

Sumário

Algoritmos Greedy

- Propriedades
- Problema da Seleção de Actividades
- Códigos de Huffman

Aula 15



Algoritmo de Prim

Prim(G, w, r)

for each $v \in G.V$

$v.key = \infty; v.\pi = \text{nil}$

$r.key := 0;$

$A := \emptyset$

let Q be a min-priority queue with content $G.V$

while $Q \neq \emptyset$

let $u = \text{ExtractMin}(Q)$

$\{if(u \notin A) A := A \cup \{(u, \pi, u)\}\}$

for each $v \in G.Adj[u]$

$\{if(v.key > w(u, v)) \text{ do } v \in Q$

$v.key := w(u, v); v.\pi := u\}$

Invariante do Algoritmo de Prim

(I1)

(I2)

Algoritmo de Prim

Prim(G, w, r)

for each $v \in G.V$

$v.key = \infty; v.\pi = \text{nil}$

$r.key := 0;$

$A := \emptyset$

let Q be a min-priority queue with content $G.V$

while $Q \neq \emptyset$

let $m = \text{ExtractMin}(Q)$

$\{if(m \notin Q) A := A \cup \{(m, \pi, m)\}\}$

for each $v \in G.Adj[m]$

$\{if(v.key > w(m, v)) \text{ do } v \in Q\}$

$v.key := w(m, v); v.\pi := m$

Invariante do Algoritmo de Prim

(I1) $A = \{(v, \pi, v) \mid v \notin Q \wedge v.\pi \neq \text{nil}\}$
é um subconjunto de uma MST

(I2) $\forall v \in Q$.
 $m.key = \min \{w(m, v) \mid v \in V \setminus Q\}$

Algoritmo de Prim

Manutenção

$$\textcircled{I_1} \quad A' = A \cup \{(u, v, w) \}$$

- Há que provar que (u, v, w) é seguno para A

Invariante do Algoritmo de Prim

$$\textcircled{I_1} \quad A = \{(v, u, v) \mid v \in Q \text{ e } v \neq u\}$$

é um subconjunto de uma MST

$$\textcircled{I_2} \quad \forall u \in Q \quad m \cdot key = \min \{w(u, v) \mid v \in V \setminus Q\}$$

Algoritmo de Prim

Invariante do Algoritmo de Prim

(I₁) $A = \{(v.\pi, v) \mid v \in Q \text{ e } v.\pi \neq \text{nil}\}$
 é um subconjunto de uma MST

(I₂) $\forall u \in Q$.
 $m \cdot key = \min \{ w(u, v) \mid v \in V \setminus Q \}$

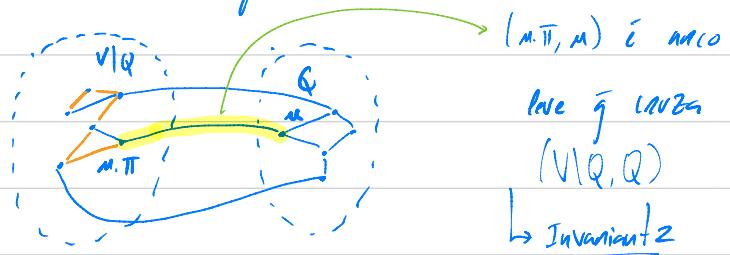
Manutenção

$$(I_1) A' = A \cup \{(m.\pi, m)\}$$

- Há que provar que $(m.\pi, m)$ é seguno para A

- Temos de encontrar um corte $(S, V \setminus S)$ que respeita A e para o qual $(m.\pi, m)$ seja ponte.

- $(V \setminus Q, Q)$ respeita A



Problema da Seleção de Actividades

Definição [Problema da 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 compatíveis.

- Duas actividades a_i e a_j são mutuamente compatíveis se $[t_i, f_i] \cap [t_j, f_j] = \emptyset$

Exemplo:

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

- Actividades Mutuamente Compatíveis:

- Actividades Mutualmente Incompatíveis:

Problema da Seleção de Actividades

Definição [Problema da 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 compatíveis.

