# Capítulo 8

# Programação Dinâmica

*" Uma base sólida de conhecimento e técnica de algoritmos é uma das características que separa o programador experiente do aprendiz. Com a moderna tecnologia de computação, é possível realizar algumas tarefas sem saber muito sobre algoritmos, mas com um boa base pode-se fazer muito, muito mais."*

**- Cormen, Leiserson, Rivest, Stein -**

## Sumário:



## 8.1- Conceitos

Na programação científica, existe um conjunto muito significativo de problemas que são resolvidos por relações de recorrência. As relações de recorrência podem ser implementadas por de algoritmos iterativos, que utilizam a estratégia de programação dinâmica, ou por algoritmos recursivos, que utilizam as estratégias de decrementar para conquistar ou dividir e conquistar.

Os algoritmos iterativos utilizam explicitamente um ciclo repetitivo, enquanto os algoritmos recursivos utilizam as chamadas sucessivas de si mesmo.

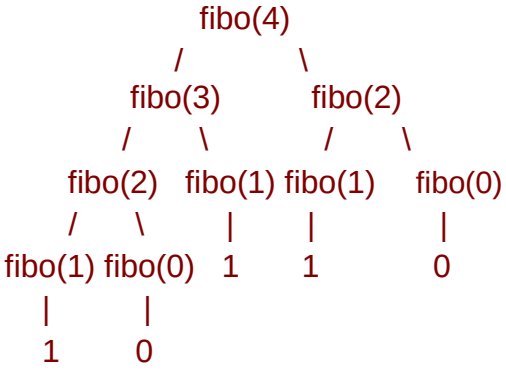
A estratégia de programação dinâmica, foi proposta pelo matemático Richard Bellmam em 1957 e pode ser comparada a outras técnicas de desenvolvimento de algoritmos.

Quando estudamos a estratégia de divisão e conquista, vimos que esse método decompõe um problema em subproblemas de menor dimensão; encontra de forma recursiva as soluções parciais desses subproblemas, e combina essas soluções para encontrar a solução do problema original.

Na programação dinâmica utilizamos a mesma estratégia, resolvemos todos os subproblemas, mas armazenamos as suas soluções para utiliza-las na resolução do problema original. Por esse facto, esta técnica assemelha-se a estratégia de

divisão e conquista. Contudo é mais eficiente porque não processa o mesmo subproblema repetidas vezes.

Por exemplo, para calcular o Fibonacci de 4, pela estratégia de divisão e conquista, repetimos duas vezes o cálculo do Fibonacci de 2; três vezes o cálculo do Fibonacci de 1, e duas vezes o cálculo do Fibonacci de 0. Essa constactação, pode ser comprovada pelo diagrama de árvore que apresentamos a seguir.




A programação dinâmica, deve ser utilizada, quando tomarmos consciência, que uma solução recursiva vai executar várias vezes os mesmos subprogramas. Para evitar esse desperdício de tempo de processamento, basta refazer o algoritmo para executar cada subproblema uma única vez e armazenar a solução parcial.

Em uma abordagem é feita de baixo para cima (*bottom-up),* e memoriza numa primeira fase, os casos de base da relação de recorrência. A partir desses casos base, vamos resolver os subproblemas maiores tendo em conta as soluções já encontradas.

A programação dinâmica é mais uma técnica para eliminar a recursão de algoritmos recursivas que possuem múltiplas chamadas.

## 8.2- Exemplos

### 8.2.1- Número de Fibonacci

A solução é muito simples, basta armazenar nas duas primeiras posições de um vector, os casos de base, ou seja, fibo(0) = 1 e fibo(1) = 1. Em seguida, utilizar essas posições para calcular os restantes elementos. Esta estratégia é descrita pela seguinte função:

```
int fiboDinamico (int n)
{
    int i, fib[50] = {0,1};
    if (n <= 1)
```

```
        return n;
    for ( i = 2; i <= n ; i++)
        fib[i] = fib[i -1] + fib[i - 2];
    return fib[n];
}
```

Vamos escrever em seguida, um algoritmo mais eficiente. Sabemos que n-ésimo termo de Fibonacci, para n > 1, depende apenas do n-1 e n-2 ésimos termos de Fibonacci. Então podemos aproveitar essa propriedade para gerar essa sequência utilizando três variáveis auxiliares: uma, denominada por anterior, que armazena o valor do Fibonacci de n - 2, outra que denominada por actual, que armazena o Fibonacci de n - 1 e uma terceira que denominada por próximo, que guardará o Fibonacci de n. Apresentamos em seguida, uma versão que implementa essa estratégia:

```
int fibIterativo (int n)
{
    int i, anterior = 0, actual = 1, proximo;
    if (n <= 1)
        return n;
    for (i = 2; i <= n ; i++)
     {
        proximo = actual + anterior;
        anterior = actual;
        actual = proximo;
     }
    return proximo;
}
```

### 8.2.2- Coeficientes Binomiais

O coeficiente binomial C(n,k) que representa o números de combinações de n k a k elementos, é descrito pela fórmula:

$$C(n, k) = \frac{n!}{k!(m-k)!}$$

Mesmo com valores que possam ser representados por variáveis do tipo Mesmo com valores que possam ser representados por variáveis do tipo inteiro, em muitos casos, os valores parcelares podem ficar corrompidos, devido ao estoiro da capacidade de representação dessas variáveis. Uma forma de resolver esse problema passa pela eliminação do cálculo do factorial, através de uma fórmula recorrente descoberta pelo Matemático Francês, Baise Pascal (1623 a 1662) que é conhecido como triângulo de Pascal.

