

Summary

- Selección de Actividades
- Problema de Machila Fraccionaria
- Códigos de Huffman

Aula 15 -

Algoritmos greedy

I - Problema de Seleção de Actividades

Definição I.1 [Problema de Seleção de Actividades]

Seja $S = \{a_1, \dots, a_n\}$ um conjunto de actividades cada uma com tempo de início s_i e tempo de fim f_i (com $0 \leq s_i < f_i < \infty$); o objectivo do problema consiste em seleccionar um conjunto máximo de actividades mutuamente compatíveis.

- **Intervais Competitivos:** Duas actividades são compatíveis se os seus intervalos não se interseccionam.

$$[s_i, f_i] \cap [s_j, f_j] = \emptyset$$

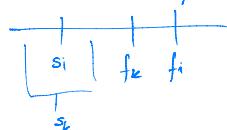
Lema I.1 [Seleção de Actividades - Greedy Choice]

Seja a_k a actividade com menor tempo de fim num conjunto de actividades S . Então a_k pertence a um conjunto máximo de actividades de S mutuamente compatíveis.

Prova:

1) Seja A um conjunto máximo de actividades de S mutuamente compatíveis. Se $a_k \in A$, o resultado está provado.

2) Suponhamos que $a_k \notin A$. Seja a_i a actividade c/ menor tempo de fim em A . Temos que:



3) $\forall a_j \in A \setminus \{a_i\}, s_j \geq f_i$

 $(\text{Se } s_j < f_i \Rightarrow f_j \leq s_i \Rightarrow i \text{ não é a actividade com menor tempo de fim})$

4) De (3)+(2), concluímos que: $\forall a_j \in A \setminus \{a_i\}, s_j \geq f_k$
 De onde segue que a_k é compatível com $A \setminus \{a_i\}$, o que nos permite concluir que o conjunto $(A \setminus \{a_i\}) \cup \{a_k\}$ é um conjunto de actividades mutuamente compatíveis de cardinalidade máxima.

■

Observação:

$$S_j = \{a_k \mid s_k \geq f_j\} \Rightarrow \text{actividades em } S \text{ q' começam depois de } a_j \text{ terminar}$$

$S \Rightarrow$ encontrar a actividade a_k em S com menor tempo de fim

Qualquer outra actividade no conjunto A tem de começar depois de a_k terminar.

Repetir para S_j .

(2)

Actividades Selección (s, f) // Actividades están ordenadas

$n = s.length;$

$A = \{a_i\};$

$k = 1;$

for $i=2$ to n

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

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

return A

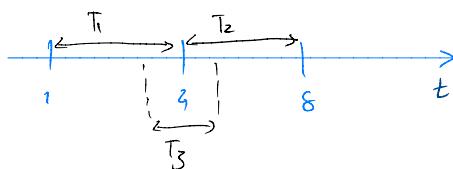
Ejemplo:

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

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

$$A = \{a_1, a_4, a_8, a_{11}\}$$

Observación: A escala de la actividad con la duración mínima
não é greedy.



II. Problema da Mochila Fracionária

- Mochila com capacidade W
- n itens com pesos: w_1, \dots, w_n
valores: v_1, \dots, v_n
- Poderemos levar uma quantidade fracionária de cada item: $0 \leq x_i \leq w_i$

$$\max \sum_{i=0}^n x_i \cdot \left(\frac{v_i}{w_i} \right)$$

$\forall i, 0 \leq x_i \leq w_i$

$$\sum_{i=0}^n x_i \leq W$$

- Para cada item i , calcula v_i/w_i

- Determinar o item k com maior valor v_k/w_k

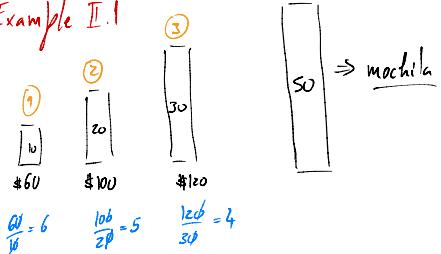
$$(x_k, W') = \begin{cases} \left(\frac{w_k}{W}, 0\right) & \text{se } W \leq w_k \\ (1, V - w_k) & \text{c.c.} \end{cases}$$

