PUC-Rio
Departamento de Informática
Prof. Marcus Vinicius S. Poggi de Aragão
Horário: 5as-feiras de 13 às 16 horas - Sala 511 RDC
9 de maio de 2013
Data da Entrega: 2 de junho de 2013
Período: 2013.1

PROJETO E ANÁLISE DE ALGORITMOS (INF 2926)

1^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) PAA131T1 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 3 a 5 alunos.

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.

- Obrigatoriamente apresente tabelas contendo três de colunas para cada algoritmo aplicado às instâncias, uma com o valor da complexidade teórica, uma com o tempo de CPU utilizado e uma com a razão destes dois valores. 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.

1. Problema de Programação Hiperbólica 0-1 Irrestrito (PPH)

- *PPH*: Dado um conjunto de pares ordenados $\{(a_1, b_1), \ldots, (a_n, b_n)\}$ e um par obrigatório (a_0, b_0), onde $a_i, b_i \in Z^+, \quad \forall\, i = 0, 1, \ldots, n$, determinar $S \subseteq N$ onde $N = \{1, \ldots, n\}$ que maximiza:

$$R(S) = \frac{a_0 + \sum\limits_{t \in S} a_t}{b_0 + \sum\limits_{t \in S} b_t}$$

Considere o seguinte lema.

Lemma 1 *Seja R^* o valor da razão máxima obtida para o (PPH) e $S*$ um subconjunto de N tal que $R(S^*) = R^*$. Então, um par t pertence a qualquer S^* se e somente se $a_t/b_t > R^*$.*

Por que o lema acima é verdadeiro ?

Utilize este lema para projetar 3 algoritmos para encontrar R^* e S^*.

1. O primeiro algoritmo inicia com $R = a_0/b_0$ e testa repetidamente se existe algum par (a_k, b_k) que satisfaz às condições do lema. No caso afirmativo, inclui o par no conjunto S, atualiza o valor de R e repete o teste. Observe que se existir um elemento em S que não satisfaz às condições do lema, este elemento deve ser removido.

   Este primeiro algoritmo deve executar em $O(n^2)$.

2. Que relação tem o PPH com o problema de ordenação ? Utilize esta observação para projetar um algoritmo de complexidade $O(n\ log\ n)$

3. Observe novamente a relação do PPH com o problema de ordenação. O que caracteriza o conjunto S^* ? Esta caracterização permite projetar um algoritmo de complexidade $O(n)$. Apresente este algoritmo.

4. Considere que o seu algoritmo do item (4) utiliza particionamentos em sequencia com pivot calculado apropriadamente para garantir a complexidade $O(n)$. Utilize agora como pivot o valor calculado pela expressão:

$$pivot \;=\; \frac{a_0 + \sum_{t \in K} a_t}{b_0 + \sum_{t \in K} b_t}$$

onde K é o conjunto de todos os itens sendo considerados.

   (a) Prove que a complexidade (pior caso) do algoritmo resultante é $O(n^2)$.

   (b) Estime sua complexidade sobre as intâncias testadas.

   (c) Assim como para os itens (a) e (b) apresente experiências computacionais comparativas.

Novamente implemente os algoritmos acima e determine qual o mais eficiente para cada instância executa. Indique para que faixas de valores de n cada algoritmo é o mais eficiente.

2. 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.