

PUC-Rio
Departamento de Informática
Prof. Marcus Vinicius S. Poggi de Aragão
Horário: 2as-feiras e 4as-feiras de 9 às 11 horas - Sala 422L
28 de setembro de 2006
Data da Entrega: 25 de outubro de 2006
Período: 2006.2

ANÁLISE DE ALGORITMOS (INF 1721)

1º Trabalho de Implementação

Descrição

Este trabalho prático consiste em desenvolver códigos para diferentes algoritmos e estruturas de dados para resolver os problemas descritos abaixo 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.
- Explicar a fundamentação do algoritmo e justificar a sua corretude. Apresentar e explicar a complexidade teórica esperada para cada algoritmo.
- 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.

A corretude código será testada sobre os conjuntos de instâncias distribuídos. O trabalho entregue deve conter:

- Um documento contendo o roteiro de desenvolvimento dos algoritmos (e dos códigos), os itens pedidos acima, comentários e análises sobre a implementação e os testes realizados (papel). **É parte importante do documento uma tabela contendo os tempos de CPU de cada um dos 7 algoritmos pedidos (descritos abaixo) para cada instância dada.**
- Um e-mail para poggi@inf.puc-rio.br deve ser enviado contendo os códigos fonte, os executáveis correspondentes e o documento final (é obrigatório o uso do ASSUNTO (ou SUBJECT) AA062T1). (Lembrem que o gmail não aceita .exe, e portanto vc deve modificar a terminação de .zip para .txt).
- O trabalho pode ser feito em grupo de até 3 alunos.

0. Estruturas de Dados

O grupo deve implementar (ou usar códigos prontos) códigos para efetuar as seguintes operações nas estruturas de dados abaixo:

1. *d-Heap*
2. *Heap* que permita a união de *heaps* (*Leftist Heap* do Knuth), com e sem operações *preguiçosas* (**LAZY**). (A documentação para isto estará disponível na página no curso).

1. Problema da Árvore Geradora Mínima (veja os links “Instâncias para AGM” para grafos de teste)

1. Implementar o Algoritmo de Kruskal utilizando uma ordenação de complexidade $O(n \log n)$ (*Merge Sort* ou *Heap Sort*, por exemplo) e com implementações de *Union & Find*:
 - (a) sem a compactação dos ponteiros.
 - (b) com a compactação dos ponteiros.

2. Implementar o Algoritmo de Prim utilizando as estruturas de dados, listadas a seguir, para selecionar o vértice mais próximo da árvore corrente. Nestas estruturas, cada vértice tem como valor-chave o peso da menor aresta que o conecta à árvore corrente.

Lista de estruturas de dados a utilizar:

- (a) *d-Heap*, para diferentes valores de d .
 - (b) *Leftist Heap* sem *lazy*.
 - (c) *Leftist Heap* com *lazy*.
3. Implementar o Algoritmo de Round-Robin (Tarjan) (equivalente ao algoritmo de Solin ou Borůvka) nesse algoritmo inicia-se com uma árvore associada a cada vértice (n árvores) armazenado-se numa *min heap* as arestas que ligam cada árvore ao restante do grafo. A cada iteração uma árvore é conectada a uma outra e suas *min heaps* combinadas. A ordem em que as árvores são combinadas segue o critério FIFO onde a ordem inicial é arbitrária (1,2,...,n por exemplo). Utilize as seguintes *heaps* com operação de união:
 - (a) *Leftist Heap* sem *lazy*.
 - (b) *Leftist Heap* com *lazy*.