Análise e Síntese de Algoritmos

Algoritmos Greedy [CLRS, Cap. 16]

2011/2012

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Árvores abrangentes
  - Caminhos mais curtos
  - Fluxos máximos
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Emparelhamento de Cadeias de Caracteres
  - Complexidade Computacional
  - Algoritmos de Aproximação

Resumo

1. Motivação
   - Selecção de Actividades
2. Caracteristicas Algoritmos Greedy
3. Exemplos
   - Problema da Mochila Fraccionário
   - Minimizar Tempo no Sistema
   - Códigos de Huffman

Técnicas para Síntese de Algoritmos

Técnicas para Síntese de Algoritmos

- Dividir para conquistar
- Programação dinâmica
- Algoritmos greedy
  - Estratégia: a cada passo da execução do algoritmo escolher opção que localmente se afigura como a melhor para encontrar solução óptima
    - Estratégia permite obter solução óptima?
  - Exemplo: Prim, Dijkstra

Motivação

Selecção de Actividades

- Seja $S = \{1, 2, \ldots, n\}$ um conjunto de actividades que pretendem utilizar um dado recurso
- Apenas uma actividade pode utilizar o recurso de cada vez
- Actividade i:
  - tempo de início: s_i
  - tempo de fim: f_i
  - execução da actividade durante $[s_i, f_i)$
- Actividades i e j compatíveis apenas se $[s_i, f_i)$ e $[s_j, f_j)$ não se intersectam
- Objectivo: encontrar conjunto máximo de actividades mutuamente compatíveis

Motivação

Selecção de Actividades

- Admitir que $f_1 \leq f_2 \leq \ldots \leq f_n$
- Escolha greedy:
  - Escolher actividade com o menor tempo de fim
- Porquê?
  - Maximizar espaço para restantes actividades serem realizadas

Seleccionar_Actividades_Greedy(s, f)

```
1  n ← length[s]
2  A ← {1}
3  j ← 1
4  for i ← 2 to n
5        do if s_i ≥ f_j
6              then A ← A ∪ {i}
7                   j ← i
8  return A
```

Motivação

Selecção de Actividades

- Algoritmo encontra soluções de tamanho máximo para o problema de selecção de actividades

Motivação

Selecção de Actividades

- Algoritmo encontra soluções de tamanho máximo para o problema de selecção de actividades
- Existe uma solução óptima que começa com escolha greedy, i.e. actividade 1
  - Seja A uma solução óptima que começa em k
  - Seja B uma solução óptima que começa em 1: $B = A - \{k\} \cup \{1\}$
  - $f_1 \leq f_k$
    - Actividades em B são mutuamente disjuntas e $|A| = |B|$
    - Logo, B é também solução óptima !

Motivação

Selecção de Actividades

- Algoritmo encontra soluções de tamanho máximo para o problema de selecção de actividades
- Após escolha greedy, problema reduz-se a encontrar solução para actividades compatíveis com actividade 1
  - Seja A solução óptima, e que começa em 1
  - $A' = A - \{1\}$ é solução óptima para $S' = \{i \in S : s_i \geq f_1\}$
  - Caso contrário, existiria uma solução $|B'| > |A'|$ para S' que permitiria obter solução B para S com mais actividades do que A; uma contradição !
- Aplicar indução no número de escolhas greedy
- Algoritmo calcula solução óptima !

Motivação

Caracteristicas Algoritmos Greedy

- Propriedade da escolha greedy
  - Óptimo (global) para o problema pode ser encontrado realizando escolhas locais óptimas
    (em programação dinâmica, esta escolha está dependente de resultados de sub-problemas)
- Sub-estrutura óptima
  - Solução óptima do problema engloba soluções óptimas para sub-problemas

Problema da Mochila Fraccionário

Definição do Problema

- Dados n objectos $(1, \ldots, n)$ e uma mochila
- Cada objecto tem um valor v_i e um peso w_i
- Peso transportado pela mochila não pode exceder W
- É possível transportar fracção x_i do objecto: $0 \leq x_i \leq 1$
- Objectivo: maximizar o valor transportado pela mochila e respeitar a restrição de peso
- Formulação Programação Linear:

$$\begin{aligned} \text{maximizar} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{sujeito a} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & 0 \leq x_i \leq 1 \end{aligned}$$

