à uma chave. Ele também calcula a posição daquela chave com uma função hash(O(1)) e, quando encontra a lista ligada daquela posição, esta é percorrida até o fim, retornando seus valores, por isso tem complexidade de O(v), sendo v = número de valores associados à chave. Desse modo, o método *findAll* tem complexidade de O(1+v).

1.2. 2ª questão

Nessa questão foi utilizado um leitor de documentos.txt que separa suas palavras pelo reconhecimento de um espaço " " entre as palavras e insere-as em uma lista ligada, isso tem complexidade de O(n), sendo n = número de palavras. O plágio é verificado quando uma sequência de m palavras consecutivas idênticas são identificadas em dois documentos diferentes.

Durante a procura de um plágio, utiliza-se um loop que percorre o número de palavras n1 do documento analisado menos m. Dentro desse loop, outro loop percorre um número d-1 de documentos inseridos na tabela. Além disso, outro loop para percorrer as palavras n2 do documento comparado menos m é utilizado. Isso nos dá uma complexidade O((n1-m) * (d-1) * (n2-m), que pode ser simplificada para O(n1*n2*d). Essa complexidade, em geral, é menor que uma complexidade fatorial como O(n*m)!.

- Letra A

Foi utilizada uma tabela hash dinâmica, que dobra seu tamanho assim que identifica que mais de 75% de sua capacidade foi ocupada.

Para a busca do plágio, uma função hash é utilizada tendo tempo médio de busca de O(1). Dessa forma, a complexidade continua O(n1*n2*d).

- Letra B

A árvore escolhida para lidar com esse problema foi a árvore AVL.

Na busca do plágio, utilizamos a busca binária com complexidade O(log n), desse modo a complexidade de busca do plágio fica O(log n + n1*n2*d), que continua menor que uma complexidade fatorial.

- Letra C

O uso da tabela hash dinâmica tem uma complexidade menor que o uso da árvore AVL. Essa árvore foi escolhida por conta da sua fácil implementação em relação às outras e pois ela é uma árvore que tende a ser mais eficiente em operações de busca do que a árvore rubro-negra(menos balanceada) e árvore B(busca em disco).

2. Testes

2.1. 1ª questão

Inserção:

- Vetor Pequeno (1.000 elementos)

```
|Tempo de Execução: 116ms
|Total de Movimentações: 2446
|Total de Comparações: 3000
```

- Vetor Médio (100.000 elementos)

```
|Tempo de Execução: 1135ms
|Total de Movimentações: 249697
|Total de Comparações: 300000
```

- Vetor Grande (1.000.000 elementos)

```
|Tempo de Execução: 7495ms
|Total de Movimentações: 2500729
|Total de Comparações: 3000000
```

Busca por uma chave:

```
|Movimentações: 1 |Comparações: 1
```

2.2. 2ª questão

Com 3 documentos pequenos inseridos e m = 5:

- Letra A

Com plágio:

```
Documento: doc0 - Sequência plagiada: [a, revolução, francesa,, um, dos]
Plágio detectado

|Tempo de Execução: 10ms
|Movimentações: 188
|Comparações: 175
```

Sem plágio:

```
Plágio não detectado

|Tempo de Execução: 13ms
|Movimentações: 285
|Comparações: 223
```

- Letra B

Com plágio:

```
Documento: doc0 - Sequência plagiada: [a, revolução, francesa,, um, dos]
Plágio detectado

|Tempo de Execução: 3ms
|Movimentações: 176
|Comparações: 192
```

Sem plágio:

```
Plágio não detectado

|Tempo de Execução: 9ms
|Movimentações: 123
|Comparações: 256
```

3. Conclusão

Este foi um trabalho que conseguiu abranger o conteúdo de árvores e tabela hash com implementações em problemas reais. Por meio deste, o aluno consegue compreender a fundo as estruturas de dados apresentadas.

Em geral, a primeira questão foi mais simples para implementar do que a segunda. Tive dificuldades, principalmente, na implementação de árvore AVL, haja vista que são muitos casos diferentes. Além disso a lógica para fazer o verificador de plágio é relativamente complexa.

4. Referências

https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/st-hash.html

https://www.javatpoint.com/java-8-multimap

https://www.geeksforgeeks.org/insertion-in-an-avl-tree/