MAC 5711 – Análise de Algoritmos

Primeiro Semestre de 2007

Primeira Prova – 23 de abril

Gabarito

1. [1,5 pontos]
   Marque F ou V para indicar se a afirmação é falsa ou verdadeira, respectivamente. Justifique sucintamente as suas respostas.

   (a) (V) $O(\lg n^k) = O(\lg n)$, onde k é uma constante positiva.
   Justificativa: Se $f(n) = O(\lg n^k)$, então existem constantes positivas c e n_0 tais que $f(n) \leq c \lg n^k$ para todo $n \geq n_0$. Como $\lg n^k = k \lg n$, temos que

   $$f(n) \leq ck \lg n \quad \text{para todo } n \geq n_0.$$

   Como k é uma constante positiva, concluímos que $f(n) = O(\lg n)$.

   (b) (V) $\log_{10} n = O(\lg n)$.
   Justificativa: Para todo $n \geq 1$, temos que

   $$\log_{10} n = \frac{\lg n}{\lg 10} \leq \lg n,$$

   pois $\lg 10 > 1$.

   (c) (F) $2^{O(\lg n)} = O(n)$.
   Justificativa: Note que $2 \lg n = O(\lg n)$. No entanto, $2^{2 \lg n} = n^2$, que não é $O(n)$.

2. [1,5 pontos]
Resolva a recorrência

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ T(n/2) + \lg n & \text{se } n = 2, 4, 8, 16, \ldots. \end{cases}$$

definida apenas para n potência de 2. Resolva a recorrência dando uma fórmula fechada para $T(n)$ (quando n é potência de 2). Note que uma fórmula fechada não tem somatórios.

Resolução: Desenrolando a recorrência, obtemos que

$$\begin{aligned} T(n) &= T(n/2) + \lg n \\ &= T(n/4) + \lg \frac{n}{2} + \lg n \\ &= T(n/8) + \lg \frac{n}{4} + \lg \frac{n}{2} + \lg n \\ &= T(1) + \sum_{k=0}^{\lg n - 1} \lg \frac{n}{2^k} \\ &= 1 + \sum_{k=0}^{\lg n - 1} (\lg n - k) \\ &= 1 + \sum_{k=0}^{\lg n - 1} \lg n - \sum_{k=0}^{\lg n - 1} k \\ &= 1 + \lg^2 n - \frac{\lg n(\lg n - 1)}{2} \\ &= 1 + \frac{\lg^2 n}{2} + \frac{\lg n}{2}. \end{aligned}$$

Agora podemos mostrar, por indução em n, que

$$T(n) = \frac{\lg^2 n}{2} + \frac{\lg n}{2} + 1$$

para todo $n \geq 1$ potência de 2:

Base: $n = 1$.

Para $n = 1$, temos que $\frac{\lg^2 n}{2} + \frac{\lg n}{2} + 1 = 1 = T(1)$.

Passo: $n \geq 2$ potência de 2.

Para $n \geq 2$ potência de 2, temos que

$$\begin{aligned} T(n) &= T(n/2) + \lg n \\ &= (\frac{1}{2} \lg^2 \frac{n}{2} + \frac{1}{2} \lg \frac{n}{2} + 1) + \lg n \quad \text{por indução} \\ &= \frac{1}{2}(\lg n - 1)^2 + \frac{1}{2}(\lg n - 1) + 1 + \lg n \\ &= \frac{1}{2}(\lg^2 n - 2 \lg n + 1) + \frac{3 \lg n}{2} + \frac{1}{2} \\ &= \frac{\lg^2 n}{2} + \frac{\lg n}{2} + 1. \end{aligned}$$

3. [3,0 pontos]
Seja S uma coleção de n inteiros. Dizemos que S tem um *representante majoritário* se há em S um inteiro m que aparece mais do que $\lfloor n/2 \rfloor$ vezes. Escreva um algoritmo

REPRESENTANTE-MAJORITÁRIO (v, n)

que recebe um inteiro n é um vetor $v[1\,..\,n]$ com uma coleção de n inteiros e devolve o representante majoritário se a coleção tiver um, ou NULL caso contrário. Seu algoritmo deve consumir tempo $O(n \lg n)$. Justifique sua resposta.

