USP-ICMC-BInfo

Introdução à Ciência de Computação II - SCC-0501

Prof. João Luís - 25/8/2011

Lista de Exercícios 3

1. Resolva as seguintes equações de recorrência:

   (a)

   $$\begin{cases} T(n) = T(n-1) + c & c \text{ constante}, n > 1 \\ T(1) = 0 \end{cases}$$

   (b)

   $$\begin{cases} T(n) = cT(n-1) & c, k \text{ constantes}, n > 0 \\ T(0) = k \end{cases}$$

   (c)

   $$\begin{cases} T(n) = 3T(n/2) + n & n > 1 \\ T(1) = 1 \end{cases}$$

   cuja solução satisfaz $T(n) = \mathcal{O}(n\ log^2\ n)$. Prove usando indução matemática em n.

2. Considere o algoritmo a seguir. Suponha que a operação crucial é o fato de inspecionar um elemento. O algoritmo inspeciona os n elementos de um conjunto e, de alguma forma, isso permite descartar $\frac{2}{5}$ dos elementos e então fazer uma chamada recursiva sobre os $\frac{3n}{5}$ elementos restantes.

   ```
   void Pesquisa (int n)
   {
    if (n <= 1)
      'inspecione elemento' e termine
    else {
      para cada um dos n elementos 'inspecione o elemento';
      Pesquisa (3n/5);
      }
   }
   ```

   (a) Escreva uma **equação de recorrência** que descreva este comportamento.

   (b) Converta esta equação para um somatório.

   (c) Dê a fórmula fechada para este somatório.

3. Escreva um algoritmo recursivo para o problema das Torres de Hanói. Mostre a equação de recorrência e calcule a complexidade do algoritmo. Suponha três pinos e, num deles, alguns discos dispostos um sobre o outro em ordem de tamanho do diâmetro (o menor no topo). O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma situação.

Veja a próxima página...

4. Seja o algoritmo recursivo de Busca Binária mostrado abaixo. Encontre a equação de recorrência do algoritmo. Em seguida, defina a sua complexidade para o pior caso.

```
int busca_b(int a[], int x, int baixo, int alto)
{
 int meio;
 if (baixo > alto)
   return(-1);
 meio = (baixo + alto)/2;
 return(x == a[meio]) ? meio : x < a[meio] ?
      busca_b(a, x, baixo, meio-1) :
      busca_b(a, x, meio+1, alto);
}
```

5. Um elemento majoritário em um arranjo A de tamanho n, é um elemento que aparece mais que $\frac{n}{2}$ vezes. Por exemplo, o arranjo 3, 3, 4, 3, 4, 4, 2, 4, 4 tem o 4 como elemento majoritário. Se não existir um elemento majoritário seu algoritmo deve indicar esse fato. Temos a seguir, um esboço de um algoritmo para solucionar o problema acima:

   > Primeiro, um candidato a elemento majoritário é encontrado (esta é a pior parte). Este candidato é o único elemento que poderá ser um possível majoritário. O segundo passo determina se o candidato selecionado é realmente majoritário. Isto é feito com uma pesquisa seqüencial no arranjo. Para encontrar um candidato a majoritário no arranjo A, forme um segundo arranjo B. Compare $A[1]$ e $A[2]$. Se forem iguais, coloque um deles em B. Caso contrário não faça nada. Compare $A[3]$ e $A[4]$. Novamente, se forem iguais coloque um deles em B. Caso contrário não faça nada. Continue dessa forma até ter processado todo o arranjo A. Então, recursivamente, encontre um candidato em B.

   (a) Como a recursividade termina?
   (b) Qual é o tempo de execução desse algoritmo?
   (c) Como podemos evitar o uso do arranjo extra B?
   (d) Escreva o programa que encontre o elemento majoritário.

References

[1] Profa. Maria Cristina Ferreira de Oliveira, *Introdução à Ciência da Computação II*. USP-ICMC, SCE0181, Segunda Lista de Exercícios.

[2] Prof. Thiago A. S. Pardo, *Introdução à Ciência da Computação II*. USP-ICMC, SCE0181, Lista de Exercícios 1.