

PUC-Rio
Departamento de Informática
Prof. Marcus Vinicius S. Poggi de Aragão
Horário: 3as-feiras de 13 às 16 horas - Sala 154L
6 de outubro de 2010
Data da Entrega: 3 de novembro de 2010
Período: 2010.2

PROJETO E ANÁLISE DE ALGORITMOS (INF 2926)

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.
- 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 distribuído. 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) PAA102T1 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 4 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 algoritmo 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*, no livro R.E. TARJAN, *Data Structures and Network Algorithms*, SIAM, 1983, e 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 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) *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 , $j = 1, \dots, n$ e valores também positivos v_j , $j = 1, \dots, n$. 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 sequência 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 instâncias testadas.
- iii. Assim como para os itens (a) e (b) apresente experiências computacionais comparativas.

3. Controle de Qualidade na Produção de Frascos de Vidro

O controle de qualidade de uma fábrica de frascos de vidro precisa determinar o maior nível de impacto que os seus frascos podem receber sem quebrar. Para este fim, a seção de controle de qualidade possui uma longa escala de alta precisão postada verticalmente. O valor na escala (x) a partir do qual o frasco quebra é o que determina a resistência de cada tipo de produto oferecido pela fábrica. Considere que esta escala possui comprimento total n . Se para determinar x , onde $1 \leq x \leq n$, somente um frasco está disponível, o único algoritmo que garante a determinação de x consiste em provocar a queda do frasco do valor 1, depois do 2, e assim por diante até o frasco quebrar com a sua queda. O número de quedas provocadas neste algoritmo é $O(n)$.

O grupo deve projetar e implementar algoritmos que simulem cada queda provocada até a determinação do valor em que o frasco quebra. A entrada do algoritmos é o valor de x (a resposta!) que será usado para determinar se o frasco vai quebrar ou não com a sua queda. O algoritmo deve executar até provar que a altura em que o frasco quebra é x .

Suponha agora que para um tipo de frasco, exista mais de um frasco disponível para determinar o valor a partir do qual quebrará. Examine os casos abaixo, proponha e implemente os algoritmos para “determinar” o valor x .

1. Considere que 2 frascos estão disponíveis. Proponha para este caso um algoritmo onde o número de quedas provocadas dos frascos é $O(\sqrt{n})$ e que a determinação do valor x é garantida.

2. Mostre que para k frascos disponíveis é possível propor um algoritmo onde o número de quedas é, no pior caso, $k \cdot n^{\frac{1}{k}}$. Apresente os algoritmos para $k = 3$ e $k = 4$. Em seguida generalize.
3. Qual a menor complexidade assintótica possível para um algoritmo determinar o valor de x ? Proponha um algoritmo que tenha essa complexidade. Quantos frascos são necessários para que esta complexidade seja atingida?

O grupo deve implementar e testar experimentalmente 5 algoritmos (para 1, 2, 3 e 4 frascos disponíveis e para um número ilimitado deles) apresentando estatísticas do tempo de CPU das execuções para cada conjunto de valores de x . Estes serão correspondentes ao número de bits utilizados para representar o valor máximo do conjunto (n é este valor máximo). Haverá conjuntos de 32, 64, 128, 192 e 256 bits. Estas instâncias do problema serão disponibilizadas na página *web* do curso.