

Lista de Exercícios 3: Relação de Recorrência

Professor: Moacir Pereira Ponti Jr.

PAE(s): Pâmela/Paulo Henrique

1. Resolva as seguintes equações de recorrência:

(i)

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

(ii)

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

(iii)

$$\begin{cases} T(n) = 3T(n/2) + n, & n > 1 \\ T(1) = 1 \end{cases} \quad (\text{resolva para } n = 2^k)$$

(iv)

$$\begin{cases} T(n) = 3T(n/3) + 1, & n > 1 \\ T(1) = 1 \end{cases} \quad (\text{resolva para } n = 3^k)$$

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);
  }
}
```

- (i) Escreva uma **equação de recorrência** que descreva este comportamento.
- (ii) Converta esta equação para um somatório.
- (iii) Dê a fórmula fechada para este somatório.

3. Considere o seguinte algoritmo recursivo que devolve a soma dos primeiros n cubos:
 $S(n) = 1^3 + 2^3 + \dots + n^3$.

```
void Cubo (int n)
{
    if (n = 1)
        return 1;
    else {
        return Cubo (n-1) + n*n*n;    }
}
```

- (i) Escreva uma **equação de recorrência** que descreva este comportamento.
 - (ii) Compare esse algoritmo com sua versão não-recursiva.
4. Implemente uma função em C que receba por parâmetro um arranjo de números float de tamanho n e, utilizando uma estratégia recursiva, some os elementos do vetor.
- (i) Faça a análise da complexidade dessa função: obtenha uma relação de recorrência pela contagem das operações, expanda a equação e a partir da expansão, conjecture acerca da ordem de crescimento.
 - (ii) (opcional) Prove por indução que a ordem de crescimento encontrada é verdadeira.
5. Implemente as versões recursiva e iterativa da função para obter números de Fibonacci em C.
- (i) Utilize a biblioteca `time.h` para medir e observar o tempo necessário para calcular $n = 15, 30, 45$ e 60 , utilizando as duas versões.
 - (ii) Faça a análise de complexidade da função recursiva: obtenha uma relação de recorrência pela contagem das operações, expanda a equação e a partir da expansão, conjecture acerca da ordem de crescimento.
 - (iii) Faça a análise de complexidade da função iterativa: obtenha uma fórmula pela contagem de operações e, encontre constantes de forma a mostrar o limite assintótico superior pela notação O (big oh).
 - (iv) Mostre que a versão iterativa é $\Theta(n)$ utilizando a notação formal.
 - (v) Mostre que a versão iterativa é $\Theta(n)$ utilizando o teorema $f(n) = \Theta(g(n))$ se e somente se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.
 - (vi) (opcional) Prove por indução que a ordem de crescimento encontrada para a versão recursiva é verdadeira.
6. 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 .

- (i) Como a recursividade termina?
- (ii) Qual é o tempo de execução desse algoritmo?
- (iii) Como podemos evitar o uso do arranjo extra B ?
- (iv) Escreva o programa que encontre o elemento majoritário.

Referências

- [1] Parte deste material foi adaptado das listas de exercícios do Prof. João Luís Garcia Rosa, ICMC/USP.