PUC-Rio
Departamento de Informática
Prof. Marcus Vinicius S. Poggi de Aragão
Horário: 3as-feiras de 13 às 16 horas - Sala 154L
26 de abril de 2009
Data da Entrega: 26 de maio de 2009
Período: 2009.1

PROJETO E ANÁLISE DE ALGORITMOS (INF 2926)

1^o 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.
- Apresentar as tabelas dos tempos de execução obtidos pelos algoritmos sobre as instâncias testadas, comparando sua evolução com a evolução dos tempos seguindo a complexidade teórica correspondente.
- 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.

A corretude código será testada sobre um conjunto de instâncias que será distribuido. 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).

- A impressão dos trechos relevantes dos códigos fonte (papel).
- Um e-mail para poggi@inf.puc-rio.br (é obrigatório o uso do ASSUNTO (ou SUBJECT) PAA091T1 deve ser enviado contendo os arquivos correspondentes ao trabalho. O NÃO ENVIO DESTE E-MAIL IMPLICA QUE O TRABALHO NÃO SERÁ CONSIDERADO.
- O trabalho pode ser feito em grupo de até 5 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. Árvore Balanceada (Árvore AVL, Árvore Vermelha-Preta, etc.)
2. *Heap* que permita a união de *heaps* (*Leftist Heap* do Knuth), com e sem operações *preguiçosas* (**LAZY**) – Alternativamente pode ser utilizada uma Heap de Fibonacci, que também utiliza operações *preguiçosas*.

1. Problema da Árvore Geradora Mínima

1. 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. (ver links na página do curso para textos sobre algoritmos para a MST, em especial para o algoritm de Round-Robin, ver também links para instâncias do problema).

   Lista de estruturas de dados a utilizar:

   (a) Árvore Balanceada de Busca

   (b) *Heap* sem *lazy* ou Heap de Fibonacci.

   (c) *Heap* com *lazy* ou Heap de Fibonacci.

   A *Heap* sugerida para ser implementada é a *Leftist Heap* ver texto no link heap na página do curso. A *Heap de Fibonacci* está descrita no livro "Introduction to Algorithms", T.H. Cormen, C.E. Leiserson e R.L. Rivest, McGraw-Hill.

2. Implementar o Algoritmo de Round-Robin (Tarjan) (equivalente ao algoritmo de Solin ou Borüvka) 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 árvore 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) *Heap* sem *lazy* ou Heap de Fibonacci.

   (b) *Heap* com *lazy* ou Heap de Fibonacci.

2. Problema da Mochila Fracionária (pode-se colocar parte de um objeto na mochila)

[KP-frac] Dado um conjunto de n objetos divisíveis com pesos positivos w_j, $j = 1, \ldots, n$ e valores também positivos v_j, $j = 1, \ldots, n$. Sabendo que uma mochila tem a capacidade W, determinar os objetos que podem ser levados na mochila cuja soma dos valores é máxima.

1. Implementar algoritmos para o problema da mochila fracionária com as seguintes complexidades teóricas em função do número n de itens candidatos a serem colocados na mochila:
   (a) $O(n \, log \, n)$
   (b) $O(n)$
   (c) Considere que o seu algoritmo do item (b) utiliza particionamentos em sequencia com pivot calculado apropriadamente para garantir a complexidade $O(n)$. Utilize agora como pivot o valor calculado pela expressão:

$$pivot \;\; = \;\; \frac{1}{|K|} \sum_{j \in K} \frac{v_j}{w_j}$$

   onde K é o conjunto de itens considerados.
      i. Prove que a complexidade (pior caso) do algoritmo resultante é $O(n^2)$.
      ii. Estime sua complexidade sobre as intâncias testadas.
      iii. Assim como para os itens (a) e (b) apresente experiências computacionais comparativas.

3. Encontrar os componentes fortemente conexos de um grafo orientado.

1. Implementar algoritmos com a seguinte complexidade::
   (a) $O(n^2 + nm)$
   (b) $O(|C|.(n + m))$ onde C é o conjunto de componentes fortemente conexos.
   (c) $O(n + m)$
2. Utilize esse algoritmo para determinar se uma instância do problema $2-SAT$ pode ser satisfeita ou não.