Fractional Knapsack (W, v, \checkmark)

```

let x[1..n] be a new array initialized to 0
for i = 1 to n
    if (w[i] < W)
        | x[i] = 1; W = V - v[i]
    else // w[i] ≥ V
        | x[i] = w[i]/V; return x
return x
  
```

Example II.1



$$x_1 = 1, \quad W = 50 - 10 = 40$$

$$x_2 = 1, \quad W = 40 - 20 = 20$$

$$x_3 = \frac{20}{30} = \frac{2}{3}, \quad W = 0$$

$$C = 1 \times 60 + 1 \times 100 + \frac{2}{3} \times 120$$

$$= 60 + 2 \times 40 = 160 + 80 = 240$$

III - Códigos de Huffman

Definição III.1 [Código & Tipos de Código]

- Dado um alfabeto C , um código binário sobre C é uma função \bar{g} que mapeia cada elemento de C numa sequência de 0's e 1's; formalmente, um código binário é uma função $\text{Code} : C \rightarrow \{0,1\}^*$.
- Um código binário diz-se de comprimento fixo (*fixed length binary code*) se todos os elementos de C são mapeados em sequências de caracteres do mesmo tamanho.
- Um código binário diz-se de comprimento variável (*variable length binary code*) se os elementos de C são mapeados em sequências de 0's e 1's de comprimento variável.

Exemplo III.1 [Códigos Binários] $C = \{a, b, c\}$

Fixed-length Code	Variable-length Code
$a - 00$	$a - 0$
$b - 01$	$b - 10$
$c - 10$	$c - 11$

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

Exemplo III.2 [Variable Length Code - Ambiguidade]

$C = \{a, b, c, d\}$	$a - 0$	• Como descodificar 001?
	$b - 01$	$- ab$
	$c - 10$	$- aad$
	$d - 1$	$\left. \begin{array}{l} \text{Não sabemos!} \\ \text{O código é ambíguo.} \end{array} \right\}$

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 $\text{Code}(c_1)$ é prefixo de $\text{Code}(c_2)$, nem vice-versa.

Variable-length Code	$\left. \begin{array}{l} a - 0 \\ b - 10 \\ c - 11 \end{array} \right\}$	Exemplo: Código Livre de Prefixo

- Como construir códigos óptimos?

Exemplo III.3 [Construção de Códigos Óptimos]

C	f	FLC	VLC	→ Comprimento Médio por Carácter
A	60%	00	0	
B	25%	01	10	- FLC: 2
C	10%	10	110	
D	5%	11	111	- VLC: $1 \times 0.6 + 2 \times 0.25 + 3 \times 0.1 + 3 \times 0.05 = 1.55$

- Objectivo [1º Formulação] Determinar o código de prefixo óptimo para um dado alfabeto C com frequências dadas por f.

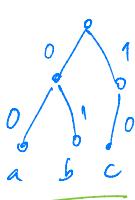
Exemplo III.4 [Códigos Binários ↔ Árvore Binária]

Fixed-length code

a - 00

b - 01

c - 10

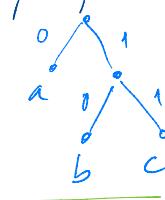


Variable-length code (Prefix-free)

a - 0

b - 10

c - 11

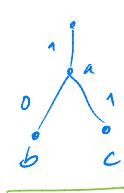


Variable-length code (Non-Prefix-free)

a - 1

b - 10

c - 11



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

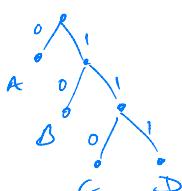
Exemplo III.5 [Descodificação]

A - 0

B - 10

C - 110

D - 111



- Como descodificar 0110111?

