UFMA - Universidade Federal do Maranhão
Componente Curricular: Estrutura de Dados II
Alunos: João Victor Abreu Machado e André Luiz Ribeiro de Araujo Lima

Introdução

Este projeto envolve a criação e implementação de algoritmos envolvendo tabela hash e árvores para responder as 2 questões apresentadas pelo professor em sala de aula, que compõem 50% da pontuação total da segunda avaliação. Foi utilizado o recurso de tipos genéricos da linguagem Java, além de terem sido implementadas as estruturas MultMap, MultMap dinâmico e árvore balanceada AVL.

Como resultado deste projeto, foram desenvolvidos métodos otimizados para resolver os problemas. Além disso, uma classe separada foi criada exclusivamente para as estruturas de dados implementadas. A classe principal é responsável por coordenar o funcionamento geral do programa.

Questão 1

Nesse problema, com objetivo de manter o custo O(1+v) pedido pela questão, foi observado que seria interessante implementar a tabela hash com encadeamento. Dessa forma, criamos uma tabela hash genérica com um conjunto de pares <Key, Value>, onde a key é utilizada para calcular o código hash e o Value, na verdade, consiste em uma lista ligada List<V> para criar a tabela hash conforme o que foi concluído anteriormente. Desse modo,

```java
package Questao1;
import Estruturas.*;
joaovictormachados *
public class MultiMapa<K, V> {

    7 usages
    private int tamanho;
    3 usages
    private int totalItens;
    9 usages
    private Entry<K, List<V> >[] tabela;

    1 usage  joaovictormachados
    public MultiMapa(int tamanho) {
        this.tamanho = tamanho;
        tabela = new Entry[tamanho];
        this.totalItens = 0;
    }
    1 usage  joaovictormachados
    public int getTamanho() { return tamanho; }
    1 usage  joaovictormachados
    public Entry<K, List<V> >[] getTabela() { return tabela; }
    1 usage  joaovictormachados
    public int getTotalItens() { return totalItens; }
```

O método put adiciona um valor a uma lista associada a uma key. Calcula o código hash e então tenta inserir diretamente no índice calculado. Então caso ocorram conflitos, tal valor será adicionado ao final da lista, com custo O(1)

```java
10 usages  joaovictormachados
public void put(K key, V value) {
    int indice = generateHash(key);
    if (tabela[indice] ==null) {
        tabela[indice] = new Entry<>(key,new List<V>());
    }
    tabela[indice].setKey(key);
    tabela[indice].getValue().add(value);
    this.totalItens +=1;
}

1 usage  joaovictormachados
public List<V> findAll(K key){
    int indice = generateHash(key);
    if (tabela[indice] != null) {
        return tabela[indice].getValue();
    }
    return null;
}
```

O método findAll, recebe uma key de tipo K, e então calcula o hash para encontrar a posição na tabela. Após esse processamento, ele retorna a lista com os valores de uma posição para depois serem impressos na tela, com custo O(1+v)

A função que gera o hash é calculada da seguinte forma:

```
2 usages   joaovictormachados
public int generateHash(K key) {
    int index = 0;
    if (key instanceof Integer) {
        index = Math.abs( (Integer) key) % tamanho;
    }
    else if(key instanceof Double) {
        int aux = ( (Double) key).intValue();
        index = Math.abs(aux) % tamanho;
    }
    else if (key instanceof Character) {
        index = ( (int) key ) % tamanho;
    }
    else if(key instanceof String) {
        int soma = 0;
        for (char c: ((String) key).toCharArray()) {
            soma += c;
        }
        index = soma % tamanho;
    }

    return index;
}
```

Exemplo de funcionamento do código:

```
Digite o tamanho do multimapa que deseja criar:
5

Indice 0: [Key: 5,Value: Vito], [Key: 5,Value: Andre],
Indice 1: [Key: 6,Value: Marcelo], [Key: 6,Value: Marcelo],
Indice 2: [Key: 7,Value: Gabriel], [Key: 7,Value: Dallyson],
Indice 3: [Key: 8,Value: Marcelino], [Key: 8,Value: Flamengo],
Indice 4: [Key: 9,Value: Patricia], [Key: 9,Value: Computacao],
Valores Associados a posição 3: Marcelino, Flamengo,
Tamanho do MultiMap: 5
Total de Elementos no MultiMap: 10
```

com a entrada:

```
MultiMapa<Integer, String> multimap = new MultiMapa<>(tamanho);
multimap.put(0, "Vito");
multimap.put(1, "Marcelo");
multimap.put(2, "Gabriel");
multimap.put(3, "Marcelino");
multimap.put(4, "Patricia");
multimap.put(5, "Andre");
multimap.put(6, "Marcelo");
multimap.put(7, "Dallyson");
multimap.put(8, "Flamengo");
multimap.put(9, "Computacao");
```