Problema da Mochila Fraccionário

Observações

- Soma do peso dos n objectos deve exceder peso limite W. Caso contrário a solução é trivial.
- Solução óptima tem que encher mochila completamente, $\sum x_i w_i = W$
- Caso contrário poderíamos transportar mais fracções, com mais valor !
- Complexidade Solução: $O(n)$ ou $O(n \lg n)$

Encher_Mochila_Greedy(v, w, W)

```
1  weight ← 0
2  while (weight < W)
3        do escolher objecto i com v_i / w_i máximo
4           if (w_i + weight ≤ W)
5              then x_i ← 1; weight ← weight + w_i
6              else x_i ← (W − weight)/w_i; weight ← W
```

Problema da Mochila Fraccionário

Optimalidade da Solução Greedy

Se objectos forem escolhidos por ordem decrescente de v_i/w_i, então algoritmo encontra solução óptima

- Admitir: $v_1/w_1 \geq \ldots \geq v_n/w_n$
- Solução calculada por algoritmo greedy: $X = (x_1, \ldots, x_n)$
  - Se $x_i = 1$ para todo o i, solução é necessariamente óptima
  - Caso contrário, seja j o menor índice para o qual $x_j < 1$
    - $x_i = 1, i < j$
    - $x_i = 0, i > j$
  - Relação de pesos: $\sum_{i=1}^{n} x_i w_i = W$
  - Valor da solução: $\sum_{i=1}^{n} x_i v_i = V(X)$

Problema da Mochila Fraccionário

Optimalidade da Solução Greedy

- Qualquer solução possível: $Y = (y_1, \ldots, y_n)$
  - Peso: $\sum_{i=1}^{n} y_i w_i \leq W$
  - Valor: $V(Y) = \sum_{i=1}^{n} y_i v_i$
- Relação X vs. Y:
  - Peso: $\sum_{i=1}^{n} (x_i - y_i) w_i \geq 0$
  - Valor: $V(X) - V(Y) = \sum_{i=1}^{n} (x_i - y_i) v_i = \sum_{i=1}^{n} (x_i - y_i) w_i (v_i/w_i)$
- Seja j o menor índice tal que $x_j < 1$. Casos possíveis:
  - $i < j \quad \Rightarrow \quad x_i = 1 \quad \wedge \quad x_i - y_i \geq 0 \quad \wedge \quad v_i/w_i \geq v_j/w_j$
  - $i > j \quad \Rightarrow \quad x_i = 0 \quad \wedge \quad x_i - y_i \leq 0 \quad \wedge \quad v_i/w_i \leq v_j/w_j$
  - $i = j \quad \Rightarrow \quad v_i/w_i = v_j/w_j$
- Verifica-se sempre que: $(x_i - y_i)(v_i/w_i) \geq (x_i - y_i)(v_j/w_j)$

Problema da Mochila Fraccionário

Optimalidade da Solução Greedy

- Considerando que $(x_i - y_i)(v_i/w_i) \geq (x_i - y_i)(v_j/w_j)$
- Verifica-se que:

$$V(X) - V(Y) = \sum_{i=1}^{n} (x_i - y_i) w_i (v_i/w_i) \geq (v_j/w_j) \sum_{i=1}^{n} (x_i - y_i) w_i \geq 0$$

- Logo, $V(X)$ é a melhor solução possível entre todas as soluções possíveis
- Algoritmo calcula solução óptima !

Minimizar Tempo no Sistema

Definição do Problema

- Dado um servidor com n clientes, com tempo de serviço conhecido (i.e. cliente i leva tempo t_i), objectivo é minimizar tempo total despendido no sistema (pelo total dos n clientes)
- Minimizar: $\sum_{i=1}^{n}$ (tempo total dispendido no sistema pelo cliente i)

Solução Greedy

Servir clientes por ordem crescente do tempo de serviço.

