

Algoritmos Greedy - Seleção de Actividades

Problema [Seleção de Actividades]

Input: • $S = \{a_1, \dots, a_n\}$

- a_i executa no intervalo $[s_i, f_i]$

Output: • $S' \subseteq S$ de tamanho máximo

tal que todas as actividades em

S' são mutuamente competitivas.

- Duas actividades a_i e a_j são mutuamente competitivas se $[s_i, f_i] \cap [s_j, f_j] = \emptyset$

Exemplo:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	12	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- Mutuamente Competitivas (exemplos):

- Mutuamente Incompetitivas (exemplos):

Algoritmos Greedy - Seleção de Actividades

Problema [Seleção de Actividades]

Input: • $S = \{a_1, \dots, a_n\}$

- a_i executa no intervalo $[s_i, f_i]$

Output: • $S^* \subseteq S$ de tamanho máximo

tal que todas as actividades em

S^* são mutuamente competitivas.

- Duas actividades a_i e a_j são mutuamente competitivas se $[s_i, f_i] \cap [s_j, f_j] = \emptyset$

Exemplo:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	12	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- Mutuamente Competitivas (exemplos):

- $(1, 4), \dots, (8, 11)$

- Mutuamente Incompetitivas (exemplos):

- $(1, 2), \dots$

Seleção de Actividades

Objetivo: Determinar o maior conjunto de actividades mutuamente compatíveis.

Escolha Greedy: Dado um conjunto de actividades S , determinar uma actividade a que pertence a um conjunto máximo de actividades mutuamente compatíveis.

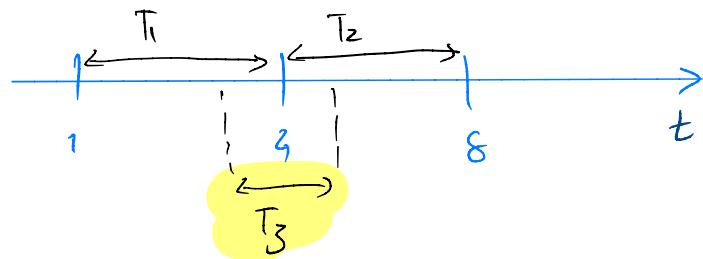
Estratégia 1: Seleccionar a actividade de menor duração.

Seleção de Actividades

Objetivo: Determinar o maior conjunto de actividades mutuamente compatíveis.

Escolha Greedy: Dado um conjunto de actividades S , determinar uma actividade a que pertence a um conjunto máximo de actividades mutuamente compatíveis.

Estratégia 1: Seleccionar a actividade de menor duração.



Wrong!

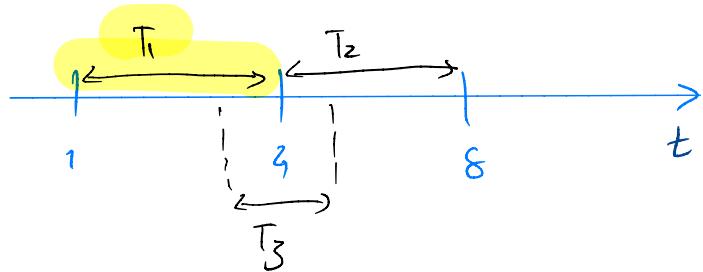
Seleção de Actividades

Objetivo: Determinar o maior conjunto de actividades mutuamente compatíveis.

Escolha Greedy: Dado um conjunto de actividades S , determinar uma actividade a que pertence a um conjunto máximo de actividades mutuamente compatíveis.

Estratégia 2: Seleccionar a actividade com menor tempo de fim.

(Ideia: Esta é a actividade q libera mais tempo para as outras actividades executarem)



Seleção de Actividades

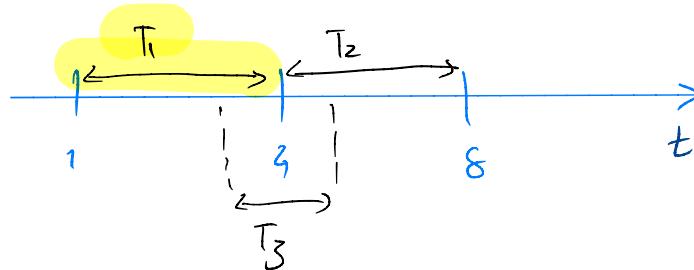
Estratégia 2: Selecionar a actividade com menor tempo de fim.

