Programação dinâmica

Programação dinâmica

Método usado na construção de soluções iterativas para problemas cuja solução recursiva tem uma complexidade elevada (exponencial, em geral)

Aplica-se, normalmente, a problemas de optimização

- Um problema de optimização é um problema em que se procura minimizar ou maximizar algum valor associado às suas soluções

Corte de varas

Uma empresa compra varas de aço, corta-as e vende-as aos pedaços

O preço de venda de cada pedaço depende do seu comprimento

Problema

Como cortar uma vara de comprimento n de forma a maximizar o valor de venda?

Comprimento i	1	2	3	4	5	6	7	8	9	10
Preço p_i	1	5	7	11	11	17	20	20	24	27

Corte de varas

Caracterização de uma solução óptima (1)

Soluções possíveis, para uma vara de comprimento 10

- Um corte de comprimento 1, mais as soluções para uma vara de comprimento 9
- Um corte de comprimento 2, mais as soluções para uma vara de comprimento 8
- Um corte de comprimento 3, mais as soluções para uma vara de comprimento 7

  ...
- Um corte de comprimento 9, mais as soluções para uma vara de comprimento 1
- Um corte de comprimento 10, mais as soluções para uma uma vara de comprimento 0

Qual a melhor?

Corte de varas

Caracterização de uma solução óptima (2)

Sejam os tamanhos dos cortes possíveis

$$1, 2, \ldots, n$$

com preços

$$p_1, p_2, \ldots, p_n$$

O maior valor de venda de uma vara de comprimento n é o máximo que se obtém

- fazendo um corte inicial de comprimento $1 \leq i \leq n$, de valor p_i, somado com
- o maior valor de venda de uma vara de comprimento $n - i$

Corte de varas

Função recursiva

Corte de uma vara de comprimento n

Tamanho dos cortes: $i = 1, \ldots, n$
Preços: $P = (p_1\ p_2\ \ldots\ p_n)$

$r_P[0..n]$: função t.q. $r_P[l]$ é o maior preço que se pode obter para uma vara de comprimento l, dados os preços P

$$r_P[l] = \begin{cases} 0 & \text{se} \quad l = 0 \\ \max\limits_{1 \le i \le l} \{p_i + r_P[l - i]\} & \text{se} \quad l > 0 \end{cases}$$

Preço máximo (chamada inicial da função): $r_P[n]$

Corte de varas

Implementação recursiva

CUT-ROD(p, l)

```
if l = 0 then
  return 0
q <- −∞
for i <- 1 to l do
  q <- max(q, p[i] + CUT-ROD(p, l - i))
return q
```

Argumentos

- p Preços das varas de comprimentos $\{1, 2, \ldots, n\}$
- l Comprimento da vara a cortar

Chamada inicial da função: CUT-ROD(p, n)

Corte de varas

Alguns números

Número de cortes possíveis

$$2^{n-1}$$

Exemplo ($n = 4$)

$4 \quad 1+3 \quad 2+2 \quad 3+1$

$1+1+2 \quad 1+2+1 \quad 2+1+1 \quad 1+1+1+1$

Número de cortes distintos possíveis

$$O\left(\frac{e^{\pi\sqrt{\frac{2n}{3}}}}{4n\sqrt{3}}\right)$$

Exemplo ($n = 4$)

$4 \quad 1+3 \quad 2+2 \quad 1+1+2 \quad 1+1+1+1$

Corte de varas

Implementação recursiva com *memoização*

MEMOIZED-CUT-ROD(p, n)

```
let r[0..n] be a new array
for l <- 0 to n do
     r[l] <- −∞
return MEMOIZED-CUT-ROD-2(p, n, r)
```

MEMOIZED-CUT-ROD-2(p, l, r)

```
if r[l] = −∞ then
  if l = 0 then
    q <- 0
  else
    q <- −∞
    for i <- 1 to l do
      q <- max(q, p[i] + MEMOIZED-CUT-ROD-2(p, l - i, r))
  r[l] <- q
return r[l]
```

NB: isto não é programação dinâmica

Corte de varas

Cálculo iterativo de $r[n]$ (1)

p_i	1	5	7	11	11	17	20	20	24	27

Preenchimento do vector r

	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	7	11	12	17	20	22	25	28

1. Caso base: $r[0] \leftarrow 0$
2. $r[1] \leftarrow \max\{\, p_1 + r[0] \,\} = \max\{\, 1 + 0 \,\}$
3. $r[2] \leftarrow \max\{\, p_1 + r[1], p_2 + r[0] \,\} = \max\{\, 1 + 1, 5 + 0 \,\}$
4. $r[3] \leftarrow \max\{\, p_1 + r[2], p_2 + r[1], p_3 + r[0] \,\} =$
   $= \max\{\, 1 + 5, 5 + 1, 7 + 0 \,\}$

...

