PUC-Rio Departamento de Informática
Prof. Marcus Vinicius S. Poggi de Aragão
Período: 2015.1 20 de maio de 2015
Data da Entrega: 20 de junho de 2015

ESTRUTURAS DE DADOS AVANÇADAS (INF 1010)

2^o Trabalho de Implementação

Descrição

A entrega do trabalho consiste de:

- OBRIGATÓRIO: Um e-mail para poggi@inf.puc-rio.br com ASSUNTO (ou SUBJECT) EDA151T2 contendo os arquivos correspondentes ao trabalho. O NÃO ENVIO DESTE E-MAIL, COMO SOLICITADO, IMPLICA QUE O TRABALHO NÃO SERÁ CONSIDERADO.
- Um documento contendo o roteiro de desenvolvimento dos algoritmos (e dos códigos), os itens pedidos, comentários e análises sobre a implementação e os testes realizados(papel).
- A impressão dos trechos RELEVANTES dos códigos fonte (papel).
- O trabalho pode ser feito em grupo de até 3 alunos.

Este trabalho prático consiste em desenvolver códigos para representar grafos e executar algoritmos para identificar elementos e características destes grafos e, principalmente, analisar o desempenho das implementações destes algoritmos com respeito ao tempo de CPU. O desenvolvimento destes códigos e a análise devem seguir os seguintes roteiros:

- Descrever os algoritmos informalmente.
- Demonstrar o entendimento do algoritmo explicando, em detalhe, o resultado que o algoritmo deve obter e justificá-lo.
- Apresentar as tabelas contendo as características obtidas, além dos tempos de execução utilizados pelos algoritmos sobre as instâncias testadas.
- Documente o arquivo contendo o código fonte de modo que cada passo do algoritmo esteja devidamente identificado e deixe claro como este passo é executado.
- Para a medida de tempo de CPU das execuções utilize as funções disponíveis no link correspondente na página do curso, um exemplo de utilização é apresentado. Quando o tempo de CPU for inferior à 5 segundos, faça uma repetição da execução tantas vezes quantas forem necessárias para que o tempo ultrapasse 5 s (faça um while), conte quantas foram as execuções e reporte a média.

- Obrigatoriamente apresente as soluções obtidas, ou uma caracterização destas soluções, no caso de serem muito extensas.
- Obrigatoriamente apresente tabelas contendo uma coluna para cada algoritmo aplicado às instâncias, com o tempo de CPU utilizado. Cada linha da tabela é associada a uma instância e contém a identificação da mesma. Nesta tabela coloque as instâncias em ordem crescente de tamanho.

A corretude código será testada sobre um conjunto de instâncias que será distribuido. A descrição das instâncias estão em anexo aos arquivos disponibilizados.

O problema a ser tratado pelos algoritmos a serem implementados é descrito a seguir.

- ***Busca, Inserção e Remoção***: Dada uma sequencia de subconjuntos do conjunto $\{1, 2, \ldots, n\}$, onde é indicado para cada subconjunto na sequência se a operação é de busca ou de inserção, fazer as respectivas buscas, sinalizando se foi encontrado ou não, inserções e remoções.

As estruturas e suas funções de inserção, busca e remoção devem ser testadas sobre os dados disponibilizados no site. Observe que caso um conjunto já existe na estrutura, este não é inserido novamente. Entretanto um contador de repetições deve fazer parte da estrutura e conter, ao longo da execução, quantas cópias de um conjunto permanecem na estrutura. No caso em que a remoção de um conjunto é requisitada e este tem apenas um cópia na estrutura,

Árvores de Busca

Algoritmos a testar/implementar:

1. Árvore de Busca sem Balanceamento

   Procedimentos: *abs_insere*, *abs_busca* e *abs_remove*.

2. Árvore α

   Procedimentos: *abα_insere*, *abα_busca* e *abα_remove*.

   Descrição: Árvore α é uma árvore binária de busca onde a sub-árvore com raiz em um nó x possui $size[x]$ elementos. Seja α uma constante tal que $1/2 \leq \alpha < 1$. Um nó x da árvore é dito balanceado se $size[left[x]] \leq \alpha.size[x]$ e $size[rigth[x]] \leq \alpha.size[x]$ onde $left[x]$ e $right[x]$ são os nós filhos à esquera e à direita do nó x, respectivamente. A árvore de busca é **α-balanceada** se todos os seus nós são α-balanceados.

   (a) Dado um nó x, arbitrário, mostre como reconstruir sua sub-árvore de modo que fique 1/2-balanceada. O algoritmo deve executar em $O(size[x])$ (que é $\Theta(size[x])$). Lembre que a sub-árvore de x já é uma sub-árvore de busca e $size[v]$ diz quantos elementos estão na sub-árvore de v.

   (b) Assuma que $\alpha > 1/2$ (ou seja é estritamente maior que 1/2). As operações de *Insere* e *Remove* são implementadas da forma habitual exceto que, quando algum nó não está mais α-balanceado, faz-se a operação do item a) (reconstrução para deixar 1/2-balanceada) é feita no nó desbalanceado de maior nível.

3. Árvore AVL

   Procedimentos: *avl_insere*, *avl_busca* e *avl_remove*.

4. Árvore B

   Procedimentos: *avB_insere*, *avB_busca* e *avB_remove*.

Implemente as 4 estruturas acima e determine qual a mais eficiente para cada instância executada. Verifique se existe uma estrutura dominante. Se existir procure explicar porque é a mais eficiente.

ESTE TRABALHO TEM COMO MAIOR OBJETIVO OS EXPERIMENTOS COM OS ALGORITMOS.

Portanto, apresente um documento que seja O MAIS DETALHADO POSSÍVEL no relato dos experimentos e na análise dos seus resultados. É permitido utilizar códigos prontos, desde que possam ser compilados e executados. O foco é no experimento.