Lema [Seleção de Actividades - Escolha Greedy]

Seja S um conjunto de actividades e seja a_0 a actividade com menor tempo de fim em S ; então a_0 pertence a conjunto máximo de actividades mutuamente competitivas S^* .

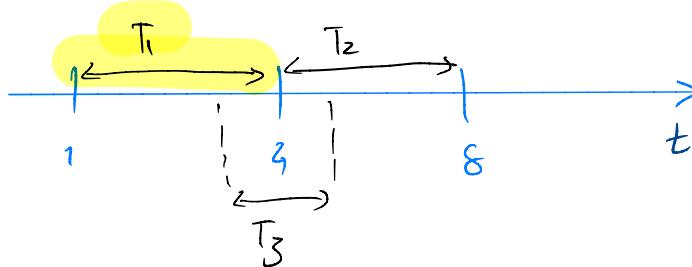
Prova:

- Seja S^* um conjunto máximo de actividades mutuamente competitivas.
- Se $a_0 \in S^*$, não há razão a provar.
- Suponhamos que $a_0 \notin S^*$; seja a_0' a actividade com menor tempo de fim em S^* . Definimos o seguinte conjunto:
$$\hat{S}^* = (S^* \setminus \{a_0'\}) \cup \{a_0\}$$
- Por definição, temos que $|\hat{S}^*| = |S^*|$
Pois juntar todas as actividades em \hat{S}^* são mutuamente competitivas $\Leftrightarrow a_0$ é competitivo com todas as actividades em $\underline{S^* \setminus \{a_0'\}}$.



Seleção de Actividades

Estratégia 2: Selecionar a actividade com menor tempo de fim.



Lema [Seleção de Actividades - Escolha Greedy]

Seja S um conjunto de actividades e seja a_0 a actividade com menor tempo de fim em S ; então a_0 pertence a conjunto máximo de actividades mutuamente compatíveis S^* .

Prova:

To prove:

a_0 é compatível com todas as actividades em $S^* \setminus \{a_0\}$.

* Tome-se $a_k \in S^* \setminus \{a_0\}$.

$$\Rightarrow s_k > f_0'$$

$$\Rightarrow f_0' > f_0$$

$$\Rightarrow s_k > f_0 \Rightarrow [s_0, f_0) \cap [s_k, f_k) = \emptyset$$

■

Seleção de Actividades - Algoritmo greedy

Activity Selector (s, f)

// Actividades estão ordenadas

$n = s.length;$

por tempo de fim

$A = \{a_1\};$

$k = 1;$

for $i=2$ to n

if $s[i] \geq f[k]$

$A = A \cup \{a_i\}; k=i$

return A

Complexidade:

Selección de Actividades - Algoritmo greedy

Activity Selector (s, f)

// Actividades están ordenadas

$n = s.length;$

por tiempo de fin

$A = \{a_i\};$

$k = 1;$

for $i=2$ to n

if $s[i] \geq f[k]$

$A = A \cup \{a_i\}; k=i$

return A

Complejidad: $O(n)$

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	12	12
f_i	4	5	6	7	9	11	10	11	12	14	16

Selección de Actividades - Algoritmo greedy

Activity Selector (s, f)

// Actividades están ordenadas

$n = s.length;$
por tiempo de fin

$A = \{a_i\};$

$k = 1;$

for $i=2$ to n

if $s[i] \geq f[k]$

$A = A \cup \{a_i\}; k=i$

return A

Complejidad: $O(n)$

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	12	12
f_i	4	5	6	7	9	11	10	11	12	14	16

$$S^* = \{1, 4, 8, 11\}$$

Códigos de Huffman

- Um código binário sobre um alfabeto \mathcal{C} é uma função $BC: \mathcal{C} \rightarrow \{0,1\}^*$
- Tipos de Código
 - Comprimento Fixo (Fixed Length Code)
Todos os elementos de \mathcal{C} são associados a uma sequência de 0's e 1's do mesmo tamanho
 - Comprimento Variável (Variable Length Code)
O código pode associar sequências (de 0's e 1's) de comprimento diferente a diferentes elementos de \mathcal{C} .

Discussão: Quando é que faz sentido usar códigos de comprimento variável?

Fixed-length Code

- a - 00
- b - 01
- c - 10

Variable-length Code

- a - 0
- b - 10
- c - 11

Exemplo

$$C = \{a, b, c, d\}$$

a - 0

• Como descodificar 001?

b - 01

c - 10

d - 1

Exemplo

$$C = \{a, b, c, d\}$$

a - 0

b - 01

c - 10

d - 1

• Como descoifrar 001?