Minimizar Tempo no Sistema

Optimalidade da Solução Greedy

- $P = p_1 p_2 \ldots p_n$, permutação dos inteiros de 1 a n
- Com clientes servidos pela ordem P, tempo de serviço do cliente a ser servido na posição i é $s_i = t_{p_i}$ (por exemplo, $s_1 = t_{p_1} = t_5$)
- Tempo total passado no sistema por todos os clientes é:

  $$T(P) = \sum_{k=1}^{n} (n-k+1)s_k$$

  - s_1 aparece n vezes
  - s_2 aparece $n-1$ vezes
  - $\ldots$
  - s_n aparece 1 vez
- Assumir clientes não ordenados por ordem crescente de tempo de serviço em P
  - Então existem a e b, com $a < b$, e $s_a > s_b$

Minimizar Tempo no Sistema

Optimalidade da Solução Greedy

- Trocar ordem dos clientes a e b, de modo a obter ordem P'
- Corresponde a P com inteiros p_a e p_b trocados

$$T(P') = (n-a+1)s_b + (n-b+1)s_a + \sum_{k=1,k\neq a,b}^{n} (n-k+1)s_k$$

- Obtendo-se,

$$\begin{aligned} T(P) - T(P') &= (n-a+1)(s_a - s_b) + (n-b+1)(s_b - s_a) \\ &= (b-a)(s_a - s_b) > 0 \end{aligned}$$

- P' é uma ordem melhor (com menor tempo de serviço)
- Algoritmo calcula solução óptima !

Códigos de Huffman

Aplicação: Compressão de Dados

- Exemplo: Ficheiro com 100000 caracteres

	a	b	c	d	e	f
Frequência ($\times$1000)	45	13	12	16	9	5
Código Fixo	000	001	010	011	100	101

- Tamanho do ficheiro comprimido: $3 \times 100000 = 300000$ bits
- Código de largura variável pode ser melhor do que de largura fixa
  - Aos caracteres mais frequentes associar códigos de menor dimensão

Códigos de Huffman

Aplicação: Compressão de Dados

- Código de comprimento variável:

	a	b	c	d	e	f
Frequência ($\times$1000)	45	13	12	16	9	5
Código Variável	0	101	100	111	1101	1100

- Número de bits necessário:
  - (45*1 + 13*3 + 12*3 + 16*3 + 9*4 + 5*4)*1000 = 224000 bits
- Códigos livres de prefixo:
  - Nenhum código é prefixo de outro código
  - 001011101 $\rightarrow$ 0.0.101.1101
  - Código óptimo representado por árvore binária (completa)

Códigos de Huffman

Códigos de Huffman

- Dada uma árvore T associada a um código livre de prefixo
  - $f(c)$: frequência (ocorrências) do caracter c no ficheiro
  - $d_T(c)$: profundidade da folha c na árvore
  - $B(T)$: número de bits necessários para representar ficheiro

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

- Código de Huffman: construir árvore T que corresponde ao código livre de prefixo óptimo
  - Começar com $|C|$ folhas (para cada um dos caracteres do ficheiro) e realizar $|C| - 1$ operações de junção para obter árvore final

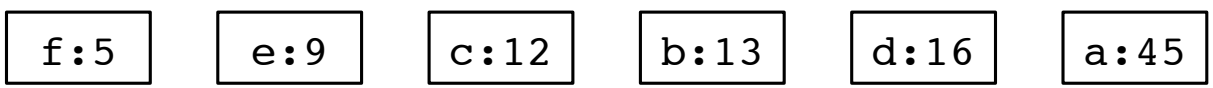
Códigos de Huffman

Huffman(C)

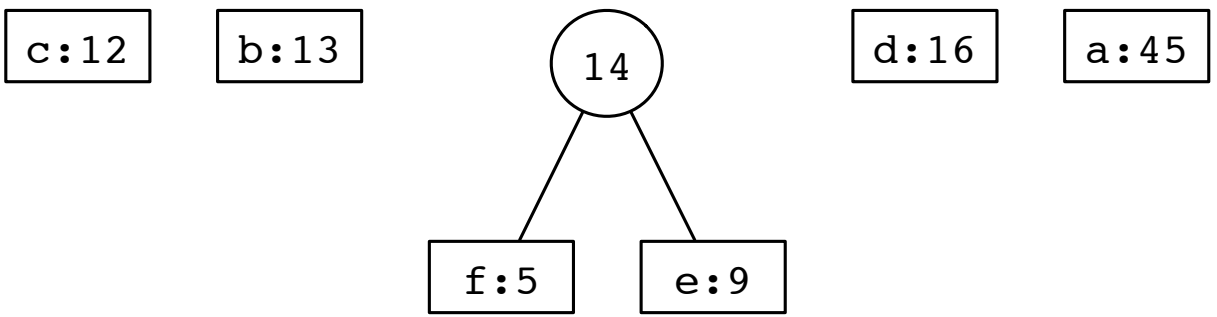
```
1  n ← |C|
2  Q ← C                          ▷ Constrói fila de prioridade
3  for i ← 1 to n − 1
4       do z ← AllocateNode()
5          x ← left[z] ← ExtractMin(Q)
6          y ← right[z] ← ExtractMin(Q)
7          f[z] ← f[x] + f[y]
8          Insert(Q, z)
9  return ExtractMin(Q)
```

