UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Relatório de Implementação e Resolução de Problemas

Alunos: Luís Gustavo Araújo Dias, Joel da Silva Pereira Filho

Problema 1

Descrição do Problema:

Implemente uma estrutura de dados que permite que vários valores sejam associados à mesma chave. Essa estrutura é chamada de multimapa. Ela deve ter um método put (k, v), que insere um item com a chave k e valor v mesmo se já houver um item com a chave k (mas não o mesmo par de valor-chave) e um método FindAll (k), que retorna todos os valores que possuem a chave k. Sua estrutura deve executar o método put (k, v) no tempo O (1) e o método FindAll (k) seja executado no tempo O (1 + v), onde v é o número de valores com chave k. OBS: Você só pode utilizar as estruturas estudadas na segunda unidade (Hash e Árvores).

Testes Realizados:

Pares de Chave-Valor Adicionados:

- Chave1: [248, 336, 263, 327, 719, 508]
- Chave2: [318, 853, 699, 447, 680, 216]
- Chave3: [109, 340, 656, 482, 277, 807]
- Chave4: [972, 451, 779, 29, 375, 989]
- Chave5: [85, 523, 49, 115, 34, 210]
- Chave6: [237, 930, 545, 919, 768, 249]
- Chave7: [31, 177, 977, 429, 707, 484]
- Chave8: [169, 141, 640, 122, 719, 805]

Tabela Hash Gerada no MultiMapa através do **put(k, v)**:

- Tamanho: 16
- Pares:
  - 0 a 3 - null
  - 3 – Key: Chave8, Tamanho da Tabela de Valores: 8, Valores: [122, 640, 141, 169, null, null, 805, 719]
  - 4 – Key: Chave7, Tamanho da Tabela de Valores: 8, Valores: [707, 429, 977, 177, 31, null, null, 484]
  - 5 – Key: Chave6, Tamanho da Tabela de Valores: 8, Valores: [249, 768, 919, 545, 930, 237, null, null]
  - 6 – Key: Chave5, Tamanho da Tabela de Valores: 8, Valores: [null, 210, 34, 115, 49, 523, 85, null]
  - 7 – Key: Chave4, Tamanho da Tabela de Valores: 8, Valores: [null, null, 989, 375, 29, 779, 451, 972]

- 8 – Key: Chave3, Tamanho da Tabela de Valores: 8, Valores: [109, null, null, 807, 277, 482, 656, 340]
- 9 – Key: Chave2, Tamanho da Tabela de Valores: 8, Valores: [853, 318, null, null, 216, 680, 447, 699]
- 10 – Key: Chave1, Tamanho da Tabela de Valores: 8, Valores: [263, 336, 248, null, null, 508, 719, 327]
- 11 a 15 – null

Valores de cada chave buscados através do **findAll(k)**:

- Chave1: [263, 336, 248, null, null, 508, 719, 327]
- Chave4: [null, null, 989, 375, 29, 779, 451, 972]
- Chave5: [null, 210, 34, 115, 49, 523, 85, null]
- Chave15: **Exception** de que *não* existe essa chave na Tabela Hash

Problema 2

Descrição do Problema:

Você foi contratado para desenvolver um verificador de plágio online, que permite que usuários enviem trabalhos escritos e verifiquem se existem copias das seções inteiras de um conjunto, D, de documentos escritos que que você carregou no programa. Você deve carregar qualquer documento, d, e separá-lo em uma sequência de suas n palavras em sua ordem dada (com duplicatas incluídas) em tempo O (n). É considerado um ato de plágio se for utilizada uma sequência de m palavras (em sua ordem) de um documento em D, onde m é um parâmetro definido pelo usuário. Implemente um programa pelo qual você pode ler um documento, d, de n palavras, e testar se ele contém algum plágio. Seu sistema deve processar o conjunto de documentos em D no tempo esperado proporcional ao seu comprimento total, o que é feito apenas uma vez. Seu programa deve detectar o plágio em tempo inferior a O (nm)!. Seu programa deverá apresentar as ocorrências do plágio (documento e parágrafo/frase).

Testes Realizados:

- **Implementação com Árvore Rubro-Negra:**

  Justificativa: Na abordagem de utilização de árvore binária na resolução do problema, optamos por utilizar a Árvore Rubro-Negra em que, com fins comparativos em relação à Árvore AVL, concluímos que, a partir da necessidade de fazer outras operações na árvore, como inserção e busca (sendo a inserção com maior frequência, considerando a limitação pela quantidade de palavras no melhor caso), a partir do conjunto de arquivos lidos para referência de plágio, a Árvore AVL seria menos eficiente justamente por ser mais balanceada que a Árvore Rubro-Negra, o que acaba sendo menor performático para o que tratamos no algoritmo. Já em relação à Árvore B, apesar de ela ser utilizada em Banco de Dados justamente pela sua alta eficiência, escolhemos a Árvore Rubro-Negra, pensando na maior simplicidade em sua implementação.