- ab

- aad

Ambiguidade!

Definição III.2 [Códigos Livres de Prefixo / Prefix-Free Codes]

Um código binário Code sobre um alfabeto C diz-se livre de prefixo se para quaisquer $c_1, c_2 \in C$, nem $Code(c_1)$ é prefixo de $Code(c_2)$, nem vice-versa.

Variable-Length Code

a - 0

b - 10

c - 11

Exemplo: Código Livre de Prefixo

Códigos de Prefixo Óptimos

- Cada símbolo do alfabeto de input C é associado a uma frequência f .
- Objetivo: Construir um código que minimize o nº médio de bits necessários para representar cada símbolo

C	f	FLC	VLC
A	60%	00	0
B	25%	01	10
C	10%	10	110
D	5%	11	111

• Comprimento Médio por Caracter

- FLC: 2

- VLC: $1 \times 0.6 + 2 \times 0.25 + 3 \times 0.1 + 3 \times 0.05 = 1.55$

Lóculos Binários \leftrightarrow Árvore Binária

- Fixed-length Code

a - 00

b - 01

c - 10

- Variable-length Code (Prefix-free)

a - 0

b - 10

c - 11

- Variable-length Code (Non-Prefix-free)

a - 1

b - 10

c - 11

Lógitos Binários \leftrightarrow Árvores Binárias

- Fixed-length Code

a - 00

b - 01

c - 10

- Variable-Length Code (Prefix-free)

a - 0

b - 10

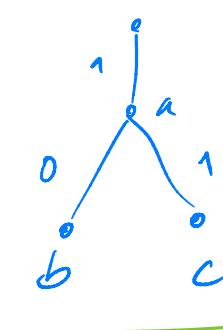
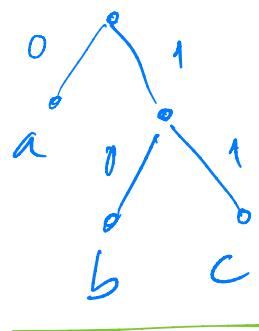
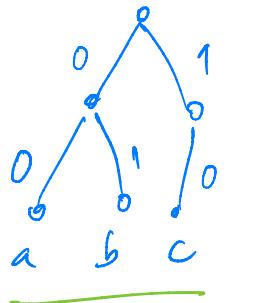
c - 11

- Variable-Length Code (Non-Prefix-free)

a - 1

b - 10

c - 11



Facto 1: Um código binário é fixo de prefixo se e só se a sua representação em árvore binária as únicas labelled nodes são as folhas.

Facto 2: O tamanho do código necessário para codificar o elemento c é C é dado pela profundidade da folha associada a c.

Códigos de Prefixo - Sumário

- Códigos binários podem ser representados por árvores binárias
- Códigos livres de prefixo também podem a árvores binárias cujas folhas estão anotadas com os elementos do alfabeto C
- $f(c)$ denota a frequência do elemento c
- $\ell_T(c)$ denota a profundidade da folha associada a c na árvore T ;
pontinho $\ell_T(c) \Rightarrow$ tamanho do código associado a c .

Seja T uma árvore binária que representa um código-binário sobre um alfabeto C com frequências f ; o custo de T é dado pelo seguinte formulário:

$$B(T) = \sum_{c \in C} f(c) \cdot \ell_T(c)$$

Gol: Dado um alfabeto C com frequências dadas por f , o nosso objectivo é calcular a árvore binária T de menor custo.

} Fórmula de Custo

Algoritmo de Huffman

Huffman(C)

$n := |C|$

let Q be a min-priority queue with content C

for $i = 1$ to $n - 1$

| $z := \text{NewNode}()$

| $x := \text{ExtractMin}(Q)$

| $y := \text{ExtractMin}(Q)$

| $z.\text{left} := x;$

| $z.\text{right} := y;$

| $z.\text{freq} := x.\text{freq} + y.\text{freq};$

| $\text{InsertKey}(Q, z)$

return $\text{ExtractMin}(Q)$

Algoritmo de Huffman

Huffman (C)