- Dois actividades a_i e a_j são mutuamente compatíveis se $[t_i, f_i] \cap [t_j, f_j] = \emptyset$

Exemplo:

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

• Actividades Mutuamente Compatíveis:
 $(1, 4), \dots$

• Actividades Mutuamente Incompatíveis:
 $(10, 11), (1, 2), \dots$

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.

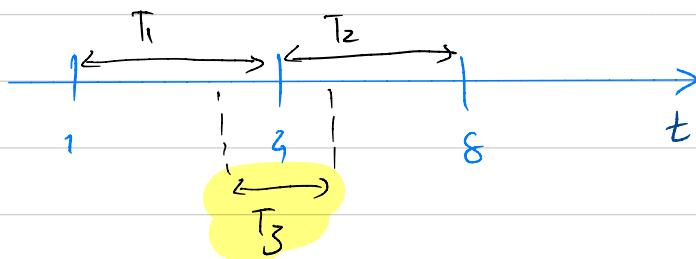
Estratégia 1: Seleccionar a actividade de menor duração.

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.

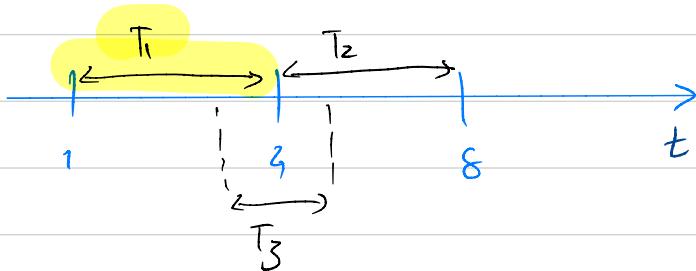
Estratégia 1: Seleccionar a actividade de menor duração.



Wrong!

Seleção de Actividades

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



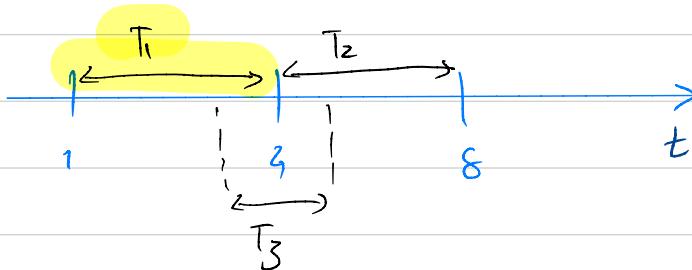
Lema [Seleção de Actividades - Escolha Greedy]

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

Prova:

Seleção de Actividades

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



Lema [Seleção de Actividades - Escolha Gráfica]

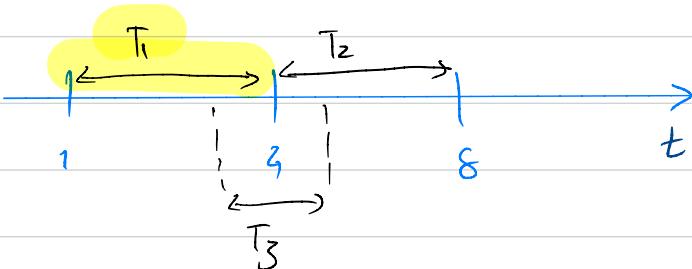
Seja S um conjunto de actividades e seja a_0 a actividade com menor tempo de fim em S ; então a_0 pertence a conjunto máximo de actividades mutuamente competitivas S^* .

Prova:

- Seja S^* um conjunto máximo de actividades mutuamente competitivas.
- Se $a_0 \notin S^*$, 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:
$$\hat{S}^* = (S^*) \setminus \{a_0'\} \cup \{a_0\}$$
- Da 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 $S^* \setminus \{a_0'\}$.