Adicionando elementos com as mesmas chaves:

```
Indice 3: [Key: 3,Value: Dallyson], [Key: 3,Value: Flamengo], [Key: 3,Value: Computacao],
Valores Associados a posição 3: Dallyson, Flamengo, Computacao,
Tamanho do MultiMap: 5
Total de Elementos no MultiMap: 3
```

Questão 2 - Letra A

Para essa questão, era necessário desenvolver um verificador de plágios utilizando uma tabela Hash dinâmica e uma outra versão utilizando uma árvore balanceada (AVL ou Rubro-Negra). Para a questão em geral, criamos arquivos de texto que seriam utilizados como uma base de dados e outros arquivos de texto no qual verificamos se possui algum trecho plagiado dos arquivos base. Para a primeira versão, nós dividimos as palavras presentes nos arquivos em seções de m palavras, cujo valor m era informado pelo usuários. Essas seções de palavras foram guardadas em uma tabela Hash dinâmica, que permite crescimento. A partir disso, nós comparamos as seções de m palavras dos documentos a serem verificados com as seções do documentos bases que estavam na tabela, acusando plágio se as seções fossem iguais. O custo da busca na tabela é O(1+v).

Para a tabela dinâmica foi criada uma nova condição,caso a tabela esteja com um percentual preenchido, calculado pelo loadfactor, a tabela automaticamente tem seu tamanho alterado, para evitar o seu preenchimento total.

Portanto, foi construída a função:

```java
public void resizeTable() {
    this.tamanho = (int) (tamanho *1.5);      //cuidado
    Entry<K,V>[] novaTabela = new Entry[tamanho];

    for (Entry<K,V> entry: tabela) {
        // É necessário recalcular as posições dos valores na nova tabela hash
        if (entry!= null) {
            int indice = generateHash1(entry.getKey());
            int increment = generateHash2(entry.getKey());

            //Colisão Ocorreu
            while (novaTabela[indice]!=null) {
                indice = (indice + increment) % tamanho;
            }
            novaTabela[indice] = entry;
        }
    }
    tabela = novaTabela;
}
```

Além disso,a tabela dinâmica não foi implementada com encadeamento, desse modo foi escolhido criar uma tabela hash com endereçamento fechado e , para calcular as posições, o double hashing.

```java
3 usages  joaovictormachados
public int generateHash1(K key) { return Math.abs(key.hashCode() % tamanho); }
3 usages  joaovictormachados
public int generateHash2(K key) {
    int hash = Math.abs(key.hashCode() % tamanho);
    int increment = hash % (tamanho/2);

    if (increment==0) {
        increment = 1;
    }
    return increment;
}
```

Sendo assim, para inserimos pares <K,V> na tabela dinâmica, foi implementada a função put

```java
1 usage  joaovictormachados
public void put(K key, V value) {
    if (key != null && value != null) {

        boolean inseriu = true;
        if (totalItens > tamanho * loadFactor) {
            resizeTable();
        }

        int indice = generateHash1(key);
        int increment = generateHash2(key);

        //É preciso verificar se onde vamos inserir tem algum elemento ou se ele não é igual
        while (tabela[indice] != null && inseriu) {
            if (tabela[indice].getValue().equals(value)){
                inseriu = false;
            }
            indice = (indice + increment) % tamanho;
        }
        if (inseriu) {
                tabela[indice] = new Entry<>(key,value);
                totalItens++;
        }
    }
}
```

