Análise Técnica de Algoritmos

Lab 04 - Dynamic Programming

## Professor: Gustavo Soares

## Elaborador: Uian Sol Gorgônio

# Problema 1: Números de Fibonacci

Tal como divisão e conquista, a programação dinâmica divide um problema em subproblemas. Alguns casos porém, os subproblemas se repetem. É viável armazenar esses valores antes calculados e utilizá-los quando um subproblema se repetir.

Considere a implementação recursiva para calcular um número de Fibonacci.

int fib(int n)  
{  
 if (n <= 1)  
 return n;  
 return fib(n-1) + fib(n-2);  
}

Números de Fibonacci em C.

fib(5)  
 / \  
 fib(4) fib(3)  
 / \ / \  
 fib(3) fib(2) fib(2) fib(1)

/ \ / \ / \  
 fib(2) fib(1) fib(1) fib(0) fib(1) fib(0)  
 / \  
 fib(1) fib(0)

Árvore de recursão para fib(5).

**Atividade:** Implemente em Python o código para o número de Fibonacci recursivo utilizando programação dinâmica para armazenar e consultar os valores já calculados.

Problema 2: O problema da mochila

Um dos mais famosos e interessantes problemas da computação. Dado um conjunto de itens, cada um com seu peso e valor, quais itens selecionar para colocar em um mochila de tal modo que o valor desses itens seja o máximo possível sem exceder a capacidade que a mochila pode armazenar em peso?

Um algoritmo ingênuo seria gerar todos os subconjuntos dos itens, e comparar o valor de cada um desses. Para N itens teríamos 2^N subconjuntos. Quando N aumenta, torna-se impraticável realizar tal computação.

A solução recursiva segue a lógica de que para cada item temos duas opções ao buscar pelo melhor subconjunto, (1) incluir o item ou (2) não incluí-lo.

(1) Se o item foi incluso, o valor do subconjunto é o valor do item + o melhor valor do subconjunto gerado a partir dos outros itens. E a capacidade da mochila fica reduzida do peso do item escolhido.

(2) Se o itens não for incluso, o valor do subconjunto é o valor do subconjunto gerado a partir dos outros itens. E a capacidade da mochila se mantem, pois nada foi colocado.

Com o caso especial: Se o peso do objeto for maior que o peso da mochila, então apenas resta a opção de não escolhe-lo.

Analise a código para essa recursão a fim de entender o processo. Considere duas listas correspondentes, uma com o valor dos objetos e outra com os seus pesos. E que, para melhorar a legibilidade, são consideradas com visibilidade global.

def mochila(objeto, capacidade):  
 if objeto == 0:  
 if pesos[0] <= capacidade:  
 return valores[0]  
 else:  
 return 0  
  
 nao\_colocar = mochila(objeto - 1, capacidade)  
 colocar = 0  
 if pesos[objeto] <= capacidade:  
 colocar = valores[objeto] + mochila(objeto - 1, capacidade - pesos[objeto])  
   
 return max(nao\_colocar, colocar)  
  
  
# Tests  
valores = [60, 100, 120]  
pesos = [10, 20, 30]  
capacidade = 30  
assert mochila(2, capacidade) == 160  
  
valores = [20, 10, 30, 40, 50]  
pesos = [1, 5, 4, 7, 3]  
capacidade = 10  
assert mochila(4, capacidade) == 100

Solução em força bruta para o problema da mochila.

Percebemos pela árvore de recursão dessa implementação que alguns subproblemas se repetem. No caso descrito, m(1, 3), m(0, 3) e m(0, 2).

m(3, 4)  
 / \   
 / \   
 m(2,4) m(2,3)  
 / \ / \   
 / \ / \  
 m(1,4) m(1,3) m(1,3) m(1,2)

/ \ / \ / \ / \   
 / \ / \ / \ / \

m(0,4) m(0,3) m(0,3) m(0,2) m(0,3) m(0,2) m(0, 2) m(0, 1)

Árvore de recursão para valores = [10, 20, 30, 40], pesos = [1, 1, 1, 1] e capacidade = 4.

**Atividade:** Modifique o código da mochila para calcular o valor da mochila com programação dinâmica armazenando os valores já calculados.