Estrutura de Dados - 2o. período de 2009

Gabarito da Primeira Avaliação à Distância

1. O algoritmo a seguir tem como entrada uma matriz M com n linhas e n colunas:

   $S1 \leftarrow M[1,1]$

   $S2 \leftarrow M[1,1]$

   para $i = 1 \ldots n$ faça

       para $j = 1 \ldots n$ faça

           se $M[i,j] < S1$

               então $S1 \leftarrow M[i,j]$

               senão se $M[i,j] > S2$ então $S2 \leftarrow M[i,j]$

   Responda:

   (a) (0,5) Que problema este algoritmo resolve?

   Resposta: Ele encontra os valores mínimo e máximo da matriz M, que são armazenados em $S1$ e $S2$, respectivamente.

   (b) (0,5) Qual é a complexidade deste algoritmo? Justifique.

   Resposta: A complexidade deste algoritmo é $O(n^2)$, já que a matriz possui n^2 elementos, e para cada um deles é executado um número constante de passos.

   (c) (0,5) Este algoritmo é ótimo? Justifique.

   Resposta: Sim. Para resolvermos este problema, precisamos ler todos os dados da entrada, que possui tamanho n^2. Logo, n^2 é um limite inferior para o problema.

2. Considere o seguinte algoritmo de ordenação de um vetor V cujas posições inicial e final são i e j, respectivamente, onde $i < j$:

   - Descubra a posição k onde o elemento mínimo de V se encontra.
   - Troque os conteúdos das posições i e k
   - Repita os dois passos acima, aplicando-os agora às posições $i + 1$ até j

   Resolva os itens a seguir:

   (a) (1,0) Escreva um algoritmo *recursivo* que implementa o processo acima.

   Resposta:

procedimento $ord(i, n)$
    se $i < n$ então
        $menor := i$
        para $j := i + 1$ até n faça
            se $V[j] < V[menor]$ então
                $menor := j$
        $aux := V[i]$
        $V[i] := V[menor]$
        $V[menor] := aux$
        $ord(i + 1, n)$

Chamada externa: $ord(1, n)$

(b) (1,0) Escreva um algoritmo *iterativo* que implementa o processo acima.

Resposta:

para $i := 1$ até n faça
    $menor := i$
    para $j := i + 1$ até n faça
        se $V[j] < V[menor]$ então
            $menor := j$
    $aux := V[i]$
    $V[i] := V[menor]$
    $V[menor] := aux$

(c) (1,0) Quais as complexidades dos algoritmos, em relação ao número de trocas?

Resposta: Ambos os algoritmos são $\Theta(n^2)$.

3. Considere uma busca linear não ordenada de um elemento x numa lista com 100 elementos, onde a probabilidade de x estar em qualquer posição na primeira metade da lista (posições de 1 a 50) é o dobro da probabilidade de x estar em qualquer posição na segunda metade da lista (posições de 51 a 100). Suponha ainda que a probabilidade de x não estar na lista é zero.

(a) (1,0) Qual é o número médio de comparações efetuadas numa busca de x?

Resposta: Seja p a probabilidade de busca de uma chave correspondente às entradas $\{E_1, E_2, \cdots, E_{50}\}$. Temos então que a probabilidade de busca de uma chave correspondente às entradas $\{E_{51}, E_{52}, \cdots, E_{100}\}$ é $p/2$. Logo, $p = 1/75$.

A expressão da complexidade média neste caso é dada por:

$$\begin{aligned} C.M. &= \frac{1}{75}(1 + 2 + 3 + \cdots + 50) + \frac{1}{150}(51 + 52 + 53 + \cdots + 100) \\ &= \frac{51}{3} + \frac{151}{6} \\ &\approx 42,17 \end{aligned}$$

(b) (1,0) Como você interpreta o resultado acima?

Resposta: Como os elementos que têm maior probabilidade de acesso estão na primeira metade da lista de 100 elementos, então a complexidade média da busca é menor do que 50.

4. Considere um volumoso cadastro de clientes, no qual as seguintes operações são executadas:

   1. Busca de um cliente no cadastro, por nome.
   2. Inserção de um novo cliente no cadastro.
   3. Remoção de um cliente do cadastro.

   Suponha ainda que, para cada cliente, existe um contador que é incrementado de uma unidade a cada vez que este é buscado no cadastro. Isto forma uma estatística dos acessos.

   Temos a seguir várias estruturas de dados que poderiam ser utilizadas para implementar o cadastro de clientes. Aponte uma possível vantagem e uma possível desvantagem de cada uma delas, em relação ao problema descrito.

   (a) (0,5) Lista linear não ordenada.

   Resposta: Vantagem: A inserção de um novo cliente no cadastro poderia ser feita em tempo constante ($O(1)$) (desconsiderando a busca prévia), pois o novo cliente poderia ser inserido no início da lista. Desvantagem: A busca por um cliente levaria, no pior caso n passos ($O(n)$) (inclusive os clientes mais acessados).

   (b) (0,5) Lista simplesmente encadeada, ordenada por nome.

   Resposta: Vantagem: No caso médio, a busca por um cliente levaria menos que n passos (embora seja $O(n)$), pelo fato de a lista estar ordenada por nome (na maioria das buscas, não seria necessário percorrer a lista até o final). Desvantagem: Clientes com pouco acesso viriam antes na lista do que clientes com muito acesso, aumentando o tempo de acesso aos clientes muito acessados.

