Programação Dinâmica

Prof. Fábio Dias

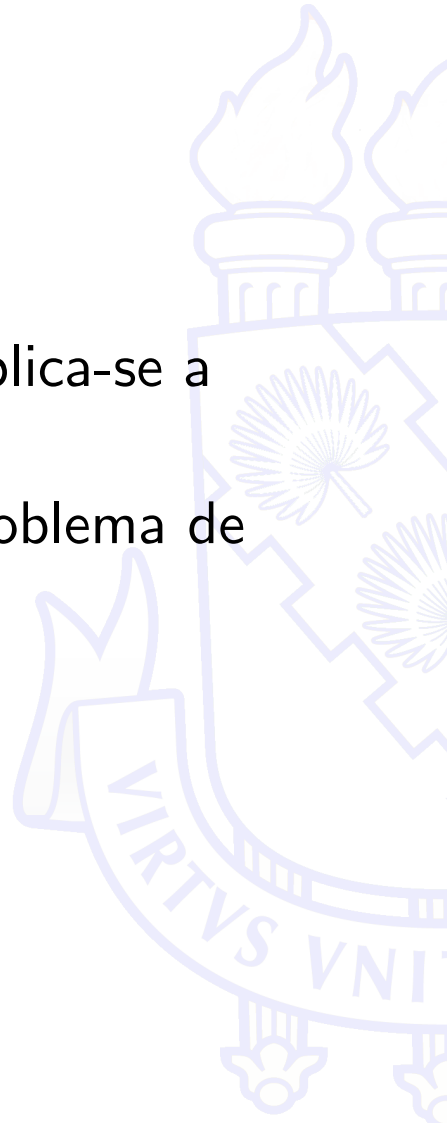
28 de janeiro de 2025

UFC CAMPUS QUIXADÁ

Introdução

- “programação” se refere a um método tabular, não ao processo de escrever código de computador;
- Assim como o método de divisão e conquista, resolve problemas combinando as soluções para subproblemas;
- A diferença que na divisão e conquista os subproblemas são independentes:
  - Resolvem os subproblemas recursivamente e depois combinam suas soluções para resolver o problema original.
- Na programação dinâmica os subproblemas se sobrepõem, isto é, quando os subproblemas compartilham subsubproblemas;
- Nesse contexto, um algoritmo de divisão e conquista seria exponencial, já na programação dinâmica, chegamos em polinomial.

Elementos de Programação Dinâmica



- Tipicamente o paradigma de programação dinâmica aplica-se a problemas de otimização;
- Para podemos aplicarmos programação dinâmica, o problema de otimização precisa ter os seguintes elementos:
  - Subestrutura ótima;
  - Sobreposição de subproblemas;
  - Algoritmo recursivo de baixo para cima.

Subestrutura ótima

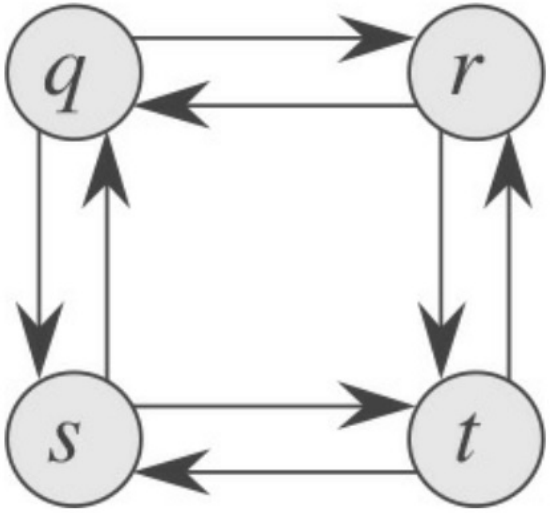
- Um problema apresenta uma subestrutura ótima se uma solução ótima para o problema contiver soluções ótimas para subproblemas;
- Problema do Caminho Mínimo:
  - Um caminho mínimo entre dois vértices contém outros caminhos mínimos;
  - Em outras palavras, dado um caminho mínimo entre dois vértices, todos os sub-caminhos contidos também são caminhos mínimos;
- Precisamos provar!!!!
- Normalmente, provamos usando o método recortar e colar;

Subestrutura ótima - Problema do Caminho Mínimo

- Seja p um caminho mínimo entre os vértices u e v no grafo ponderado G;
- Suponha que esse caminho mínimo passe pelo vértice x, o sub-caminho de p entre os vértices u a x é ótimo, ou seja, é o caminho mínimo?
- Suponha por absurdo que não seja, logo existe um outro caminho, digamos q de custo $w(q) < w(p(u, x))$.
- Sabemos que $w(p) = w(p(u, x)) + w(p(x, v))$.
- Mas observe que podemos criar outro caminho entre u e v $q + p(x, v)$, de custo $w(q) + w(p(x, v)) < w(p)$, absurdo, pois p é mínimo.

