MAC0338 - ANÁLISE DE ALGORITMOS

- **AULA 10**
  - [x] programação dinâmica: introdução
  - [x] corte de hastes
- **AULA 11**
  - [ ] programação dinâmica
  - [ ] produto de cadeias de matrizes
  - [ ] subsequência comum mais longa: introdução
- **AULA 12**
  - [ ] programação dinâmica
  - [ ] subsequência comum mais longa
- **AULA 13**
  - [ ] programação dinâmica
  - [ ] ABB ótima

programação dinâmica

O QUE É?

Esse método, ou estratégia de projeto de algoritmos, é uma espécie de tradução interativa inteligente da recursão e pode ser definido, vagamente, como "recursão com o apoio de uma tabela".

Como em um algoritmo recursivo, cada instância do problema é resolvida a partir da solução de instâncias menores, ou melhor, de subinstâncias da instância original.

A característica distintiva da programação dinâmica é a tabela que armazena as soluções das várias subinstâncias. O consumo de tempo do algoritmo é, em geral, proporcional ao tamanho da tabela.

Para que o método da programação dinâmica possa ser aplicado, é preciso que o problema tenha estrutura recursiva: a solução de toda instância do problema deve "conter" soluções de subinstâncias da instâncias.

Essa estrutura recursiva é representada por uma recorrência, e a recorrência pode ser traduzida em um algoritmo recursivo. Em alguns casos, o algoritmo recursivo refaz a solução de cada subinstância muitas vezes, e isso torna o algoritmo ineficiente. Nesses casos, é possível armazenar as soluções da subinstância numa tabela e assim evitar que elas sejam recalculadas.

EXEMPLO: NÚMEROS DE FIBONACCI

Considere a sequência 0, 1, 1, 2, 3, 5, 8 ... de números de Fibonacci.

Esses números são definidos pela recorrência:

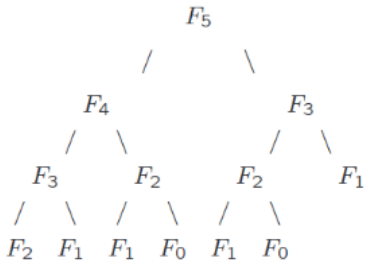
$$F(n) = F(n-1) + F(n-2)$$

a partir dos valores iniciais $F(0) = 0$ e $F(1) = 1$

Nosso problema é calcular $F(n)$ dado n. O seguinte algoritmo resolve o problema "de cima para baixo", em estilo recursivo:

```
FIB (n)
1  se n ≤ 1
2       devolva n e pare
3  devolva FIB (n−1) + FIB (n−2)
```

Esse algoritmo é muito ineficiente pois recalcula $F(i)$ várias vezes para cada i, como a figura abaixo sugere.



Para evitar que cada $F(i)$ seja recalculado várias vezes, podemos empregar o método de programação dinâmica. O algoritmo abaixo usa uma tabela $f[0..n]$ para armazenar os números de Fibonacci à medida que são calculados, "de baixo para cima".

```
FIB-PD (n)
1  f[0] := 0
2  f[1] := 1
3  para i := 2 até n
4       f[i] := f[i−1] + f[i−2]
5  devolva f[n]
```

Esse algoritmo consome apenas $\Theta(n)$ unidades de tempo.

CORTES DE HASTES

Sejam $p_1 \dots p_n$ inteiros positivos que correspondem, respectivamente, ao preço de venda de hastes de tamanho $1, \dots, n$. Dado um inteiro positivo n, o problema consiste em maximizar o lucro l_n obtido com a venda de uma haste de tamanho n, que pode ser vendida em pedaços de tamanho inteiro.

Para exemplificar o problema, considere uma haste de tamanho 6 com preços dos pedaços na tabela abaixo.

n	p_1	p_2	p_3	p_4	p_5	p_6
6	3	8	14	15	10	20

Note que se a haste for vendida sem nenhum corte, então temos lucro $l_6 = 20$. Caso cortemos um pedaço de tamanho 5, então a única possibilidade é vender uma parte de tamanho 5 e outra de tamanho 1, que fornece um lucro de $l_6 = p_5 + p_1 = 13$, o que é pior que vender a haste inteira. Se venderemos dois pedaços de tamanho 3, obtemos um lucro total de $l_6 = 2p_3 = 28$, que é o maior lucro possível.