O resultado da aplicação do código é:

```
|----------MENU-----------|
1 - Questão 1
2 - Questão 2 com Tabela Hash
3 - Questão 2 com Árvore AVL
4 - Sair
Opção:
2

Digite o valor de m:
4
Lendo o arquivo: base01
Lendo o arquivo: base02

Plágio no documento teste01, parágrafo 1, no seguinte trecho: nem o vento por
Plágio no documento teste01, parágrafo 2, no seguinte trecho: um bom dia para
Plágio no documento teste01, parágrafo 2, no seguinte trecho: bom dia para estudar
Plágio no documento teste01, parágrafo 2, no seguinte trecho: dia para estudar Estrutura
Plágio no documento teste01, parágrafo 2, no seguinte trecho: para estudar Estrutura de
Plágio no documento teste01, parágrafo 2, no seguinte trecho: estudar Estrutura de Dados
```

utilizando a seguinte entrada:

```java
public static void questao2_A (int m) {
    String pastaDocumentos = "C:\\Users\\andre\\IdeaProjects\\Trabalho2_ED2\\src\\Questao02\\DocumentosBase";
    String pastaVerificar = "C:\\Users\\andre\\IdeaProjects\\Trabalho2_ED2\\src\\Questao02\\TestarPlagio";


    PlagioTabela questao = new PlagioTabela(m);
    questao.uploadDirectory(pastaDocumentos);

    String plagio = questao.verificaPlagioNaTabela(pastaVerificar);
    System.out.println();
    System.out.println(plagio);
}
```

LETRA B

Para este problema, optamos por construir uma implementação usando uma Árvore AVL. A escolha da Árvore AVL foi motivada por diversos fatores:

Balanceamento: isso implica que a diferença de altura entre as subárvores esquerda e direita de qualquer nó não pode exceder uma unidade. Esse critério faz com que a operação de busca tenha um ótimo desempenho, pois o fato da árvore AVL ser mais rigorosa facilita na operação de busca, apesar de deixar as operações de inserção e exclusão um pouco mais lentas que a Rubro-Negra, por exemplo. Como nesse caso haveria poucas inserções e muitas operações de busca, a AVL atende muito bem na resolução do problema proposto.

Facilidade de Implementação: As árvores AVL utilizam apenas rotações para manter o equilíbrio, simplificando o entendimento e a manutenção do código. O balanceamento é alcançado por meio de operações simples e de fácil aplicação.

A árvore AVL foi implementada como uma árvore de strings, e para inserir os valores que serão verificados para plágio, utilizamos uma função de inserção:

```java
public boolean insere (String palavra) {
    raiz = insereNaArvore(palavra, raiz);
    return true;
}

3 usages
public No insereNaArvore (String palavra, No selecionado) {
    palavra = palavra.toLowerCase();
    if(selecionado == null)
        selecionado = new No( palavra, esquerda: null, direita: null );
    else if(converte(palavra) < selecionado.chave) selecionado.esquerda = insereNaArvore(palavra, selecionado.esquerda);
    else if(converte(palavra) > selecionado.chave) selecionado.direita = insereNaArvore(palavra, selecionado.direita);
    selecionado = consertaArvore (selecionado);
    return selecionado;
}
```

A função “converte” converte uma String em um código int

```java
public void uploadArquivo(String diretorio){
    File pasta = new File(diretorio);

    if(pasta.exists() && pasta.isDirectory()){
        File[] arquivos = pasta.listFiles();

        if(arquivos != null){
            for(File a : arquivos){
                if(a.isFile()){
                    dividirTexto(a);
                }
            }
        }
        else{
            System.out.println("Pasta vazia!");
        }
    }
    else{
        System.out.println("Pasta inexistente ou invalida!");
    }
}
```

```java
public void dividirTexto(File arquivo){
    try(BufferedReader l = new BufferedReader(new FileReader(arquivo))){
        String linha;
        System.out.println("Lendo o arquivo: " + arquivo.getName());

        while((linha = l.readLine()) != null){
            String[] palavras = linha.split( regex: " ");
            StringBuilder secaoBuilder = new StringBuilder();

            for(int i = 0; i < palavras.length - m + 1; i++){
                secaoBuilder = new StringBuilder("");

                for(int j = 0; j < m; j++){
                    secaoBuilder.append(palavras[i+j]);
                    secaoBuilder.append(" ");
                }
                String secao = secaoBuilder.toString();
                arvore.insere(secao.toLowerCase());
            }
        }
    }
    catch (IOException e){
        e.printStackTrace();
    }
}
```

