Sequências e subsequências

Seja x a sequência

$$x_1 \, x_2 \, \ldots \, x_m, \quad m \geq 0$$

A sequência $z = z_1 \, z_2 \ldots z_k$ é uma subsequência de x se

$$z_j = x_{i_j}, \quad j = 1, \ldots, k \quad \text{e} \quad i_j < i_{j+1}$$

Exemplo

$$x = \text{A B C B D A B}$$

São (algumas) subsequências de x:

A   A B D   B B B   C B A   B C B A

A B C B D A B (*toda a sequência*) λ (*a sequência vazia*)

Não são subsequências de x:

A A A   D C   E

Subsequências comuns

Sejam x e y as sequências

$$x_1\, x_2\, \ldots\, x_m \quad \text{e} \quad y_1\, y_2\, \ldots\, y_n, \quad m, n \geq 0$$

A sequência z é uma subsequência comum a x e y se

- z é uma subsequência de x e
- z é uma subsequência de y

Exemplo

$$x = \text{A B C B D A B}$$
$$y = \text{B D C A B A}$$

Subsequências comuns a x e a y

A   A B   C B A λ ...

Maiores subsequências comuns a x e a y

B C A B   B C B A   B D A B

Maior subsequência comum

Longest common subsequence

Problema

Dadas duas sequências x e y

$$x_1\, x_2\, \ldots\, x_m \quad \text{e} \quad y_1\, y_2\, \ldots\, y_n, \quad m, n \geq 0$$

determinar uma maior subsequência comum a x e a y

Número de subsequências de uma sequência de comprimento m

$$2^m$$

Maior subsequência comum

Caracterização de uma solução óptima (para sequências não vazias)

$$x = x_1\, x_2\, \ldots\, x_m \quad \text{e} \quad y = y_1\, y_2\, \ldots\, y_n, \quad m, n > 0$$

Se o último símbolo é o mesmo ($x_m = y_n$)

Uma maior subsequência comum a x e y será uma maior subsequência comum a

$$x_1\, x_2\, \ldots\, x_{m-1} \quad \text{e} \quad y_1\, y_2\, \ldots\, y_{n-1}$$

acrescida de x_m

Se terminam com símbolos diferentes ($x_m \neq y_n$)

Uma maior subsequência comum a x e y será uma maior de entre as maiores subsequências comuns a

$$x_1\, x_2\, \ldots\, x_{m-1} \quad \text{e} \quad y_1\, y_2\, \ldots\, y_n$$

e as maiores subsequências comuns a

$$x_1\, x_2\, \ldots\, x_m \quad \text{e} \quad y_1\, y_2\, \ldots\, y_{n-1}$$

Maior subsequência comum

Função recursiva

Comprimento de uma maior subsequência comum às sequências

$$x = x_1\, x_2\, \ldots\, x_m \quad \text{e} \quad y = y_1\, y_2\, \ldots\, y_n, \quad m, n \geq 0$$

$c_{xy}(0..m, 0..n)$: $c_{xy}(i, j)$ é o comprimento das maiores subsequências comuns a $x_1 \ldots x_i$ e $y_1 \ldots y_j$

$$c_{xy}(i,j) = \begin{cases} 0 & \text{se} \quad i = 0 \lor j = 0 \\ 1 + c_{xy}(i-1, j-1) & \text{se} \quad i, j > 0 \land x_i = y_j \\ \max\{c_{xy}(i-1, j), c_{xy}(i, j-1)\} & \text{se} \quad i, j > 0 \land x_i \neq y_j \end{cases}$$

Chamada inicial da função

Comprimento de uma maior subsequência comum a x e y: $c_{xy}(m, n)$

Maior subsequência comum

Tabela para $c_{xy}(i, j)$

A função c_{xy} tem dois argumentos, logo, os valores da função serão guardados numa matriz

Valores possíveis para i

- O valor inicial é $m \geq 0$
- Nas chamadas recursivas, o valor do primeiro argumento mantém-se ou diminui em 1 unidade
- O caso base é atingido quando i é 0

Valores possíveis para j

- O valor inicial é $n \geq 0$
- Nas chamadas recursivas, o valor do segundo argumento mantém-se ou diminui em 1 unidade
- O caso base é atingido quando j é 0