Resolução: Vamos usar divisão e conquista. A idéia é encontrar recursivamente o representante majoritário das duas metades do vetor (se existirem) e verificar se um deles é o representante majoritário do vetor todo. Note que o vetor tem um representante majoritário, então ele é o representante majoritário de pelo menos uma das metades do vetor.

```
REPRESENTANTE-MAJORITÁRIO (v, n)
 1.  devolva RMAJ-REC (v, 1, n)

RMAJ-REC (v, p, r)
 1.  se p = r
 2.    então devolva v[p]
 3.    senão q ← ⌊(p+r)/2⌋
 4.          m1 ← RMAJ-REC (v, p, q)
 5.          se m1 ≠ NULL e CONTA (v, p, r, m1) > (r − p + 1)/2
 6.            então devolva m1
 7.            senão m2 ← RMAJ-REC (v, q + 1, r)
 8.                  se m2 ≠ NULL e CONTA (v, p, r, m2) > (r − p + 1)/2
 9.                    então devolva m2
10.                    senão devolva NULL
```

O algoritmo CONTA recebe um vetor v, índices p e r e um elemento m e devolve o número de vezes que m aparece no vetor $v[p\,..\,r]$. É fácil implementar o algoritmo CONTA de modo que ele consuma tempo $\Theta(n)$, onde $n = r - p + 1$.

Assim sendo, o tempo consumido pelo algoritmo RMAJ-REC (v, p, r) nas linhas 3 a 10, descontado o tempo das chamadas recursivas, é $\Theta(n)$, onde novamente $n = r - p + 1$. Disso concluímos que o tempo de pior caso de RMAJ-REC (v, p, r) é dado pela recorrência $T(n) = 2T(n/2) + \Theta(n)$, cuja solução é $T(n) = \Theta(n \lg n)$.

4. [3,0 pontos]
Seja n o número de participantes em uma competição que consiste em duas provas: uma de natação e uma de corrida. Cada participante é identificado por um número em $\{1, \ldots, n\}$ e deve completar a sua prova de natação antes de começar a sua corrida. Apenas uma faixa da piscina está reservada para uso da competição, assim sendo os participantes não podem fazer suas provas de natação simultaneamente. Eles podem, porém, fazer suas corridas simultaneamente se necessário. Com isso, cada participante pode começar a sua corrida imediatamente após terminar sua prova de natação.

Para $i = 1, \ldots, n$, são dados inteiros positivos tn_i e tc_i representando os tempos previstos de prova de natação e de corrida do participante i. Queremos determinar uma ordem em que os participantes devem fazer a prova de natação de modo a minimizar a duração total da competição. A competição começa no momento em que o primeiro dos participantes começa a sua prova de natação e termina quando o último dos participantes conclui a sua corrida.

Um aluno, pensando em projetar um algoritmo guloso para esse problema, pensou em dois critérios gulosos que poderia aplicar para isso:

(a) Ordenar os participantes pelo valor de tn_i, em ordem crescente;

(b) Ordenar os participantes pelo valor de tc_i, em ordem decrescente.

Para cada um dos dois critérios, prove ou dê um contra-exemplo para a seguinte afirmação:

> O algoritmo que devolve os participantes ordenados por esse critério funciona.

Resolução:

O algoritmo derivado do critério (a) não funciona. Veja por exemplo o que ele faz para a instância com dois participantes, e com $tn_1 = tc_1 = 1$ e $tn_2 = tc_2 = 2$. Para tal instância, esse algoritmo coloca o participante 1 para nadar antes do participante 2, e com isso a conclusão da competição se dá em $tn_1 + tn_2 + tc_2 = 1 + 2 + 2 = 5$. Porém, se colocarmos o participante 1 para nadar depois do participante 2, temos um tempo de conclusão de $tn_2 + tc_2 = 4$, pois o participante 1 nada e corre enquanto o participante 2 corre. Essa segunda opção é a melhor para essa instância. Portanto o algoritmo derivado de (a) não funciona.