* ACD

(Seguir os ramos da árvore; quando chegarmos a uma folha voltarmos à raiz).

Nota: Tamanha do código necessário para codificar o elemento c ∈ C é dada pela profundidade da folha associada a c.

Suuncião

- Códigos binários podem ser representados por árvores binárias
- Nós livres de profundo (nós folhas) podem a árvores binárias cujas folhas estão anotadas com os elementos do alfabeto C
- $f(c)$ denota a frequência do elemento c
- $d_T(c)$ denota a profundidade da folha associada a c na árvore T ;
- ponto a $d_T(c) \rightarrow$ comprimento do código associado a c .

Definição III.3 [Fusco de Custo]

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:

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

• Objectivo [2º Formulário]

Dada um alfabeto C com frequências dadas por f , o nosso objectivo é calcular a árvore binária T de menor custo $B(T)$

Algoritmo de Huffman

Huffman(C)
 $n := |C|$

Let Q be a min-heapify queue with content C

```

for i=1 to n-1
  z := NewNode()
  x := ExtractMin(Q)
  y := ExtractMin(Q)
  z.left := x;
  z.right := y;
  z.freq := x.freq + y.freq;
  InsertKey(Q, z)
return ExtractMin(Q)

```

Complexidade: $O(n \cdot \lg n)$
 com $n = |C|$

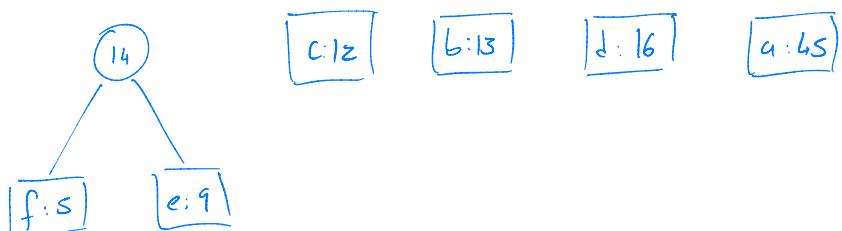
Exemplo III.6 [Códigos de Hoffman]