A tabela terá índices $(0..m) \times (0..n)$

Maior subsequência comum

Tabulação de $c_{xy}(i, j)$

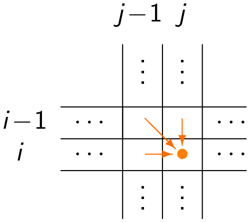
Caso base

Se $i = 0$ ou $j = 0$, então $c(i, j) = 0$

Casos recursivos

O valor de $c(i, j)$ depende:

- Do valor de $c(i-1, j-1)$ ou
- Dos valores de $c(i-1, j)$ e de $c(i, j-1)$



Para garantir que os valores necessários já estão calculados:

- As linhas são calculadas da linha 1 à linha m
- Dentro de cada linha, os valores são calculados da coluna 1 à coluna n

Argumentos da função iterativa

Os parâmetros do problema, que são as sequências x e y

Maior subsequência comum

Cálculo iterativo de $c[m, n]$

LONGEST-COMMON-SUBSEQUENCE-LENGTH(x, y)

```
m <- |x|
n <- |y|
let c[0..m, 0..n] be a new array
for i <- 1 to m do                    // caso base (j = 0)
  c[i, 0] <- 0
for j <- 0 to n do                    // caso base (i = 0)
  c[0, j] <- 0
for i <- 1 to m do                    // linhas 1 a m
  for j <- 1 to n do                  // colunas 1 a n
    if x[i] = y[j] then
      c[i, j] = 1 + c[i - 1, j - 1]
    else if c[i - 1, j] >= c[i, j - 1] then
      c[i, j] = c[i - 1, j]
    else
      c[i, j] = c[i, j - 1]
return c[m, n]
```

Maior subsequência comum

Análise da complexidade

LONGEST-COMMON-SUBSEQUENCE-LENGTH($x_1 \ldots x_m$, $y_1 \ldots y_n$)

Todas as operações executadas têm custo constante

Ciclo 4–5 é executado m vezes

Ciclo 6–7 é executado $n+1$ vezes

Ciclo 8–15 é executado m vezes

Ciclo 9–15 é executado n vezes em cada iteração do ciclo 8–15

$$\Theta(m) + \Theta(n+1) + \Theta(mn) = \Theta(mn)$$

Complexidade temporal $\Theta(mn)$

Complexidade espacial $\Theta(mn)$

Maior subsequência comum

Construção da sequência

Para identificar a maior subsequência comum, basta saber se a posição a partir da qual o valor de $c[i, j]$ foi calculado foi

$c[i-1, j-1]$ (NW)	ou	$c[i-1, j]$ (N)	ou	$c[i, j-1]$ (W)

Se foi a partir de $c[i-1, j-1]$, o símbolo $x_i = y_j$ é um elemento da subsequência

É possível reconstruir a maior subsequência comum calculada, seguindo as direcções NW, N e W, partindo da posição $[m, n]$ e até chegar à linha ou à coluna 0

Maior subsequência comum

Construção da solução

LONGEST-COMMON-SUBSEQUENCE(x, y)

```
m <- |x|
n <- |y|
let c[0..m, 0..n] and d[1..m, 1..n] be new arrays
for i <- 1 to m do
  c[i, 0] <- 0
for j <- 0 to n do
  c[0, j] <- 0
for i <- 1 to m do
  for j <- 1 to n do
    if x[i] = y[j] then
      c[i, j] = 1 + c[i - 1, j - 1]
      d[i, j] = NW
    else if c[i - 1, j] >= c[i, j - 1] then
      c[i, j] = c[i - 1, j]
      d[i, j] = N
    else
      c[i, j] = c[i, j - 1]
      d[i, j] = W
return c and d
```

Maior subsequência comum

Resultado da aplicação a x = ABCBDAB e y = BDCABA

			B	D	C	A	B	A	y
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	**0**	N 0	N 0	N 0	NW 1	W 1	NW 1	
B	2	0	**NW 1**	**W 1**	W 1	N 1	NW 2	W 2	
C	3	0	N 1	N 1	**NW 2**	**W 2**	N 2	N 2	
B	4	0	NW 1	N 1	N 2	N 2	**NW 3**	W 3	
D	5	0	N 1	NW 2	N 2	N 2	**N 3**	N 3	
A	6	0	N 1	N 2	N 2	NW 3	N 3	**NW 4**	
B	7	0	NW 1	N 2	N 2	N 3	NW 4	**N 4**	
x	i								