Nas duas imagens acima, é mostrado como nós recebemos a pasta com os arquivos base, extraímos os arquivos da pasta, capturamos as palavras e dividimos em seções de tamanho m (o m passado pelo usuário).

```java
public String verificaPlagio(String diretorio){
    ArrayList<String[]> secaoPalavras = verificaArquivo(diretorio);
    StringBuilder plagioBuilder = new StringBuilder();

    for (String[] secao : secaoPalavras) {
        String secaoLowerCase = secao[0].toLowerCase();
        boolean valor = arvore.busca(secaoLowerCase);

        if (valor == true) {
            plagioBuilder.append("Plágio no documento ")
                    .append(secao[2])
                    .append(", parágrafo ")
                    .append(secao[1])
                    .append(", no seguinte trecho: ")
                    .append(secao[0])
                    .append("\n");
        }
    }

    if (plagioBuilder.length() == 0) {
        plagioBuilder.append("Sem plágio :)");
    }

    String plagio = plagioBuilder.toString();

    return plagio;
}
public ArrayList<String[]> verificaArquivo(String diretorio) {
    File d = new File(diretorio);
    ArrayList<String[]> secaoPalavras = new ArrayList<>();

    if (d.exists() && d.isDirectory()) {
        File[] files = d.listFiles();

        if (files != null) {
            for (File f : files) {
                if (f.isFile()) {
                    processaArquivo(f, secaoPalavras);
                }
            }
        }
    }
    return secaoPalavras;
}
```

```java
private void processaArquivo(File file, ArrayList<String[]> secaoPalavras) {
    try (BufferedReader leitor = new BufferedReader(new FileReader(file))) {
        String linha;
        int paragrafo = 0;

        while ((linha = leitor.readLine()) != null) {
            String[] palavras = linha.split( regex: " ");
            if (palavras.length > 1) {
                paragrafo++;
            }

            for (int i = 0; i < palavras.length - m + 1; i++) {
                String[] secao = new String[3];
                secao[0] = "";
                secao[1] = Integer.toString(paragrafo);
                secao[2] = file.getName();

                for (int j = 0; j < m; j++) {
                    secao[0] += palavras[i + j] + " ";
                }
                secaoPalavras.add(secao);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Nas três imagens acima é mostrado como nós utilizamos a função verificaPlagio, passando o diretório da pasta com os arquivos para serem testados como parâmetro. Após isso, nós fizemos a mesma operação que foi feita com os documentos base: extraímos os arquivos da pasta, capturamos as palavras e dividimos em seções com m palavras. Depois, buscamos cada uma dessas seções na árvore, caso haja correspondência, indicamos o plágio.

Para a questão B foi implementada a busca de plágio utilizando árvore AVL, tendo o resultado

```
|----------MENU-----------|
1 - Questão 1
2 - Questão 2 com Tabela Hash
3 - Questão 2 com Árvore AVL
4 - Sair
Opção:
3

Digite o valor de m:
4
Lendo o arquivo: base01
Lendo o arquivo: base02

Plágio no documento teste01, parágrafo 1, no seguinte trecho: nem o vento por
Plágio no documento teste01, parágrafo 2, no seguinte trecho: um bom dia para
Plágio no documento teste01, parágrafo 2, no seguinte trecho: bom dia para estudar
Plágio no documento teste01, parágrafo 2, no seguinte trecho: dia para estudar Estrutura
Plágio no documento teste01, parágrafo 2, no seguinte trecho: para estudar Estrutura de
Plágio no documento teste01, parágrafo 2, no seguinte trecho: estudar Estrutura de Dados
```

Como os resultados da questão A e B são os mesmos, optamos por deixar apenas esse print do funcionamento em relação aos problemas.

Utilizando as seguintes entradas:

```java
public static void questao2_B (int m) {
    ArvoreAVL arvore = new ArvoreAVL();

    String pastaDocumentos = "C:\\Users\\andre\\IdeaProjects\\Trabalho2_ED2\\src\\Questao02\\DocumentosBase";
    String pastaVerificar = "C:\\Users\\andre\\IdeaProjects\\Trabalho2_ED2\\src\\Questao02\\TestarPlagio";

    PlagioArvore plagioA = new PlagioArvore(m);
    plagioA.uploadArquivo(pastaDocumentos);
    String plagio = plagioA.verificaPlagio(pastaVerificar);
    System.out.println();
    System.out.println(plagio);