c (n , k) = 0 se k > n
c (n , k) = 1 se k = 0
c (n , k) = 1 se n = 0
c (n , k) = c(n - 1, k - 1) + c(n - 1, k) se n > 0 e k > 0

A condição k > n serve para evitar que a função entre num processo recursivo anormal, por esse facto, a sua solução é trivial. Se k = 0 ou n = 0 a solução também é trivial. Para k > 0 e n > 0 a resolução passa pela separação do problema em dois subproblemas de menor complexidade, o cálculo de C (n-1, k-1) e cálculo de C (n-1, k). A solução do problema original consiste na soma das soluções dos problemas parciais. Então, este problema recursivo pode ser modelado por:

Caso Base : 0 se k > n ou 1 se n = 0 ou k = 0
Passo recursivo: c (n - 1, k-1) + c (n - 1, k)

que implementada pela seguinte função:

```
int combRec (int n, int k)
{
  if ( k > n )
     return 0;
   if ( k = 0 || n = 0 )
     return 1;
   return combRec (n-1, k-1) + combRec (n-1 , k);
}
```

Esta implementação não é aconselhável porque a função repete o cálculo de algumas combinações se os números inteiros n e k forem muito grandes. Mas a grande deficiência do algoritmo, resida na dupla chamada recursiva, que provoca o crescimento muito rápido de chamadas, diminuindo desse modo a velocidade de processamento a medida que o programa vai sendo executado.

A solução dinâmica que apresentamos em seguida exige a utilização de uma matriz para armazenar os coeficientes binomiais.

```
int CombDinamico ( int n , int k)
{
   int i, j, min, bicoef[30][30];
   if ( k > n )
      return 0;
   if ( k == 0 || n == 0 )
      return 1;
   for ( i = 0 ; i <= n ; i ++ )
     {
        min = minimo (i , k);              /*  o menor valor entre i e k  */
        for ( j = 0 ; j <= min ; j++ )
           if ( j == 0 || j == i )
              bicoef[i][j] = 1;
```

```
        else
            bicoef[i][j] = bicoef[i-1][j-1] +bicoef[i-1][j];
      }
    return bicoef[n][k];
  }
```

Esta solução pode ser melhorada em termos de utilização de memória, pois vejamos: para calcular o coeficiente binomial de ordem n, só temos a necessidade de conhecer os coeficientes binomiais de ordem n-1. Então, só necessitamos de ter duas linhas do triângulo de Pascal. A linha actual que está a ser calculada e a linha anterior completamente calculada. Como exercício desenvolva essa versão melhorada.

### 8.2.3- Problema de Mochila

## 8.3 - Recomendações Bibliográficas

Para o leitor aprofundar os seus conhecimentos sobre a estratégia de divisão e conquista e programação dinâmica, recomendamos os seguintes livros:

Cormen T. H.;- *Introduction to algorithms,* Cambridge, Massachusetts, MIT press, 2009.

Levitin A.;- *Introduction to the Design and Analysis of Algorithms,* 3rd Edition, Pearson, 2011.

Robert E. S.;- Programming Abstraction in C: a Second Course in Computer Science, Addison.Wesley, 1998.

Routo T. ; - *Desenvolvimento de Algoritmos e Estruturas de Dados*, McGraw_hill, Brasil, 1991.

Sedgewick R. ;- *Algoritms in C*, Massachusetts, Addison-Wesley, Reading, Massachusetts,1990.

## 8.4 - Exercícios

**8.4.1**-Desenvolva uma função que recebe como parâmetro um número inteiro não negativo e retorna o n-ésimo termo de Lucas:

| | | |
|---|---|---|
| Lucas(n) = 2 | se | n = 0 |
| Lucas(n) = 1 | se | n = 1 |

Lucas (n) = Lucas(n-1) + Lucas(n-2) se n > 2

**8.4.2-**Desenvolva uma função que recebe como parâmetro um número inteiro não negativo e retorna a potência desse número, descrita pela seguinte fórmula recorrente:

$a^n = 1$ se n = 0
$a^n = ( a^{n/2})^2$ se n é par
$a^n = a \times ( a^{n/2})^2$ se n é impar

**8.4.3**- Dada a fórmula recorrente.

G(n) = 2 se n = 0
G(n) = 1 se n = 1
G(n) = 3 se n = 2
G(n)= G(n-1) + 5G(n-2) + 3G(n-3) se n > 3

Desenvolva uma função que recebe como parâmetro um número inteiro não negativo n, e retorne o valor de G(n), utilizando o paradigma da programação dinâmica.

**8.4.4**-Desenvolva uma função para calcular o n-éssimo termo de Tribonacci. A sequência de Tribonacci pode ser definida pela seguinte relação de recorrência.

Tribonacci(n) = 0 se n = 0
Tribonacci(n) = 1 se n = 1ou n = 2
Tribonacci(n) = tribonacci(n-1) + tribonacci(n-2) + tribonacci(n-3) se n > 3