PUC-Rio
Departamento de Informática
Prof. Marcus Vinicius S. Poggi de Aragão
Horário: 4as-feiras de 13 às 16 horas - Sala 511 RDC
26 de abril de 2017
Data da Entrega: 22 de maio de 2017
Período: 2017.1

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:

- Um e-mail para poggi@inf.puc-rio.br (é obrigatório o uso do ASSUNTO (ou SUBJECT) PAA151T1 deve ser enviado contendo os arquivos correspondentes ao trabalho. O NÃO ENVIO DESTE E-MAIL IMPLICA QUE O TRABALHO NÃO SERÁ CONSIDERADO.
- 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. 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) PAA171T1 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.

1. Problema de Caminho-mais-Curto.

1. Implementar o algoritmo de Dijkstra para encontrar o caminho mais curto entre um vértice *fonte* e os demais vértices de um grafo orientado $G = (V, E)$ onde a distância associada a um arco $e \in E$ é dada por um inteiro positivo $\ell_e = \ell_{vw}$ onde v e w são o vértice de partida e de chegada do arco e, respectivamente. (As complexidades devem ser apresentadas em função de $n = |V|$ e $m = |E|$).

   Algoritmo Dijkstra (s - *fonte*)

Passo 0: *Inicialização*

Seja S o conjunto de vértices com caminho mais curto a partir de s determinado, e $\overline{S}$ seu complemento ($\overline{S} = V - S$).

$S \leftarrow \emptyset$

$d(i) \leftarrow +\infty \; \forall \, i \in V$;

$d(s) \leftarrow 0$; $pred(s) \leftarrow 0$;

Passo 1: *Iteração*

Enquanto $S \subset V$ faça

1.1 Encontre $v \in \overline{S}$ t.q. $d(v) = \min\limits_{w \in \overline{S}} d(w)$

1.2 $S \leftarrow S \bigcup \{v\}$; $\overline{S} \leftarrow \overline{S} \setminus \{v\}$;

1.3 Para todo $w \in \Gamma^+(v)$
Se $d(v) + \ell_{vw} < d(w)$
então $d(w) \leftarrow d(v) + \ell_{vw}$; $pred(w) \leftarrow v$;

Lista de estruturas de dados a utilizar para armazenar os valores $d(i)$, $i = 1, \ldots, n$ nos passos **1.1** e **1.3**:

(a) Um vetor.

(b) Árvore Balanceada de Busca (AVL, ou vermelha e preta).

(c) Heap de Fibonacci.

(d) *Buckets* Assuma que $C = \max_{(v,w)\in E} l_{vw}$ (qual a maior distância possível em um caminho ?). Observe que a distância $d(v)$ do vértice selecionado no passo **1.1** a cada iteração é maior ou igual à do vértice selecionado na iteração anterior. Considere que *buckets* (ou caixas) são criadas para os valores de 0 a nC. Cada *bucket* utiliza uma lista duplamente encadeada para armazenar os vértices v tais que $d(v)$ tem o valor correspondente ao do *bucket*. Um vetor (d) é utilizado para indicar em que *bucket* o vértice está armazenado.

(e) Árvore α

Esta estrutura de dados é descrita a seguir. O grupo deve responder às questões nesta descrição.

Considere 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.

i. 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.

ii. Mostre que a operação de busca em uma árvore α-**balanceada** com n elementos é feita em $O(log\ n)$. (Fácil !!)

iii. ESTE ITEM NÃO É PARA SER RESPONDIDO ! A partir deste item assuma que $\alpha > 1/2$ (ou seja é estritamente maior que 1/2). Suponha que as operações de *Insert* e *Delete* são implementadas da forma habitual, sem rotações, exceto que quando algum nó não está mais α-balanceado, a operação do item a) (reconstrução para deixar 1/2-balanceada) é feita no nó desbalanceado de maior nível.

A idéia é analisar este esquema de reconstrução utilizando o método do pontencial para análize *amortizada*. Para um nó x na árvore binária T define-se:

$$\Delta(x) = |size[left[x]] - size[right[x]]|$$

Define-se também o potencial na ávore T, $\Phi(T)$ como:

$$\Phi(T) = c \sum_{x\in T|\Delta(x)\geq 2} \Delta(x)$$

onde c é uma constante sufucientemente grande que depende de α.

iv. Argumente que qualquer árvore binária de busca tem potencial não-negativo e que uma árvore 1/2-balanceada tem potencial igual a ZERO.

v. Suponha que m unidades de potencial podem pagar pela reconstrução de uma sub-árvore com m nós. Que valor, em função de α, deve ter c para que a operação de reconstrução tome tempo *amortizado* constante ($O(1)$), quando é aplicada sobre uma sub-árvore que não está α-balanceada ?

vi. Mostre que as operações de *Insert* e *Delete* tomam tempo *amortizado* $O(log\ n)$.

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. Multiplicação de Polinômios

O grupo deve implementar 4 algoritmos para calcular o polinômio produto de outros dois polinômios. A entrada será dada pelos $n+1$ coeficientes de cada polinômio de grau n. A saída deverá ser os $2n+1$ coeficientes do polinômio produto. O algoritmos seguem:

1. Algoritmo que consiste da multiplicação direta dos polinômios de entrada ($O(n^2)$).
2. Algoritmo utilizando divisão-e-conquista ($O(n^{log_2 3})$).
3. Algoritmo utilizando a DFT e a FFT (*Fast Fourier Transform*) ($O(n \ log \ n)$).
4. Algoritmo de Jiang e Wu (paper disponibilizado)

Deve-se determinar qual o algoritmo mais eficiente para todos os valores de n para o conjunto de instâncias disponibilizado.