   (c) (0,5) Lista simplesmente encadeada, ordenada pelo contador.

   Resposta: Vantagem: Os clientes mais acessados seriam os primeiros da lista, diminuindo a complexidade média das buscas na lista. Desvantagem: A inserção de um novo cliente seria feita em $\Theta(n)$, pois ele deve entrar no final da lista (já que um novo cliente não tem nenhum acesso contabilizado).

   (d) (0,5) Lista duplamente encadeada, ordenada pelo contador.

   Resposta: Vantagem: A inserção de um novo cliente seria feita em $O(1)$. Desvantagem: A busca por um cliente não presente na lista levaria sempre n passos (portanto, $\Theta(n)$ para qualquer entrada correspondente a fracasso na busca).

   (e) (0,5) Árvore Binária de Busca, organizada por nome.

Resposta: Vantagem: A busca de um cliente seria $O(\log n)$ (e não $O(n)$, como nas listas). Desvantagem: Clientes mais acessados poderiam ser folhas na árvore, ao invés de estarem em níveis mais altos da árvore, aumentando o tempo de acesso a eles.

5. (2,0) Um micro-processador executa vários programas, da seguinte forma: enquanto um deles está sendo executado, os outros ficam numa fila de espera. A cada 50 microssegundos, o programa em execução pode voltar para o último lugar na fila de espera (caso ainda reste processamento para ele) ou ser encerrado (caso contrário); imediatamente, o primeiro programa na fila de espera passa a ocupar o processador pelos próximos 50 microssegundos.

   Suponha que cada novo programa que entra no sistema para ser executado é colocado sempre no último lugar da fila de espera.

   Suponha também que, quando um programa encerra seu processamento antes dos 50 microssegundos regulamentares, o primeiro programa da fila de espera passa a ocupar o processador imediatamente.

   Escreva um algoritmo que, inicialmente, leia uma sequência de n triplas $(a_1, t_1, p_1,)$, $(a_2, t_2, p_2), \ldots, (a_n, t_n, p_n)$, onde a_i representa a identificação de um programa, t_i o instante de tempo em que ele entra no sistema pela primeira vez, e p_i a duração total do programa. (Suponha que t_i e p_i são dados em microssegundos, e que $t_1 < t_2 < \ldots < t_n$.)

   A seguir, o programa lê um número t de microssegundos e exibe o estado atual do sistema: programa em execução, programas na fila de espera (em ordem), e tempo restante de processamento para cada um deles.

   Resposta: Seja F uma fila encadeada que contém as n triplas. Assume-se que, durante a execução do sistema, uma vez que uma tripla foi reinserida no final da fila em um instante t, por necessitar de mais de 50 microsegundos, todas as demais triplas possuem $t_i < t$, ou seja, já entraram no sistema (para evitar inconsistência na fila, relativa a triplas que ainda nem entraram no sistema estarem antes de triplas que já foram executadas).

   <u>Algoritmo:</u>

   // construção da lista

   ocupar(pt)
   $pt \uparrow .a := a_1$
   $pt \uparrow .t := t_1$
   $pt \uparrow .p := p_1$
   $pt \uparrow .prox := \lambda$
   $ptlista := pt$
   $ultimo := pt$
   para $i := 2$ até n faça

```
        ocupar(pt)
        pt ↑ .a := a_i
        pt ↑ .t := t_i
        pt ↑ .p := p_i
        pt ↑ .prox := λ
        ultimo ↑ .prox := pt
        ultimo := pt

// simulação do sistema
tempo := 0        // o sistema inicia no tempo 0
leia (tfinal)     // tempo t lido, para o qual será exibido o estado do sistema

enquanto tfinal > tempo faça
    se ptlista ↑ .t <= tempo então   //programa do início da fila já está no sistema
        se (tfinal − tempo) > 50 então
            se ptlista ↑ .p <= 50 então     //o processo sai do sistema
                tempo := tempo + ptlista ↑ .p
                pt := ptlista
                ptlista := ptlista ↑ .prox
                desocupar(pt)
            senão                           //o processo volta ao fim da fila
                ptlista ↑ .p := ptlista ↑ .p − 50
                pt := ptlista
                ptlista := ptlista ↑ .prox
                ultimo ↑ .prox := pt
                ultimo := pt
                pt ↑ .prox := λ
                tempo := tempo + 50
        senão
            se (tfinal − tempo) > ptlista ↑ .p então
                tempo := tempo + ptlista ↑ .p
                pt := ptlista
                ptlista := ptlista ↑ .prox
                desocupar(pt)
            senão
                ptlista ↑ .p := ptlista ↑ .p − (tfinal − tempo)
                tempo := tfinal       //força o fim da simulação
    senão
        tempo := ptlista ↑ .t    // o sistema aguarda até o processo entrar na fila

// impressão do estado do sistema
imprimir(Programa em execução:, ptlista ↑ .a, ptlista ↑ .t, ptlista ↑ .p)
ptlista := ptlista ↑ .prox
imprimir(Fila de espera:)
```

enquanto $ptlista \neq \lambda$ faça

$\quad$ imprimir$(ptlista \uparrow .a, ptlista \uparrow .t, ptlista \uparrow .p)$

$\quad$ $ptlista := ptlista \uparrow .prox$