Maior subsequência comum a x e y calculada: B C B A

Maior subsequência comum

Reconstrução da subsequência

$d[1..m, 1..n]$: $d[i, j]$ é

- NW se $x_i = y_j$
- N se a subsequência calculada é comum a $x_1 \ldots x_{i-1}$ e $y_1 \ldots y_j$
- W se a subsequência calculada é comum a $x_1 \ldots x_i$ e $y_1 \ldots y_{j-1}$

PRINT-LCS(d, x, i, j)

```
if i = 0 or j = 0 then
  return
if d[i, j] = NW then
  PRINT-LCS(d, x, i - 1, j - 1)
  print x[i]
else if d[i, j] = N then
  PRINT-LCS(d, x, i - 1, j)
else // d[i, j] = W
  PRINT-LCS(d, x, i, j - 1)
```

Dilema do ladrão — Versão 1

Problema

Produtos	Quantidade	Valor
A	10 kg	600
B	20 kg	1000
C	30 kg	1200
D	40 kg	1400

Capacidade do saco (Máximo a transportar)
50 kg

O que levar, para maximizar o produto do roubo?

$$10\,\text{kg} \times \text{A} + 20\,\text{kg} \times \text{B} + 20\,\text{kg} \times \text{C}$$

$$\text{Peso total} = 10 + 20 + 20 = 50$$

$$\text{Valor total} = 10 \times \frac{600}{10} + 20 \times \frac{1000}{20} + 20 \times \frac{1200}{30} = 2400$$

Dilema do ladrão — Versão 1

Solução

Algoritmo

1. Calcular valor por quilo
2. Ordenar produtos por ordem decrescente do valor por quilo
3. Enquanto ainda há algum produto disponível e espaço no saco

   Colocar no saco a maior quantidade possível do próximo produto disponível

É um algoritmo *greedy*

Dilema do ladrão — Versão 2

Artigos indivisíveis — *Knapsack* 0-1

Artigos	Peso (kg)	Valor
A	10	600
B	20	1000
C	30	1200
D	40	1400

Capacidade do saco (Máximo a transportar)
50 kg

O que levar, para maximizar o produto do roubo?

Artigos	Peso total	Valor
A + B	30	1600
A + C	40	1800
A + D	50	2000
B + C	50	2200

Dilema do ladrão — Versão 2

Como escolher?

Escolher artigos mais valiosos

D + A

Escolher artigos com maior valor/kg

A + B

Escolher... como, para chegar a?

B + C

Não existe uma estratégia *greedy* que funcione

Estratégia (exaustiva) possível

Examinar todas as alternativas e escolher uma a que corresponda o maior valor

Resulta? Quantas são? $O(2^{\text{Número de artigos}})$

Dilema do ladrão — Versão 2

Caracterização de uma solução óptima (1)

O maior valor possível de obter com os artigos $\{A, B, C, D\}$ e capacidade 50 kg é o máximo entre

- A soma entre o valor de D e o maior valor possível de obter com os artigos $\{A, B, C\}$ e capacidade $50 - 40 = 10$ kg e
- O maior valor possível de obter com os artigos $\{A, B, C\}$ e capacidade 50 kg

O maior valor possível de obter com os artigos $\{A, B, C\}$ e capacidade j kg é o máximo entre

- A soma entre o valor de C, v_C, e o maior valor possível de obter com os artigos $\{A, B\}$ e capacidade $j - w_C$ kg (se w_C, o peso de C, não for superior a j) e
- O maior valor possível de obter com os artigos $\{A, B\}$ e capacidade j kg

Dilema do ladrão — Versão 2

Caracterização de uma solução óptima (2)

Sejam v_i e w_i, respectivamente, o valor e o peso do artigo i

Em geral, o maior valor possível de obter com os artigos

$$1, \ldots, i$$

e capacidade j será o máximo entre:

