MAC 5711 - Análise de Algoritmos

Departamento de Ciência da Computação
Segundo semestre de 2017

Lista 3

1. Seja $M(n)$ definida pela recorrência

$$\begin{aligned} M(0) &= 1 \\ M(1) &= 1 \\ M(n) &= \min_{0 \le k \le n-1} \{M(k) + M(n-k-1)\} + n \quad \text{para } n = 2, 3, 4, \ldots \end{aligned}$$

   Mostre que $M(n) \ge \frac{1}{2}(n+1)\lg(n+1)$ para todo $n \ge 0$.

2. Qual é o consumo de espaço do QUICKSORT no pior caso?

3. Quando um algoritmo recursivo tem como último comando executado, em algum de seus casos, uma chamada recursiva, tal chamada é denominada *recursão de cauda* (*tail recursion*). Um exemplo de recursão de cauda acontece no QUICKSORT.

   Toda recursão de cauda pode ser substituída por uma repetição. No caso do QUICKSORT, obtemos o seguinte:

```
QUICKSORT (A, p, r)
1   enquanto p < r
2         q ← PARTICIONE (A, p, r)
3         QUICKSORT (A, p, q − 1)
4         p ← q + 1
```

   Mostre como essa idéia pode ser usada (de uma maneira mais esperta) para melhorar significativamente o consumo de espaço no pior caso do QUICKSORT.

4. Considere o seguinte algoritmo que determina o segundo maior elemento de um vetor $v[1\,..\,n]$ com $n \ge 2$ números positivos distintos.

```
Algoritmo Máximo (v, n)
 1. maior ← 0
 2. segundo_maior ← 0
 3. para i ← 1 até n faça
 4.     se v[i] > maior
 5.         então segundo_maior ← maior
 6.               maior ← v[i]
 7.         senão se v[i] > segundo_maior
 8.                   então segundo_maior ← v[i]
 9. devolva segundo_maior
```

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 *segundo_maior* é alterada (ou seja, o número de execuções das linhas 5 e 8 do algoritmo) numa chamada de Máximo(v, n). Note que X é uma variável aleatória. Calcule o valor esperado de X.

5. Considere o seguinte algoritmo que calcula o maior e o menor elemento de um vetor $v[1\,..\,n]$ com elementos distintos.

```
Algoritmo MaiorMenor (v, n)
 1. maior ← v[1]
 2. menor ← v[1]
 3. para i ← 2 até n faça
 4.     se v[i] > maior
 5.         então maior ← v[i]
 6.         senão se v[i] < menor
 7.                 então menor ← v[i]
 8. devolva maior, menor
```

   Suponha que a entrada do algoritmo é uma permutação de 1 a n escolhida uniformemente dentre todas as permutações de 1 a n.

   Qual é o número esperado de comparações executadas na linha 6 do algoritmo? Qual é o número esperado de atribuições efetuadas na linha 7 do algoritmo?

6. **(CLRS 8.4-3)** Seja X uma variável aleatória que é igual ao número de caras em duas jogadas de uma moeda justa. Quanto vale $E[X^2]$? Quanto vale $E[X]^2$?

7. Tanto MERGESORTquanto QUICKSORTforam vistos para ordenar um vetor. Considere agora o caso em que os dados estão montados numa lista (simplesmente) ligada. É importante não piorar a complexidade dos algoritmos; deveriam ser mais eficientes que criar um vetor de apontadores para as células, ordenar esse vetor conforme o conteúdo das células, e usar esse vetor para reordenar as células. Cada algoritmo traz desafios diferentes:

   - MERGESORT: intercalar duas listas ordenadas é fácil. Mas como dividir uma lista em duas do mesmo tamnho de forma eficiente? Para vetores, essa divisão era trivial; é preciso tomar cuidado para não aumentar a complexidade do algoritmo. O gasto de memória deve ser constante, a menos da pilha de recursão.
   - QUICKSORT: Aqui o desafio é fazer o particionamento de forma estável e concatenar sublistas rapidamente. Tente fazer também a forma aleatorizada.

   Em cada caso, justifique a complexidade.

   Em princípio, uma célula da lista tem um campo `Info` do `TipoOrd` e um campo `Next` usado para apontar outra célula. Mas evite usar referências de baixo nível, onde der. Assuma a existência de métodos "óbvios", como pendurar uma célula em outra, percorrer a lista até que algo aconteça, etc.