  Diretório do Arquivo para Verificação:

  - src/reports/problem_2/files/PlagedFileLoremIpsum.txt
    - Quantidade de Palavras Consecutivas: 40
  - src/reports/problem_2/files/EmptyFile.txt
    - Quantidade de Palavras Consecutivas: 9
  - src/reports/problem_2/files/DifferenteFileLoremIpsum.txt
    - Quantidade de Palavras Consecutivas: 900

Diretório do Conjunto de Arquivos de Referência:

- src/reports/problem_2/references/

Resultados:

- src/reports/problem_2/files/PlagedFileLoremIpsum.txt
  - **Plágio**:PLAGIARISM
    Trecho de Plágio: "*Nulla enim mauris, hendrerit scelerisque erat in, consectetur dignissim lectus. Sed mollis convallis dictum. Morbi vel ipsum non massa dignissim*",
    Nome do Arquivo Plageado: OriginalFileLoremIpsum.txt,
    Tempo de Execução (em milissegundos): 87ms
- src/reports/problem_2/files/EmptyFile.txt
  - **Plágio:** NOT_PLAGIARISM,
    Tempo de Execução (em milissegundos): 43ms
- src/reports/problem_2/files/DifferenteFileLoremIpsum.txt
  - **Plágio:** NOT_PLAGIARISM,
    Tempo de Execução (em milissegundos): 81ms

- **Implementação com HashTable:**

  Diretório do Arquivo para Verificação:

  - src/reports/problem_2/files/PlagedFileLoremIpsum.txt
    - Quantidade de Palavras Consecutivas: 40
  - src/reports/problem_2/files/EmptyFile.txt
    - Quantidade de Palavras Consecutivas: 9
  - src/reports/problem_2/files/DifferenteFileLoremIpsum.txt
    - Quantidade de Palavras Consecutivas: 900

  Diretório do Conjunto de Arquivos de Referência:

  - src/reports/problem_2/references/

  Resultados:

  - src/reports/problem_2/files/PlagedFileLoremIpsum.txt
    - **Plágio**:PLAGIARISM
      Trecho de Plágio: "*Nulla enim mauris, hendrerit scelerisque erat in, consectetur dignissim lectus. Sed mollis convallis dictum. Morbi vel ipsum non massa dignissim*",
      Nome do Arquivo Plageado: OriginalFileLoremIpsum.txt,
      Tempo de Execução (em milissegundos): 327ms
  - src/reports/problem_2/files/EmptyFile.txt
    - **Plágio:** NOT_PLAGIARISM,
      Tempo de Execução (em milissegundos): 54ms
  - src/reports/problem_2/files/DifferenteFileLoremIpsum.txt
    - **Plágio:** NOT_PLAGIARISM,
      Tempo de Execução (em milissegundos): 115ms

Comparação de Resultados:

Como demonstrado nos resultados acima (e no conjunto de relatórios de resultados no código-fonte do programa) a abordagem utilizando Árvore Rubro-Negra foi mais eficiente e rápida do que a implementação da Hash Table em todos os 3 casos que testamos, seja em um arquivo "vazio", um mesmo arquivo de plágio e um arquivo que não era plágio. Para medir isso, introduzimos em cada um dos algoritmos um "contador" de tempo em milissegundos e tiramos a dirença entre cada um. No geral, a abordagem da **Árvore Rubro-Negra** para a resolução do problema obteve média de *70.3ms*, enquanto o da **Hash Table** teve uma média de *165.3ms*.

Conclusões:

O trabalho, no geral, foi bem desafiador, desde a implementação "na mão" de uma estrutura de MultiMap, onde, inclusive, implementamos a melhoria de ser dinâmico e o reutilizamos na resolução do segundo problema onde foi pedido uma implementação com HashTable. As maiores dificuldades que enfrentamos, partindo das maiores para as "menores" foram:

- No segundo problema, tivemos dificuldade ao delimitar a quantidade de palavras consecutivas para detecção de plágios. Inicialmente, resolvemos parcialmente o problema sem delimitar a quantidade de palavras, considerando um trecho inteiro de plágio (uma frase ou um parágrafo) para a verificação. A partir daí, tivemos que pesquisar algumas abordagens e tratamentos de Strings para resolver o problema.
- Também no segundo problema, tivemos dificuldade incialmente em pensar em como utilizar as estruturas de HashTable e Árvores para a solução do problema. Inicialmente, pensamos em soluções práticas e simples de fazer, mas que fugiam um pouco da proposta (já que não seriam necessariamente utilizadas as estruturas solicitadas).
- No primeiro problema, em relação ao desenvolvimento do MultiMap, tivemos dificuldade em desenvolver a dinamicidade que propomos como melhoria, o que, na verdade, foi uma confusão de conceitos e ideias para essa abordagem.

Fontes e Referências:

- *https://www.tabnine.com/code/java/methods/java.lang.StringBuilder/append*
- *https://www.guj.com.br/t/manipulacao-de-arquivos-txt-em-java-eclipse/410313/6*
- *https://cursos.alura.com.br/forum/topico-classe-scanner-delimitador-59273*
- *https://github.com/iamareebjamal/RedBlackTree-GUI/blob/master/src/areeb/trees/rbt/RBNode.java*
- *https://github.com/iamareebjamal/RedBlackTree-GUI/blob/master/src/areeb/trees/rbt/RedBlackTree.java*
- *Slides de HashTable e Árvores Rubro-Negras no SIGAA*