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

$$\Rightarrow s_k \geq f_0'$$

$$\Rightarrow f_0' \geq f_0$$

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

□

Seleção de Actividades - Algoritmo Greedy

Activity Selection (s, f)

$n = s.length$

$A := \{a_1\}$

$k := 1$

for $i=2$ to n

 if $s[i] > f[k]$

$A := A \cup \{a_k\}$

$k := i$

return A

// As actividades encontram-se inicialmente
ordenadas por tempo de fim

Análise de Complexidade

• $O(n)$

\Rightarrow Porque admitimos que as
actividades se encontram desde o
início correctamente ordenadas.

Seleção de Actividades - Algoritmo Greedy

Activity Selection (s, f)

$n = s.length$

$A := \{a_1\}$

$k := 1$

for $i=2$ to n

if $s[i] > f[k]$
 $A := A \cup \{a_k\}$

$k := i$

return A

// As actividades encontram-se inicialmente
ordenadas por tempo de fim

Exemplo:

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

Seleção de Actividades - Algoritmo Greedy

Activity Selection (s, f)

$n = s.length$

$A := \{a_1\}$

$k := 1$

for $i=2$ to n

if $s[i] > f[k]$
 $A := A \cup \{a_k\}$

$k := i$

return A

// As actividades encontram-se inicialmente
ordenadas por tempo de fim

Exemplo:

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

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

Códigos de Huffman

Definição [Código Binário]

Dado um alfabeto \mathcal{C} , um código binário sobre \mathcal{C} é uma função $BC : \mathcal{C} \rightarrow \{0, 1\}^*$, que mapeia os elementos de \mathcal{C} em sequências de 0's e 1's.

Tipos de Código

- Códigos de Comprimento Fixo
(Fixed length code)

- Códigos de Comprimento Variável
(Variable length code)

Todos os elementos de \mathcal{C} são mapeados em sequências de 0's e 1's do mesmo tamanho.

Fixed-length code
a - 00
b - 01
c - 10

Códigos que associam sequências de 0's e 1's de comprimento diferente a diferentes elementos de \mathcal{C}

Variable-length code
a - 0
b - 10
c - 11

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

Códigos de Huffman

- Qual o problema com o seguinte código?

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

a - 0

b - 01

c - 10

d - 1

Códigos de Huffman

- Qual o problema com o seguinte código?

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

a - 0

b - 01

c - 10

d - 1

- Como decodificar 0010?

- aada

- ab a

- aac

- ...

Definição [Códigos Livres de Prefixo (Prefix-tree codes)]

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

Código 1:

a - 0

b - 10

c - 11

Código 2: a - 0

b - 01

c - 10

d - 1

Códigos de Huffman

- Qual o problema com o seguinte código?

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

a - 0

b - 01

c - 10

d - 1

• Como decodificar 0010?

• aad a

• ab a

• aac

...

Definição [Códigos Livres de Prefixo (Prefix-tree codes)]

Um código binário C sobre um alfabeto \mathcal{C} diz-se **livre de prefixo** se e somente se para quaisquer elementos $c_1, c_2 \in \mathcal{C}$ nem $\text{Code}(c_1)$ é prefixo de $\text{Code}(c_2)$ nem vice-versa.

Código 1:

a - 0 livre de

b - 10 prefixo

c - 11

Código 2: a - 0

b - 01

c - 10

d - 1

} Não livre de prefixo

Códigos de Prefixo Óptimos

- Cada símbolo do alfabeto de input C é associado a uma frequência f .
- [Ex]: Construir um código que minimiza o n° médio de bits necessários para representar cada símbolo

Códigos Binarios \leftrightarrow Árboles Binarias

- 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

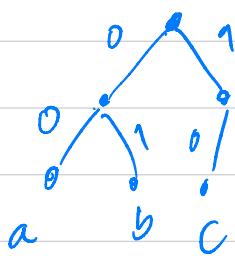
Códigos Binários \leftrightarrow Árvores Binárias

- Fixed-length code

a - 00

b - 01

c - 10



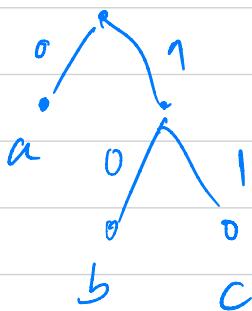
Facto 1: Um código binário é livre de prefixo se e na sua representação em árvore binária os únicos nós anotados com elementos de C são as folhas.