ALGORITMO

Primeiro vamos construir um algoritmo de divisão e conquista natural para o problema do corte de hastes. Podemos definir l_n recursivamente definindo onde aplicar o primeiro corte na haste. Assim, se o melhor lugar para realizar o primeiro corte na haste é no ponto i (onde $1 \leq i \leq n$), então o lucro total é dado por $l_n = p_i + l_{n-i}$ que é o preço do pedaço de tamanho i somado ao maior lucro possível obtido com a venda do restante da haste, que tem tamanho $n-i$. Portanto, temos:

$$l_n = \max_{1 \leq i \leq n} \{ p_i + l_{n-1} \}$$

A igualdade sugere o seguinte algoritmo para resolver o problema, onde p é um vetor contendo os preços dos pedaços de uma haste de tamanho n.

```
CORTE HASTES-DV(n,p)
1 se n = 0 então
2     retorna 0
3 lucro = 0
4 para i = 1 até n faça
5     valor = p_i + CORTE HASTES-DV(n − i,p)
6     se lucro < valor então
7         lucro = valor
8 retorna lucro
```

Recorrência:

$$l_n = \begin{cases} 0, \text{ se } n = 0 & \text{(sem certeza)} \\ \max\limits_{1 \leq i \leq n} \{ p_i + l_{n-1} \} \end{cases}$$

Apesar de ser um algoritmo intuitivo e calcular corretamente o lucro máximo possível, ele é extremamente ineficiente, pois muito trabalho é repetido pelo algoritmo.

De fato, seja $T(n)$ o tempo de execução de CORTE HASTES-DV(n,p). Vamos utilizar o método da substituição para provar que $T(n) \geq 2^n$.

- claramente temos $T(0) = 1 = 2^0$
- suponha que $T(m) \geq 2^m$ para todo $0 \leq m \leq n-1$.
- portanto, notando que $T(n) = 1 + T(0) + T(1) + \dots + T(n-1)$, obtemos

$$\begin{aligned} T(n) &= 1 + T(0) + T(1) + \dots + T(n-1) \\ &\geq 1 + (2^0 + 2^1 + \dots + 2^{n-1}) \\ &= 2^n \end{aligned}$$

Assim, o problema possui a propriedade de sobreposição de subproblemas e também possui a propriedade de subestrutura ótima. Portanto, o problema tem os ingredientes necessários para que um algoritmo de programação dinâmica o resolva de forma eficiente.

Abaixo apresentamos um algoritmo com abordagem top-down para o problema do corte de hastes. Esse algoritmo mantém a estrutura de CORTE HASTES-DV(n,p), salvando os valores de soluções ótimas de subproblemas em um vetor $r[0 \dots n]$, de modo que $r[i]$ contém o valor de uma solução ótima para o problema de corte de uma haste de tamanho i. Ademais, vamos manter um vetor $s[0 \dots n]$ tal que $s[j]$ contém o primeiro lugar que deve-se efetuar o corte em uma haste de tamanho j.

Algoritmo 36: CORTE HASTES-TD(n,p)

```
1 Cria vetores r[0..n] e s[0..n]
2 r[0] = 0
3 para i = 1 até n faça
4 |  r[i] = −1
5 retorna CORTE HASTES-AUX(n, p, r, s)
```

Algoritmo 37: CORTE HASTES-AUX(n,p,r,s)

```
1 se r[n] ≥ 0 então
2 |  retorna r[n]
3 lucro = −1
4 para i = 1 até n faça
5 |  (valor, s) = CORTE HASTES-AUX(n − i, p, r, s)
6 |  se lucro < p_i + valor então
7 |  |  lucro = p_i + valor
8 |  |  s[n] = i
9 r[n] = lucro
10 retorna (lucro, s)
```

O algoritmo CORTE HASTES-TD (n) inicialmente cria os vetores r e s, faz r[0] = 0 e inicializa todas as outras entradas de r com -1, representando que ainda não calculamos esses valores. Feito isso, CORTE HASTES-AUX (n, p, r, s) é executado.

Inicialmente, nas linhas 1 e 2, o algoritmo CORTE HASTES-AUX (n, p, r, s) verifica se o subproblema em questão já foi resolvido. Caso o subproblema não tenha sido resolvido, então o algoritmo vai fazer isso de modo muito semelhante ao algoritmo CORTEHASTES-DV. A diferença é que agora salvamos o melhor local para fazer o primeiro corte em uma haste de tamanho n em s[n].

https://docplayer.com.br/88752081-Analise-de-algoritmos-e-estruturas-de-dados.html