- A soma de v_i e o maior valor possível de obter com os artigos

  $$1, \ldots, i-1$$

  e capacidade $j - w_i$ (só se $w_i \leq j$) e
- O maior valor possível de obter com os artigos

  $$1, \ldots, i-1$$

  e capacidade j

Dilema do ladrão — Versão 2

Função recursiva

n artigos, capacidade C

Valor dos artigos: $V = (v_1\ v_2\ \ldots\ v_n)$
Peso dos artigos: $W = (w_1\ w_2\ \ldots\ w_n)$

$m_{vw}(0..n, 0..C)$: $m_{vw}(i, j)$ é o maior valor possível com os artigos $1..i$ e capacidade j, dados os valores V e os pesos W

$$m_{vw}(i,j) = \begin{cases} 0 & \text{se } i = 0 \vee j = 0 \\ m_{vw}(i-1, j) & \text{se } i, j > 0 \wedge w_i > j \\ \max \left\{ \begin{array}{c} v_i + m_{vw}(i-1, j-w_i), \\ m_{vw}(i-1, j) \end{array} \right\} & \text{se } i, j > 0 \wedge w_i \leq j \end{cases}$$

Chamada inicial da função

Valor máximo total do conteúdo do saco: $m_{vw}(n, C)$

Dilema do ladrão — Versão 2

Cálculo iterativo de $m[n, C]$

BOTTOM-UP-MAX-LOOT-VALUE(V, W, c)

```
n <- |v|
let m[0..n, 0..c] be a new array

for i <- 1 to n do                       // caso base
  m[i, 0] <- 0
for j <- 0 to c do
  m[0, j] <- 0

for i <- 1 to n do                       // casos recursivos
  for j <- 1 to c do
    if w[i] > j then
      m[i, j] <- m[i - 1, j]
    else if v[i] + m[i - 1, j - w[i]] >= m[i - 1, j] then
      m[i, j] <- v[i] + m[i - 1, j - w[i]]
    else
      m[i, j] <- m[i - 1, j]

return m[n, c]
```

Dilema do ladrão — Versão 2

Análise da complexidade (1)

BOTTOM-UP-MAX-LOOT-VALUE($\mathrm{v}_1 \ldots \mathrm{v}_n$, $\mathrm{w}_1 \ldots \mathrm{w}_n$, c)

Ciclo 3–4 é executado n vezes

Ciclo 5–6 é executado $c + 1$ vezes

Ciclo 7–14 é executado n vezes

Ciclo 8–14 é executado c vezes em cada iteração do ciclo 7–14

Complexidade temporal $\Theta(nc)$

Complexidade espacial $\Theta(nc)$

Dilema do ladrão — Versão 2
Análise da complexidade (2)

A complexidade é expressa em função da dimensão das entradas

Entrada	Dimensão	
$v_1 \ldots v_n$	n	
$w_1 \ldots w_n$	n	
c	$\log c = b$	(nº de algarismos de c)

A complexidade temporal de BOTTOM-UP-MAX-LOOT-VALUE é, então

$$\Theta(nc) = \Theta(n\,2^b)$$

que é exponencial na dimensão de c

Trata-se de um algoritmo pseudo-polinomial

Dilema do ladrão — Versão 2

Construção da solução

BOTTOM-UP-MAX-LOOT(V, W, c)

```
n <- |v|
let m[0..n, 0..c] and a[0..n, 1..c] be new arrays
for i <- 1 to n do                       // caso base
  m[i, 0] <- 0
for j <- 0 to c do
  m[0, j] <- 0
  a[0, j] <- 0
for i <- 1 to n do                       // casos recursivos
  for j <- 1 to c do
    if w[i] > j then
      m[i, j] <- m[i-1, j]
      a[i, j] <- a[i-1, j]
    else if v[i] + m[i-1, j-w[i]] >= m[i-1, j] then
      m[i, j] <- v[i] + m[i-1, j-w[i]]
      a[i, j] <- i
    else
      m[i, j] <- m[i-1, j]
      a[i, j] <- a[i-1, j]
return m and a
```

Dilema do ladrão — Versão 2

Artigos a escolher

$a[0..n, 1..c]$: $a[i, j]$ é o último dos artigos $1..i$ a escolher para obter o maior valor para a capacidade j

PRINT-LOOT(a, w, c)

```
n <- |w|
while c > 0 and a[n, c] != 0 do
  k <- a[n, c]
  print "article: " k
  c <- c - w[k]
  n <- k - 1
```