- Variable-length code (Prefix-free)

a - 0

b - 10

c - 11



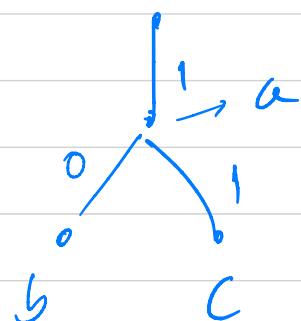
Facto 2: O tamanho do código necessário para codificar o elemento $c \in C$ é dado pela profundidade da folha associada a c .

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

a - 1

b - 10

c - 11



Códigos de Prefixo Óptimos

- Cada símbolo do alfabeto de input C é associado a uma frequência f .
- Objectivo (versão 1): Construir um código que minimize o nº médio de bits necessários para representar cada símbolo

Definição [Custo de Código Binário]

Seja T uma árvore binária que representa um código binário sobre um alfabeto C com frequências f ; o custo de T é definido como se segue:

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

Facto: A profundidade de L na árvore T corresponde ao tamanho do código que T associa a C .

- Objectivo (versão 2): Deu um alfabeto C e uma função de frequências f , calcule a árvore T que representa um código binário de C com menor valor de $B(T)$.

Códigos de Prefixo óptimos

Definição: Custo de Código Binário

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

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:

- VLC:

Códigos de Prefixo óptimos

Definição: Custo de Código Binário

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

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

→ Comprimento Médio por Caractere

- FLC: 2

$$\begin{aligned} - VLC: & 1 \times 0.6 + 2 \times 0.25 + 3 \times 0.1 + 3 \times 0.05 \\ & = 1.55 \end{aligned}$$

Códigos de Prefixo - Descodificação

Exemplo:

A - 0

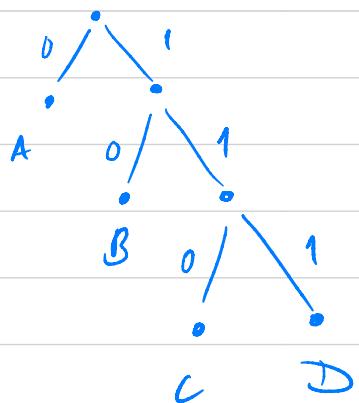
- Queremos descodificar: 0110 111

B - 10

C - 110

① Árvore Binária que representa o código

D - 111



② Descodificam: segui os ramos da árvore a partir da raiz quando chegarmos a uma folha voltaremos à raiz.

Códigos de Prefixo - Descodificação

Exemplo:

A - 0

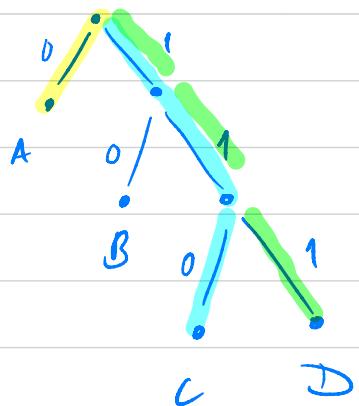
- Queremos descodificar: 0110 111

B - 10

C - 110

① Árvore Binária que representa o código

D - 111



② Descodificam: segui os ramos da árvore a partir da raiz quando chegarmos a uma folha voltaremos à raiz.