$n := |C|$

let Q be a min-priority queue with content C

for $i = 1$ to $n - 1$

| $z := \text{NewNode}()$

| $x := \text{ExtractMin}(Q)$

| $y := \text{ExtractMin}(Q)$

| $z.\text{left} := x;$

| $z.\text{right} := y;$

| $z.\text{freq} := x.\text{freq} + y.\text{freq};$

| $\text{InsertKey}(Q, z)$

return $\text{ExtractMin}(Q)$

Complexidade: $O(n \cdot \log n)$

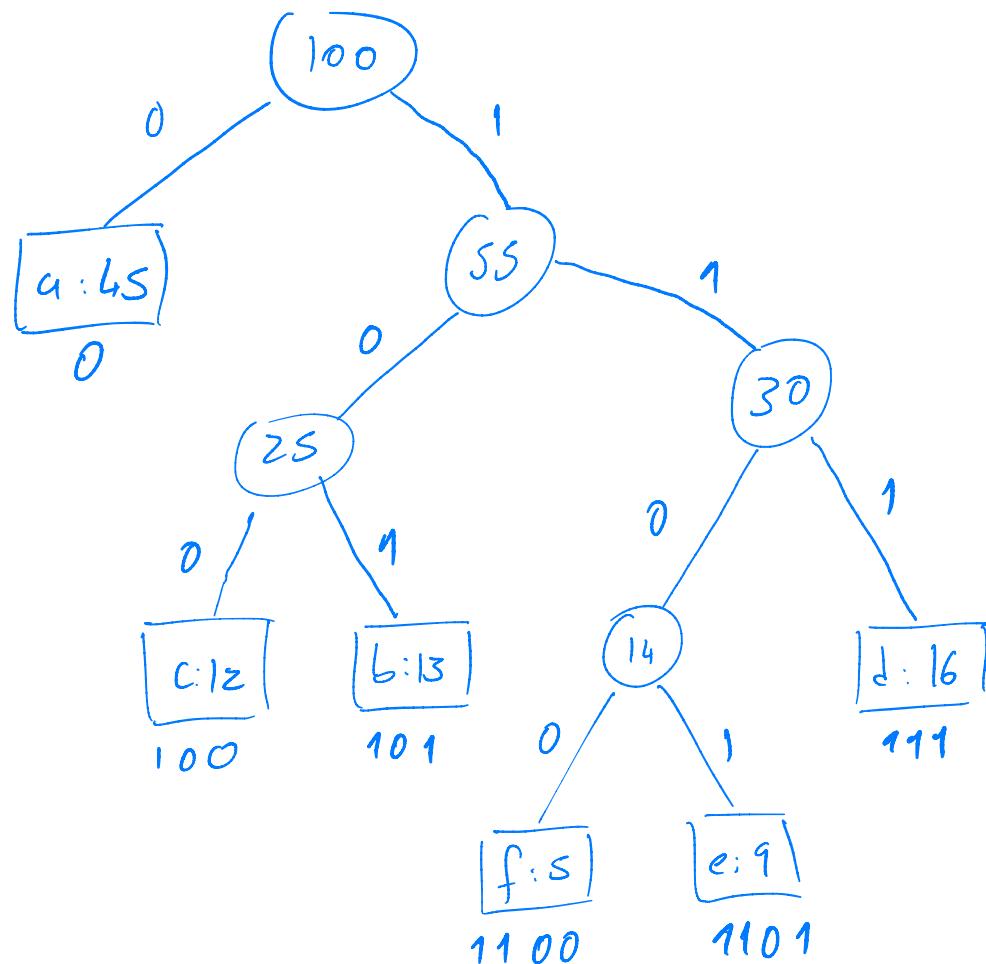
com $n = |C|$

Algoritmo de Huffman - Exemplo

- f: 5, e: 9, c: 12, b: 13, d: 16, a: 45

Algoritmo de Huffman - Exemplo

- f: 5, e: 9, c: 12, b: 13, d: 16, a: 45



Lema [Huffman - Escolha Greedy]

Seja C um alfabeto com função de frequências f e $a \neq b$ os elementos de C com menor frequência, existe uma árvore binária óptima para C em que $a \neq b$ são irmãos.

- Seja T uma árvore binária óptima para C .

Dois casos a considerar:

- $a \neq b$ são irmãos em T - Dado
- $a \neq b$ não são irmãos em T

- $a \neq b$ não são irmãos em T .

Ideia: construir uma árvore binária para C , T' , tal que $a \neq b$ são irmãos em T' e $B(T') \leq B(T)$. (pois T é óptima, concluímos que $B(T') = B(T)$, donde segue que T' é também óptima para C).