}
```

Comparando os dois métodos:

Utilizando a tabela hash, temos uma busca mais eficaz, visto que o custo é de O(1+v) comparado aos custos de O(log n) aproximadamente nas árvores.
Para a escolha das árvores, escolhemos a AVL pois ela é mais robusta, e apesar de possuir inserção e remoção mais lenta, ela possui uma busca mais rápida e menos custosa que a rubro-negra. Assim, como faríamos muitas buscas, escolhemos utilizar a AVL, somado a isso, a árvore AVL apresenta implementação menos complexa e trabalhosa para os desenvolvedores, o que facilita na manutenção do código e prevenção de erros.
Portanto, considerando ambas as estruturas de dados, a Tabela Hash demonstra um desempenho significativamente melhor na detecção de plágio em comparação

com uma Árvore Binária de Pesquisa genérica. Assim, foi decidido que a Tabela Hash oferece resultados superiores na verificação de plágio, enquanto ambas as soluções têm custos equivalentes para a distribuição de sequências de tamanho M na tabela.

Conclusão

Este projeto desempenhou um papel crucial na consolidação de nosso conhecimento sobre conceitos como Tabela Hash e Árvore AVL. Além disso, ele ampliou nosso entendimento prático dessas estruturas, destacando sua utilidade no mundo real. Trabalhar em equipe neste projeto envolveu muitas discussões construtivas para determinar as abordagens mais eficazes, permitindo-nos atender aos requisitos do enunciado.

Durante a execução do trabalho, adquirimos diversas lições valiosas. Uma delas foi a implementação manual das estruturas, em vez de depender de estruturas preexistentes em Java. Essa abordagem nos proporcionou uma compreensão mais profunda de como essas estruturas funcionam. Além disso, aprimoramos o desempenho das soluções ao refinar constantemente nossos algoritmos, mesmo após encontrar soluções iniciais para cada desafio.

Percepções e conclusão do discente João Victor Abreu Machado

As principais dificuldades neste trabalho residiam no desenvolvimento de um algoritmo com o desempenho designado. Apesar de existirem soluções mais simples,foi preciso quebrar um pouco mais a cabeça para alcançar o desempenho medido pelo professor nas estruturas implementadas. No entanto, isso contribui para uma melhor assimilação do conteúdo, pois percebemos a importância crucial da escolha da estrutura certa, com as características adequadas para facilitar a resolução e otimização do problema. Assim, todas as dificuldades da tarefa concentraram-se na decisão de como resolver o problema e na escolha da abordagem mais viável.

Entre as partes que fiquei responsável por construir, a questão 1 e a questão 2A, tive algumas dificuldades em relação a criação de uma tabela hash genérica e otimizada. Desse modo, construí duas tabelas hash diferentes, uma para cada questão. As tabelas eram muito parecidas, onde a principal diferença entre as duas foi a escolha do tratamento de colisões, onde na questão 1 foi utilizado hash com encadeamento. Já para a segunda questão, foi implementado uma tabela hash dinâmica com encadeamento fechado, utilizando hashing duplo. Nesse caso, quando a tabela atingia um certo preenchimento, era recalculada e aumentada.

A parte mais difícil, foi pensar em como seria resolvido o problema em relação aos plágios nos arquivos, foi algo que tive uma certa dificuldade, mas com o tempo consegui pensar em uma solução interessante para implementação e resolução.

Percepções e conclusão do discente André Luiz Ribeiro de Araujo Lima

Durante o desenvolvimento deste trabalho, me deparei com algumas dúvidas sobre o conteúdo que apenas na prática pude ter. Para resolver esse problema, pesquisei em sites, vídeos, livros, formas de como fazer isso funcionar, assim, consegui assimilar o conteúdo muito melhor, consumindo de várias fontes, principalmente sobre o conteúdo de árvores, visto que fiquei responsável pela questão 02 utilizando árvores. De início, não sabia por onde começar a fazer um detector de plágio, pois nunca tinha me deparado com isso, mas ao decorrer do trabalho, conseguir assimilar e perceber como um detector de plágio usaria os princípios que eu já havia visto em aula, por isso, aos poucos foi fazendo sentido para mim e a implementação foi tomando forma até chegarmos no resultado final.

No mais, este trabalho foi uma ótima oportunidade para praticar o que foi aprendido em sala e de ter dúvidas para assim ir atrás das respostas, consumindo de várias fontes para poder aumentar o meu conhecimento. Além de todo aprendizado, também sinto que pude melhorar minha comunicação e trabalho em grupo, ou seja, foi muito proveitoso.

Referências

https://www.geeksforgeeks.org/implementing-our-own-hash-table-with-separate-chaining-in-java/
https://www.youtube.com/watch?v=1Ovg3IC-p5A&t=1584s&ab_channel=KracX
https://mauricio.github.io/2020/10/15/implementando-uma-hashtable-em-java.html
https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/
https://www.baeldung.com/java-avl-trees
https://www.javatpoint.com/avl-tree-program-in-java