UNIVERSIDADE FEDERAL DO MARANHÃO
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
ESTRUTURA DE DADOS II
Professor: João Dallyson
Aluno: Fabrício de Jesus Costa

RELATÓRIO DA 2ª ATIVIDADE PRÁTICA

São Luís
2019

UNIVERSIDADE FEDERAL DO MARANHÃO
CCET - CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
São Luís, 31 de maio de 2019
Disciplina: Estrutura de dados II Prof: João Dallyson
Aluno: Fabrício de Jesus Costa Turma: 2019.1

Relatória da Atividade Prática II

Introdução

Operações de busca, inserção e remoção podem ser muito custosos em estruturas de dados lineares como listas, arranjos ou vetores, para solucionar esse problema estruturas como árvores de busca foram pensados, mas mesmo com essa solução fez-se necessário aprimorar ainda mais essas estruturas para que as operações sejam realizadas de forma eficiente, pois as árvores de busca podem se degenerar ou desbalancear, assim piorar a eficiência dos algoritmos aplicados nesta árvore, no entanto surgiram os conceitos de árvores balanceadas como as árvores AVL, Rubro-negra e a árvore B, que vêm se mostrando muito úteis.

A árvore AVL utiliza o fator de balanceamento do nó para verificar se há desbalanceamento e se necessário aplicar rotações (simples, à esquerda ou direita, ou dupla) assim a árvore fica o mais próximo do balanceamento, as árvores AVL não são garantia de balanceamento.

A árvore rubro-negra semelhante a árvore AVL, também realiza rotações, a diferença fica por conta de um atributo dos nós dessa árvore, que são ou vermelhos ou pretos, existem propriedades que devem ser satisfeitas e caso sejam violadas implicam em rotações para manter o balanceamento da árvore.

A árvore B é uma árvore estritamente balanceada onde cada nó pode possuir um número mínimo ou máximo de filhos ou chaves determinado por t onde t é o número mínimo e 2t - 1 o número máximo, essa árvore realiza operações de divisão de nó ou concatenação de nó, garantindo o balanceamento da árvore.

Nesta unidade vimos que se precisamos realizar operações de inserção, busca, remoção com um custo menor do que em estruturas de dados lineares como vetores/arranjos podemos utilizar técnicas nessas mesmas estruturas de dados como o hashing que utiliza de uma função para calcular a posição do elemento com base na chave dada.

O Programa

A atividade consiste na implementação dos algoritmos de hashing e árvores balanceadas visto em sala de aula aplicados em um programa que recebe como entrada arquivos de textos, o programa lê esses arquivos e conta a ocorrência de cada palavra ignorando palavras de tamanho menor ou igual a 3, ele permite a busca de palavras, se as encontrar incrementa a ocorrência da mesma, se não, adiciona a palavra à estrutura de dados utilizada. O algoritmo permite listar as palavras encontradas com base em suas ocorrências no texto, em ordem decrescente.

As estruturas de dados implementadas foram: Hash(de encadeamento aberto), Árvore AVL, Árvore Rubro-negra e Árvore B.

Os algoritmos de hashing foram implementados da seguinte forma: três classes foram criadas, uma classe que descreve o slot (Node) onde ficam os dados e a chave, uma classe Table uma lista de Node que descreve cada posição da tabela possui uma função (full) que informa se a lista está cheia e a classe Hash que descreve uma lista de Table e possui as função function(int tamanho, int chave, bool estado) essa função é responsável por calcular a posição da chave na tabela, função insert(node no) que insere um elemento na tabela usando a função function() para calcular a posição do elemento na tabela, a função search(string palavra) procura a palavra na tabela se não encontrar insere, função imprimir() que retorna uma lista ordenada de palavras e ocorrências com base na ocorrência.

Os algoritmos de Árvore AVL foram implementados dessa maneira: Uma classe AVLNode que descreve cada nó da árvore, classe AVLTree que realiza as operações na árvore usando a classe AVLNode, funções implementadas na AVLtree: inserção, que tem base da inserção da árvore binária de busca, search que procura a palavra e incrementa a sua ocorrencia (essa função todas as estruturas de dados possuem assim como inserção e imprimir que é semelhante entre as árvores), imprimir que retorna uma lista ordenada.

A Árvore rubro-negra está implementada com duas classes sendo uma a classe RBNode que é o nó onde ficam armazenados os dados e a chave, a outra classe implementa todas as funções, inserção, busca e impressão.

A Árvore B está implementada com duas classes uma para o nó e outra para as funções de inserção, busca e impressão.

A lista ordenada retornada pela função ‘imprimir’ é ordenada pelo algoritmo heapsort que foi discutido em sala de aula na primeira unidade e foi anteriormente (na atividade prática 1) implementado, porém esse algoritmo de heapsort está ordenando em ordem decrescente usando o ‘minheapfy’ que está sendo chamado heapfy somente.

Um arquivo py auxiliar contém funções úteis para a execução do programa tais quais: mergesort (utilizado na árvore B), heapsort (previamente explicado) e algoritmos de manipulação de strings (contagem de ocorrências, tokenização, etc...), nesse arquivo ocorrem as operações de transformação de letras maiúsculas para minúsculas e a inserção de cala palavra do texto em uma lista.

Análise de execução

No geral o algoritmo funciona instantaneamente com todos os textos apresentados, foram criados 3 arquivos de textos com números variados de palavras, sendo o menor o ‘test1.txt’ e o maior o ‘loremipsum.txt’.

Em questões gerais o hash pode, no pior caso, ter o pior desempenho entre as estruturas de dados apresentadas.

Foi muito difícil determinar qual algoritmo tem o melhor desempenho pois com os textos que foram trabalhados o retorno foi instantâneo, mas de acordo com o que foi estudado a ideia é que a árvore B tem o melhor desempenho.

Conclusão

Algoritmos de espalhamentos (hashing) e as árvores balanceadas são eficiente nas tarefas de inserção, busca e remoção, são muito úteis pois tornam o processo rápido, têm sido muito utilizadas em banco de dados e eu outras áreas da computação.