Tempo de Execução: $O(n \lg n)$

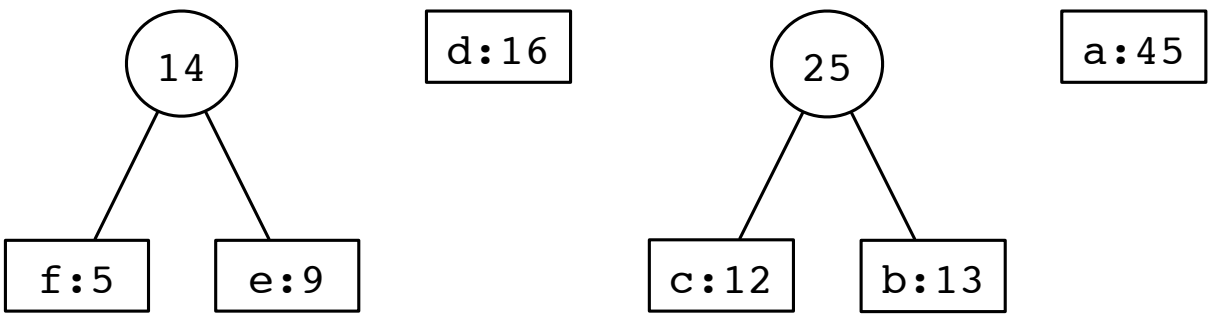
Códigos de Huffman



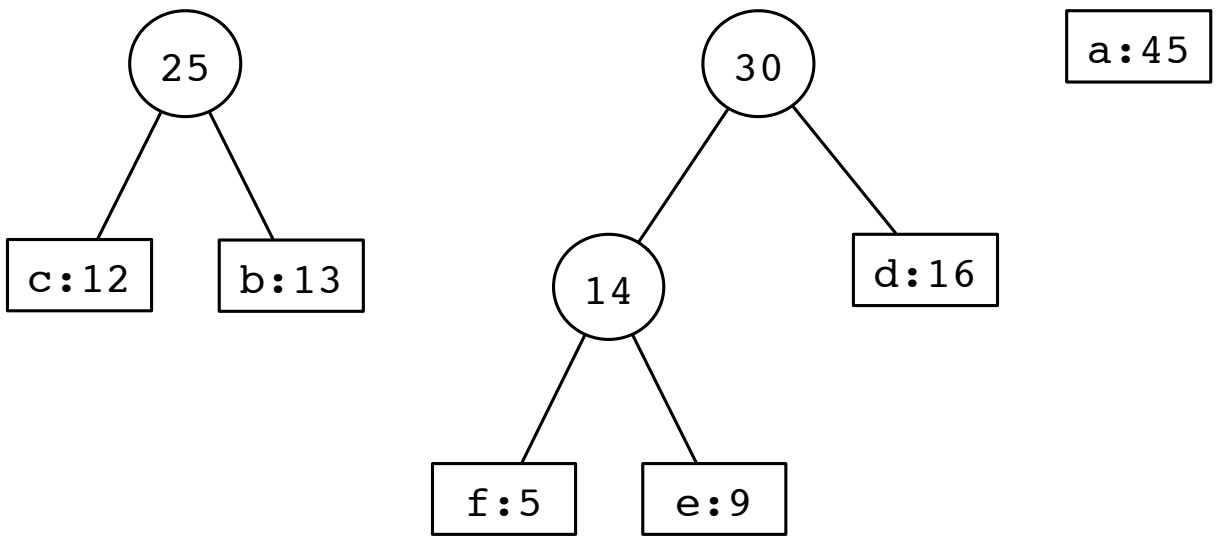
Códigos de Huffman



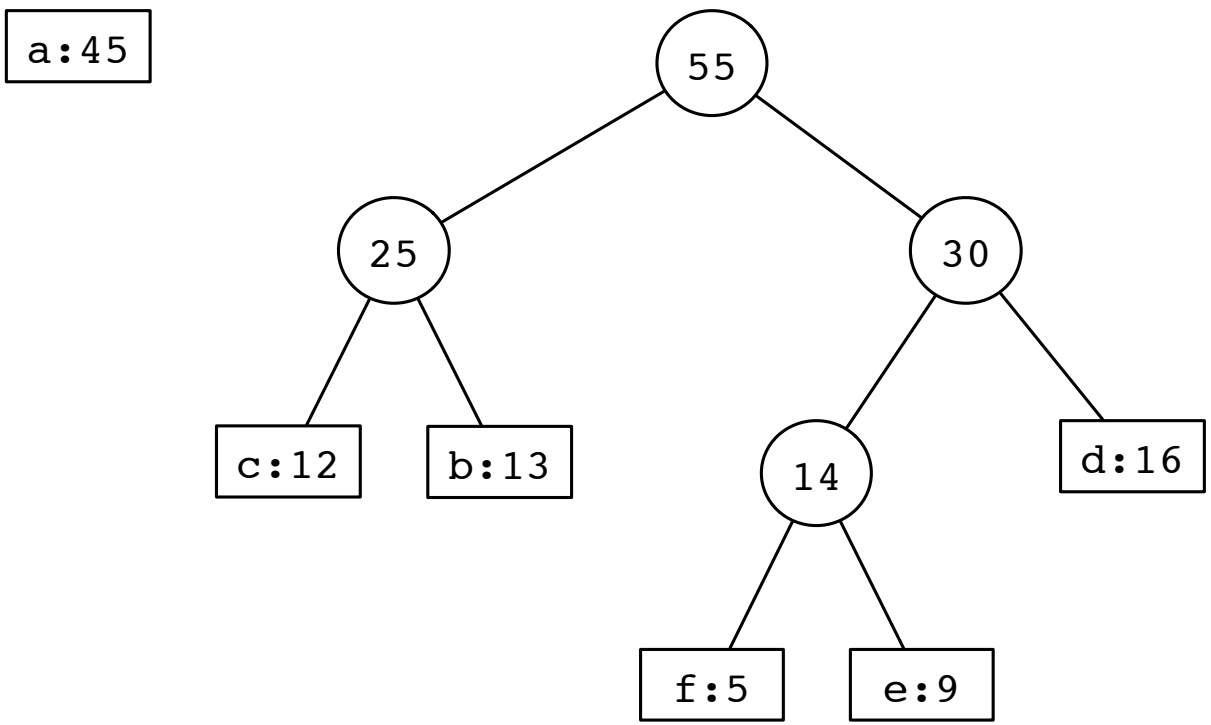
Códigos de Huffman



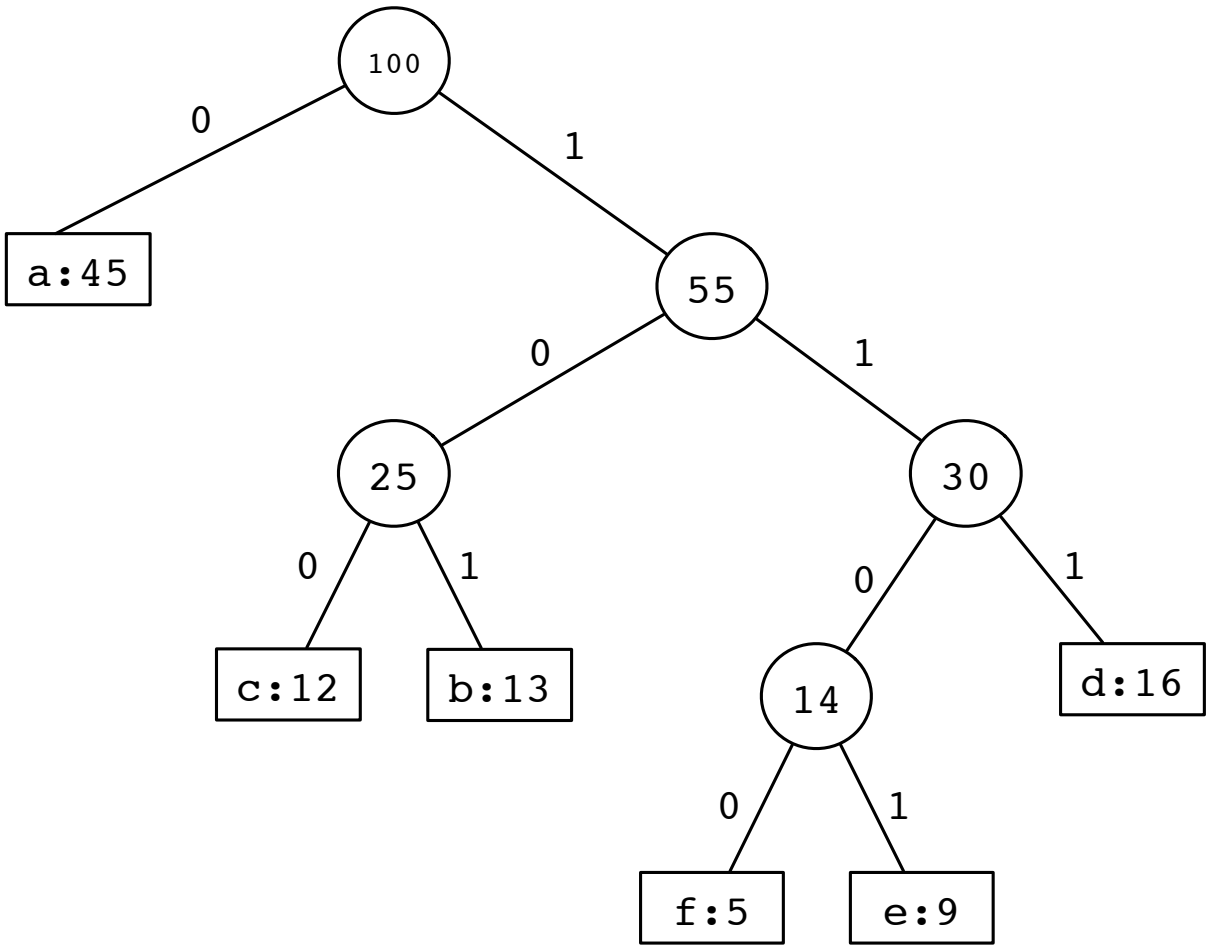
Códigos de Huffman



Códigos de Huffman



Códigos de Huffman



Códigos de Huffman

- Código de Huffman para alfabeto C:
  - Cada caracter $c \in C$ com frequência $f[c]$
  - x, y caracteres em C com as menores frequências