ACD

Códigos de Prefixo - Sómanio

- Códigos binários podem ser representados por árvores binárias
- Códigos Livres de Prefixo correspondem a árvores binárias cujas folhas estão anotadas com os elementos do alfabeto C .
- $f(c)$ - frequência do elemento c ($c \in C$)
- $d_T(c)$ - profundidade de c na árvore T
 \Rightarrow comprimento do código \bar{c} de T associado a c
- Custo do código representado por T :

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

Algoritmo de Huffman

Huffman(C)

$n := |C|$

let Q be a min-priority queue with content C

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

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

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

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

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

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

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

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

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 $n - 1$

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

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

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

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

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

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

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

return ExtractMin(Q)

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

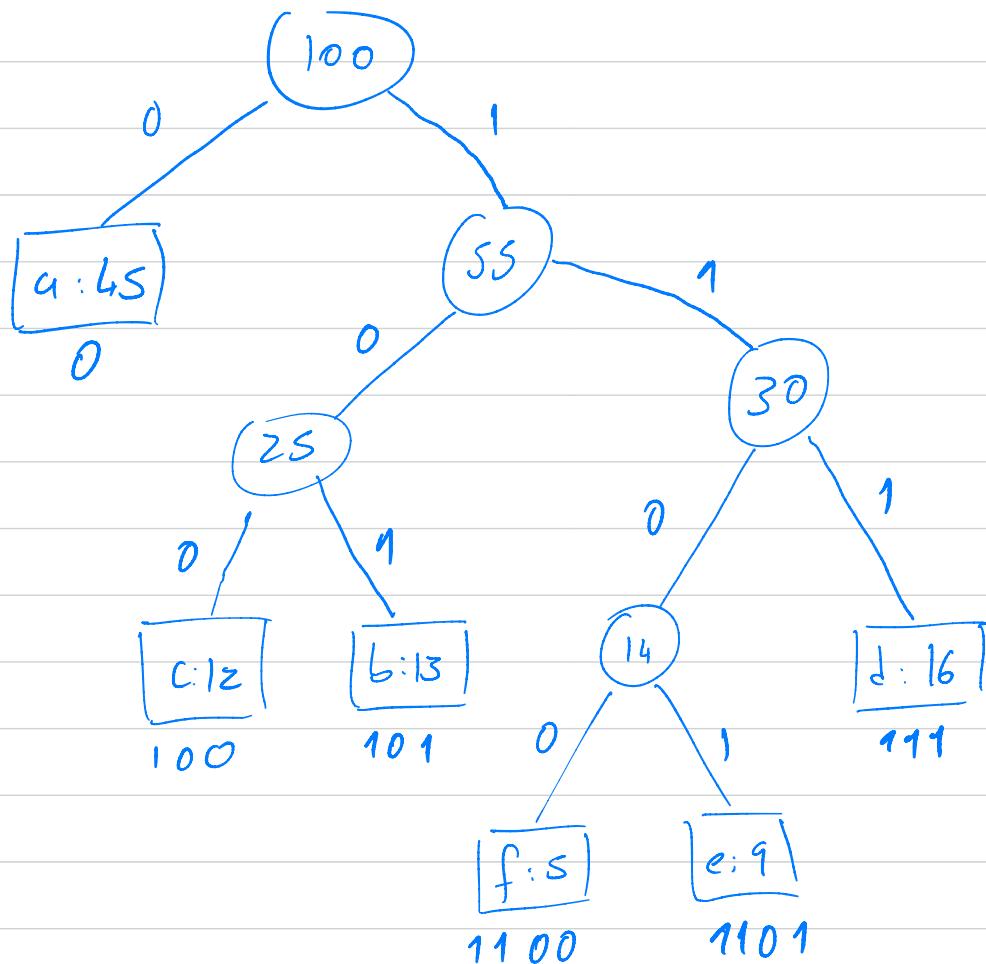
com $n = |C|$

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



Algoritmo de Huffman - Escolha óptima

Lema [Algoritmo de Huffman - Escolha Óptima]

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

Prova

- Seja T uma árvore binária óptima para (C, f) . Se $a \neq b$ são irmãos em T não há razão a provar.

Suponhamos então que $a \neq b$ não são irmãos em T . Seja T' obtida trocando o irmão de a por b em T .