Já o algoritmo derivado de (b) funciona. Vamos mostrar uma prova em duas etapas. Primeiro mostraremos que todas as ordenações que seguem a ordem descrita em (b) têm o mesmo tempo de conclusão. Em tais ordenações, participantes com o mesmo tempo de corrida vêm em bloco, um atrás do outro, em uma ordem arbitrária. Note que se trocarmos a ordem de participantes em um mesmo bloco (ou seja, que têm o mesmo tempo de corrida), o tempo de conclusão não se altera. De fato, isso é verdade pois o último a terminar em cada um destes blocos é sempre o último a começar a nadar no bloco, e o último a nadar no bloco termina de nadar sempre no mesmo instante, independente da ordem no bloco. Ou seja, o tempo de conclusão não é afetado por trocas dentro de um mesmo bloco.

Dada uma ordem dos participantes, chamemos de inversão um par (i, j) de participantes em que i vem antes de j nessa ordem porém $tc_i < tc_j$. Seja O uma ordem ótima dos participantes que tem o menor número possível de inversões. Vamos mostrar que o número de inversões de O é zero. Suponha, por contradição, que não é 0. Neste caso, existe um par de participantes (i, j) tal que j vem logo depois que i na ordem O e $tc_i < tc_j$. Seja O' a ordem obtida de O trocando-se a posição de i e j. Claro que o momento de conclusão de cada participante distinto de i e j é o mesmo em O e em O'. Claro também que o momento de conclusão de j é menor em O' que em O, pois o participante começou a nadar mais cedo. Por outro lado, o momento de conclusão de i aumentou, e passou a ser $T + tp_j + tp_i + tc_i$, onde T é o momento em que i começa a nadar em O. Mas, em O, o momento de conclusão de j é $T + tp_i + tp_j + tc_j > T + tp_j + tp_i + tc_i$, ou seja, o maior dos momentos de conclusão em O' não é maior que o maior dos momentos de conclusão de O. Isso significa que O' é ótimo também. Porém isso é uma contradição à escolha de O já que O' tem uma inversão a menos.

5. [2,5 pontos]
Considere um vetor $v[1\,..\,n]$ com $n \geq 2$ números positivos distintos. Chame de continuidade discreta uma posição i do vetor, com $1 \leq i \leq n-1$, tal que $|v[i] - v[i+1]| \leq 1$. Considere o seguinte algoritmo que determina o número de continuidades discretas em um vetor $v[1\,..\,n]$ com $n \geq 2$ números positivos distintos.

Algoritmo CONTINUIDADES (v, n)

```
1.  c ← 0
2.  para i ← 1 até n − 1 faça
3.      se |v[i] − v[i + 1]| ≤ 1
4.          então c ← c + 1
5.  devolva c
```

Suponha que v é uma permutação de 1 a n escolhida ao acaso dentre todas as permutações de 1 a n, de acordo com a distribuição uniforme de probabilidade. Seja X o número de vezes que a variável c é alterada (ou seja, o número de execuções da linha 4 do algoritmo) numa chamada de CONTINUIDADES (v, n). Note que X é uma variável aleatória. Calcule o valor esperado de X.

Resolução: Seja X_i a variável aleatória que vale 1 se a linha 4 é executada na i-ésima iteração do **para** da linha 2, e vale 0 caso contrário. Claro que $X = \sum_{i=1}^{n-1} X_i$. Ou seja, para calcular $\mathrm{E}[X]$ basta calcularmos $\mathrm{E}[X_i]$.

Para cada i, temos que $\mathrm{E}[X_i] = \Pr[X_i = 1]$. Como v é uma permutação de 1 a n escolhida ao acaso dentre todas as permutações de 1 a n de acordo com a distribuição uniforme de probabilidade, temos que

$$\begin{aligned}
\Pr[X_i = 1] &= \frac{\#\text{ permutações em que } v[i] = v[i+1] \pm 1}{n!} \\
&= \frac{2(n-1)!}{n!} \\
&= \frac{2}{n}.
\end{aligned}$$

Mas então $\mathrm{E}[X] = \sum_{i=1}^{n-1} \mathrm{E}[X_i] = \frac{2(n-1)}{n} = 2 - \frac{2}{n}$.