**Relatório da 3ª Série de Exercícios**

Algoritmos e Estruturas de Dados

**Trabalho realizado por:**

Carlos Florêncio nº:39250

Oxana Dizdari nº:39278

**Turma: LI31D**

Índice

**Introdução3**

**Grupo 1: Exercícios4**

**Problema: A importância dos indivíduos nas redes sociais 5**

**Introdução**

Nesta terceira série de exercícios é pretendido consolidar os conhecimentos obtidos nas aulas durante o semestre inteiro. Necessitamos de utilizar várias estruturas, tais como as árvores, os grafos, os nós. Não podemos esquecer a importância de melhor escolha dos algoritmos de ordenação e de termos sempre em conta os custos de qualquer código que criemos.

**Grupo 1: Exercícios**

**1.**

**1.1) Objectivo:** Realizar o método *public static Integer lower(Node<Integer> root, int k)* que retorna o maior Integer i presente na árvore binária de pesquisa com raíz root que seja estritamente menor do que k, ou null, caso este não exista. Assuma que não existem elementos repetidos na árvore binária de pesquisa. Para isso percorrem-se os valores da árvore enquanto são menores que k, e retorna-se o maior elemento encontrado até lado.

**1.2) Objectivo:** Realizar O método *public static int countLeavesAtLevel(Node<Integer> root, int k)* que retorna o número de folhas na árvore binária de pesquisa com raíz root que se encontram no nível k. Chamamos recursivamente o método, decrementando o k, até este ser zero. Se o k chegou ao zero e os filhos da root não são nulos, entao podemos incrementar o contador. Assim sucessivamente.

**1.3) Objectivo:** realizar o método *public static <E> boolean isBalanced(Node<E> root)* que verifíca se a árvore binária, referenciada por root, é balanceada. Para uma árvore ser balanceado, a difença entre as aulturas de cada ramo da árvore tem de ser no máximo 1. Para isso calculamos a altura do ramo esquerdo e do ramo direito através de um método auxiliar. De seguida, vemos se a diferença, em módulo, é ou não menor ou igual que um. Em caso afirmativo retornamos true.

**2.**  Realize a classe *AutoCompleteUtils*, com métodos utilitários para a criação de uma aplicação, semelhante à existente nos telemóveis, que sugira palavras, completando automaticamente palavras parcialmente escritas.

**2.1) Objectivo:** implementar o método *public static TNode loadWordsFromFile(TNode root, String fileName)* que armazena, numa árvore n-ária referenciada por root todas as palavras que ocorrem no ficheiro detexto. A nossa extrutura Tnode contém um array de caminhos para formas uma palvra, a letra corrente e uma variável booleana que nos diz se no momento estmos perante uma palavra ou não. Admitimos que os caracteres possíveis são apenas as letras de ‘a’ a ‘b’, assim torna se mais fácil guardar as referências para as prxímas letras. Para este exercício, lemos as palavras de um ficheiro e adicionamo-las ao TNode passado por parâmetros.

**2.2) Objectivo:** criar o método public static TNode longestWithPrefix(TNode root, String prefix) que retorna a referência para o nó da árvore n-ária referenciada por root que contenha o maior prefixo da palavra prefix. Para isso, percorremos a root pelo caminho definido pela String prefix. No momento é que o caminho é interrompido, retorna a útlima refência encontrada diferente de null.

**2.3) Objectivo:** implementar o método public static int countPossibleWords(TNode root, String prefix) que retorna o número de palavras que tenham como prefixo a palavra prefix e que existam na árvore n-ária referenciada por root. Inicialmente, temos de seguir o caminho do prefixo outra vez, mas desta vez temos de chegar ao fim desse prefixo, se não conseguirmos, retornamos zero. Caso consigamos, a partir da última letra do prefixo temos vários caminhos a percorrer. Assim contamos as palavras todas, passando por todos os caminhos possíveis.

**3.**

Recendo um array de vértices, ordenamos todas as suas adjacências em cada um dos vértices, usando o selection sort. De seguida adicionamos essas adjacencias a uma estrutura pair. Esta irá servir para criar um minHeap e estar sempre perante um peso mais baixo dos caminhos do grafo.

**Problema**