11. $r[10] \leftarrow \max\{\, p_1 + r[9], p_2 + r[8], \ldots, p_4 + r[6], \ldots, p_{10} + r[0] \,\}$

Corte de varas

Cálculo iterativo de $r[n]$ (2)

BOTTOM-UP-CUT-ROD(p, n)

```
let r[0..n] be a new array
r[0] <- 0
for l <- 1 to n do
  q <- −∞
  for i <- 1 to l do
    q <- max(q, p[i] + r[l - i])
  r[l] <- q
return r[n]
```

Complexidade de BOTTOM-UP-CUT-ROD($p_1\ p_2\ \ldots\ p_n$)

Ciclo 3–7 é executado n vezes

Ciclo 5–6 é executado l vezes, $l = 1, \ldots, n$

$$1 + 2 + \ldots + n = \sum_{l=1}^{n} l = \frac{n(n+1)}{2} = \Theta(n^2)$$

Todas as operações têm custo constante

Complexidade temporal $\Theta(n^2)$

Complexidade espacial $\Theta(n)$

Corte de varas

Construção da solução (1)

O valor máximo por que é possível vender uma vara é calculado pela função BOTTOM-UP-CUT-ROD

Quais os cortes a fazer para obter esse valor?

Para o preenchimento da posição `l` do vector `r[]`, é escolhido o valor máximo de `p[i] + r[l - i]`

O facto de o valor máximo incluir a parcela `p[i]` significa a inclusão de um pedaço de vara de comprimento `i`

Logo, o valor máximo por que é possível vender uma vara de comprimento `l` (vector `s[]`) será obtido

- com um pedaço de comprimento `i` e
- o valor máximo por que é possível vender uma vara de comprimento `l - i`

Corte de varas

Construção da solução (2)

p_i		1	5	7	11	11	17	20	20	24	27
	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	7	11	12	17	20	22	25	28
$s[i]$		1	2	3	4	1	6	7	2	2	4

1. Caso base: $r[0] \leftarrow 0$
2. $r[1] \leftarrow \max\{\, p_1 + r[0] \,\} = \max\{\, 1 + 0 \,\}$, $s[1] \leftarrow 1$
3. $r[2] \leftarrow \max\{\, p_1 + r[1], p_2 + r[0] \,\} = \max\{\, 1 + 1, 5 + 0 \,\}$, $s[2] \leftarrow 2$
4. $r[3] \leftarrow \max\{\, p_1 + r[2], p_2 + r[1], p_3 + r[0] \,\} =$
   $= \max\{\, 1 + 5, 5 + 1, 7 + 0 \,\}$, $s[3] \leftarrow 3$

   ...

11. $r[10] \leftarrow \max\{\, p_1 + r[9], p_2 + r[8], \ldots, p_4 + r[6], \ldots, p_{10} + r[0] \,\}$,
    $s[10] \leftarrow 4$

Corte de varas

Construção da solução (3)

$s[1..n]$: $s[l]$ é o primeiro corte a fazer numa vara de comprimento l

EXTENDED-BOTTOM-UP-CUT-ROD(p, n)

```
let r[0..n] and s[1..n] be new arrays
r[0] <- 0
for l <- 1 to n do
  q <- −∞
  for i <- 1 to l do
    if q < p[i] + r[l - i] then
      q <- p[i] + r[l - i]
      s[l] <- i           // corte feito na posição i
  r[l] <- q
return r and s
```

Corte de varas

Resolução completa

PRINT-CUT-ROD-SOLUTION(p, n)

```
(r, s) <- EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
print "The best price is ", r[n]
while n > 0 do
  print s[n]
  n <- n - s[n]
```

Programação dinâmica

Condições de aplicabilidade

A programação dinâmica aplica-se a problemas que apresentam as características seguintes:

Subestrutura óptima (*Optimal substructure*)

- Um problema tem subestrutura óptima se uma sua solução óptima é construída com recurso a soluções óptimas de subproblemas

Subproblemas repetidos (*Overlapping subproblems*)

- Existem subproblemas repetidos quando os subproblemas de um problema têm subproblemas em comum

Programação dinâmica

Aplicação

1. Caracterização de uma solução óptima
2. Formulação recursiva do cálculo do valor de uma solução óptima
3. Cálculo iterativo do valor de uma solução óptima, por tabelamento
4. Construção de uma solução óptima