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

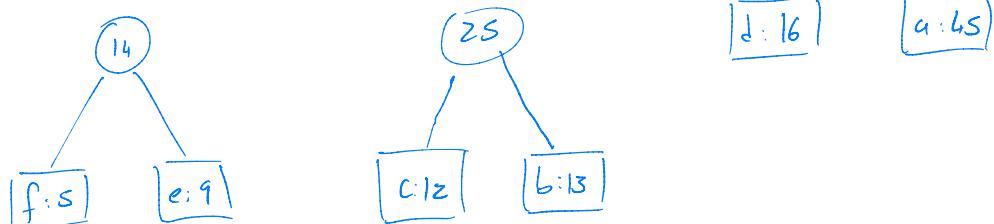
0



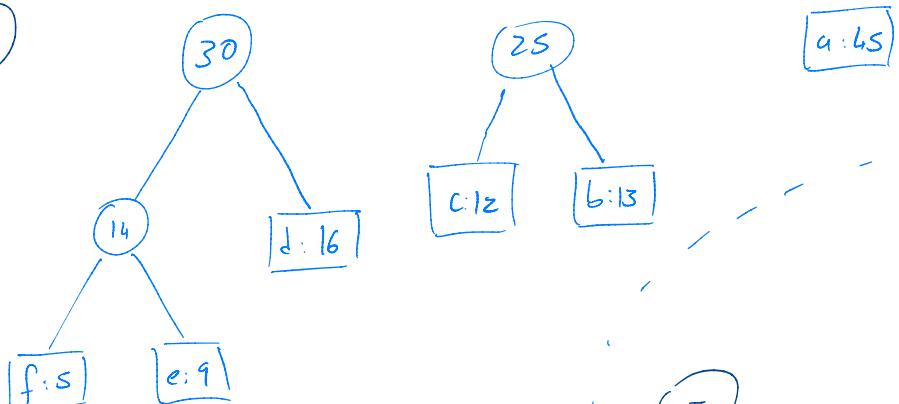
1



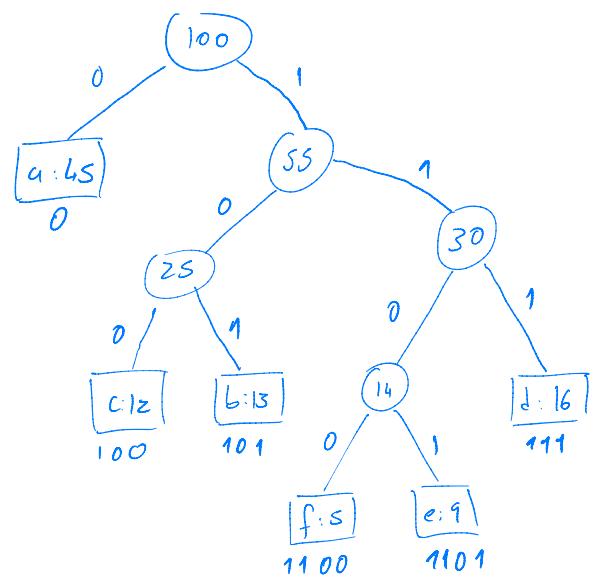
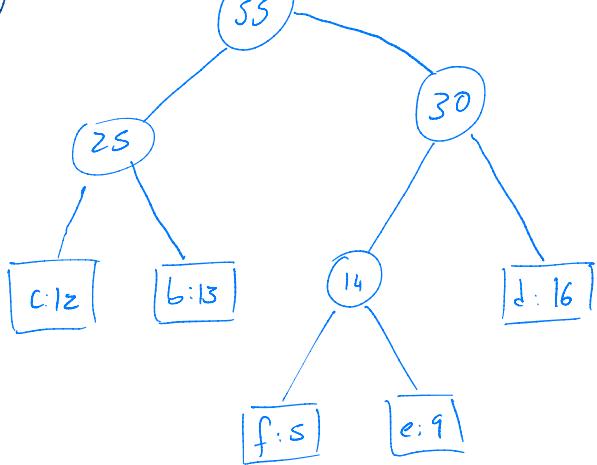
2



3



4



Lema III.1 [Huffman - Escolha Greedy]

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

Prova:

- Seja T uma árvore binária óptima para C .

Dois casos a considerar:

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

- $a \in b$ não são irmãos em T .

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

- Seja T' a árvore que se obtém a partir de T fazendo o irmão de a por b . Seja d o irmão de a em T .

$$\begin{aligned}
 B(T) - B(T') &= \sum_{c \in C} f(c) \cdot d_T(c) - \sum_{c \in C} f(c) \cdot d_{T'}(c) \\
 &= (f(b) \cdot d_T(b) + f(d) \cdot d_T(d)) - (f(b) \cdot d_{T'}(b) + f(d) \cdot d_{T'}(d)) \\
 &= (f(b) \cdot d_T(b) + f(d) \cdot d_T(d)) - (f(b) \cdot d_T(d) + f(d) \cdot d_T(b)) \\
 &= f(b) \cdot (d_T(b) - d_T(d)) + f(d) \cdot (d_T(d) - d_T(b)) \\
 &= f(b) \cdot (\underbrace{d_T(b) - d_T(a)}_{\geq 0}) + f(d) \cdot (\underbrace{d_T(d) - d_T(a)}_{\geq 0}) \\
 &= (\underbrace{d_T(a) - d_T(b)}_{\geq 0}) \cdot (\underbrace{f(d) - f(b)}_{\geq 0})
 \end{aligned}$$

$$\geq 0$$

Teorema III.2 [Huffman - Correção]

Seja C um alfabeto com função de frequências f , o algoritmo de Huffman calcula um código binário óptimo para C .

Prova

* Dado C com função de frequências f , provaremos que o algoritmo de Huffman calcula uma árvore binária óptima para C por indução no tamanho de C .