Subestrutura ótima - Caminho simples mais longo não ponderado

- Encontrar um caminho simples (sem ciclos, ou seja, sem repetir vértices) de u para v que consista no maior número de arestas;
- Seja p um caminho simples mais longo entre os vértices u e v que passe pelo vértice x;
- Será que $p(u, x)$ é o caminho simples mais longo entre os vértices u e v?



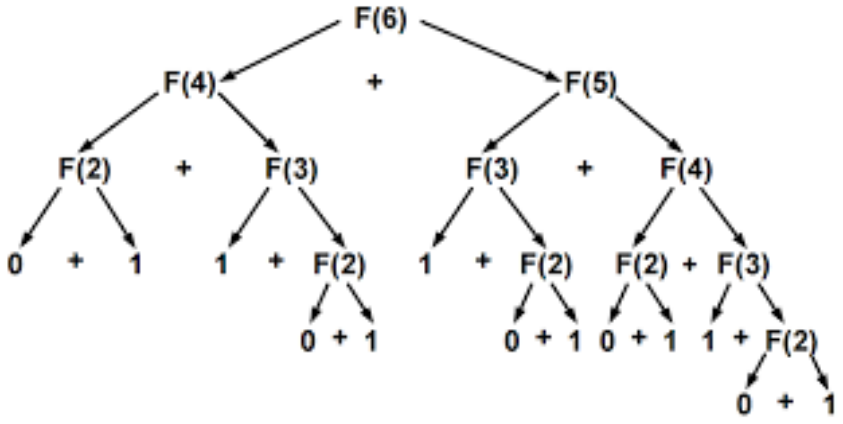
Sobreposição de subproblemas

- Quando subproblemas compartilham subsubproblemas;
- Nesse contexto, algoritmo recursivo normal para o problema resolve os mesmos subproblemas repetidas vezes;
- Quando analisarmos o número total de subproblemas distintos, chegamos a um polinômio no tamanho de entrada;
- Algoritmos de programação dinâmica, normalmente tiram proveito de subproblemas sobrepostos resolvendo cada subproblema uma vez e depois armazenando a solução em uma tabela onde ela pode ser examinada quando necessário, usando um tempo constante por busca.

Sobreposição de subproblemas - Fibonacci

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1. \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

(a) Recursão de Fibonacci



(b) Árvore das chamadas recursivas

```python
def fibonacci(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

Para $n = 10$, chegamos $[34, 55, 34, 21, 13, 8, 5, 3, 2, 1, 1]$

Algoritmo de baixo para cima - Fibonacci

Um algoritmo de programação dinâmica resolve cada subsubproblema só uma vez e depois grava sua resposta em uma tabela, evitando assim, o trabalho de recalcular a resposta toda vez que resolver cada subsubproblema;

```python
def fibonacci_memorizado(n):
    fib = [-1]*(n + 1)
    fib[0] = 0
    fib[1] = 1
    return fibonacci_memorizado_aux(n, fib)

def fibonacci_memorizado_aux(n, fib):
    if fib[n] != -1:
        return fib[n]
    else:
        if fib[n-1] == -1:
            fib[n-1] = fibonacci_memorizado_aux(n-1, fib)
        if fib[n-2] == -1:
            fib[n-2] = fibonacci_memorizado_aux(n-21, fib)
        fib[n] = fib[n-1] + fib[n-2]
        return fib[n]
```

(c) Algotimo Memorizado

```python
def fibonacci_baixo_cima(n):
    fib = [-1]*(n + 1)
    fib[0] = 0
    fib[1] = 1
    i = 2
    for i in range(2, n+1):
        fib[i] = fib[i-1] + fib[i-2]
    return fib[n]
```

(d) Algoritmo de baixo para cima

Sobreposição de subproblemas

O desenvolvimento de um algoritmo de programação dinâmica segue uma sequência de quatro etapas:

- Caracterizar a estrutura de uma solução ótima;
- Definir recursivamente o valor de uma solução ótima;
- Calcular o valor de uma solução ótima, normalmente de baixo para cima;
- Construir uma solução ótima com as informações calculadas.

Perguntas?!