- Resta provar que

$$B(T') \leq B(T)$$

Algoritmo de Huffman - Escolha óptima

Lema [Algoritmo de Huffman - Escolha Óptima]

Seja C um alfabeto e f função de frequências. Se $a \in C$ é o elemento de C com menor frequência, então, existe uma árvore binária óptima para $C \setminus \{a\}$ na qual $a \in C$ seja ínvel.

Prova

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

Teorema III.2 [Huffman - Comprovação]

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 .

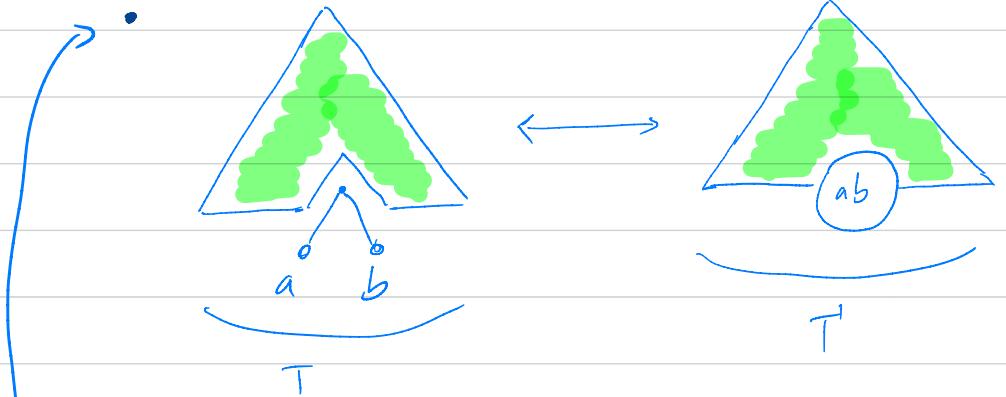
• Base $|C|=2 \quad C=\{a,b\}$  $B(T) = f(a) + f(b)$

• Passo, $|C|=n=n+1$

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

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

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

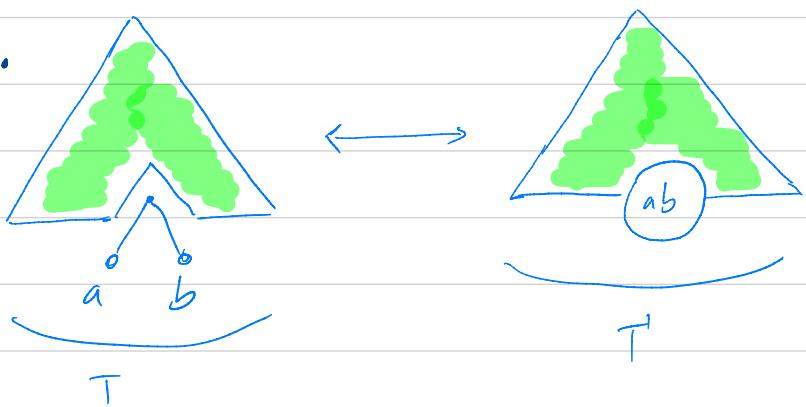


- Construiremos uma nova instância instância do problema de tamanho $n-1$.

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

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

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



Falta apenas argumentar que a melhor solução para (C, f) corresponde necessariamente à melhor solução para (C', f') .

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) \cdot d_T(c) - \sum_{c \in C'} f'(c) \cdot d_{T'}(c) \\ &= f(a) \cdot d_T(a) + f(b) \cdot d_T(b) - \underbrace{f'(x_{ab}) \cdot d_{T'}(x_{ab})}_{d} \\ &= (d+1)(f(a) + f(b)) - d(f(a) + f(b)) \end{aligned}$$

$$= \underbrace{f(a) + f(b)}_{k \Rightarrow \text{constante}}$$

$$B(T) = B(T) - k$$

↳ A melhor solução para C' corresponde à melhor solução para C .