- Base $|C|=2$

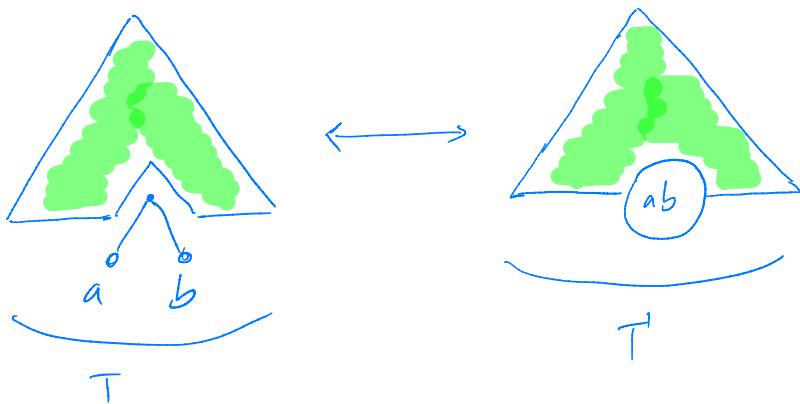
$$C = \{a, b\} \quad \Delta_a \wedge_b \quad B(T) = f(a) + f(b)$$

- Passo, $|C|=n=n+1$

- Sejam a e b os elementos de C com menor frequência.

- O Lema II.1 garante que há uma árvore óptima na qual a e b são irmãos.

Então só temos de alterar para as árvores em que a e b são irmãos:

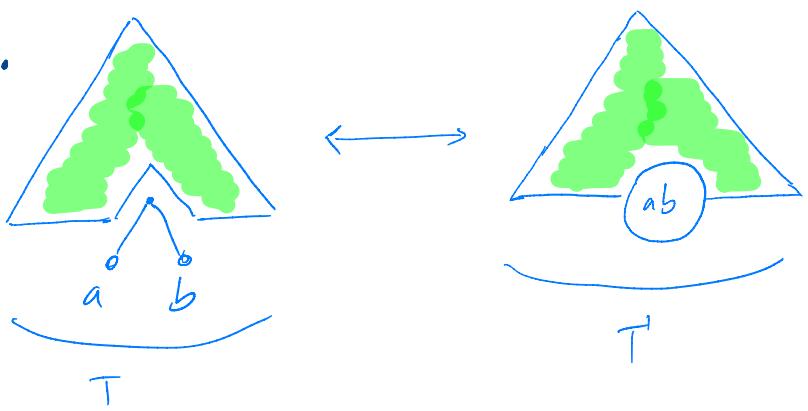


- Construímos uma nova instância instância do problema de tamanho $n-1$.

$$- C' = (C \setminus \{a, b\}) \cup \{x_{ab}\}$$

$$- f' = f|_{C \setminus \{a, b\}} [x_{ab} \mapsto f(a) + f(b)]$$

- A hipótese de indução garante que o algoritmo de Huffman calcula a árvore binária óptima para o alfabeto C' com frequências dadas por f' .



Falke apenas argumentar que a melhora solubilidade para C corresponde necessariamente a meliora solubilidade para A.

$$\begin{aligned}
 B(T) - B(T') &= \sum_{c \in C} f(c) \cdot d_T(c) - \sum_{c \in C'} f'(c) \cdot d_{T'}(c) \\
 &= f(a) \cdot d_T(a) + f(b) \cdot d_T(b) - \underbrace{f'(x_{ab}) \cdot d_{T'}(x_{ab})}_{d} \\
 &= \cancel{d} (f(a) + f(b)) - \cancel{d} (f'(x_{ab})) \\
 &= \underbrace{f(a) + f(b)}_{k \Rightarrow \text{constante}}
 \end{aligned}$$

$$\underbrace{B(T) = B(T) - k}$$

\hookrightarrow A melhora em riqueza para C' corresponde à melhora inversa para C.