  - Código óptimo para C representado por árvore binária T

Códigos de Huffman

Optimalidade da Solução Greedy

Facto 1: (propriedade da escolha greedy)

- Existe código livre de prefixo para C tal que os códigos para x e y (com as menores frequências) têm o mesmo comprimento e diferem apenas no último bit
  - T árvore que representa código óptimo
  - b e c caracteres são nós folha de maior profundidade em T
  - Admitir, $f[b] \leq f[c]$, e $f[x] \leq f[y]$
  - Notar também que, $f[x] \leq f[b]$, e $f[y] \leq f[c]$
  - T': trocar posições de b e x em T
  - T'': trocar posições de c e y em T'
  - Neste caso, $B(T) \geq B(T') \wedge B(T') \geq B(T'')$
  - Mas, T é óptimo, pelo que $B(T'), B(T'') \geq B(T)$
  - Logo, T'' também é uma árvore óptima !

Códigos de Huffman

Optimalidade da Solução Greedy

Facto 2: (sub-estrutura óptima)

- Sendo z um nó interno de T, e x e y nós folha
- Considerar um caracter z com $f[z] = f[x] + f[y]$
- Então $T' = T - \{x, y\}$ é óptima para $C' = C - \{x, y\} \cup \{z\}$
  - $B(T) = B(T') + f[x] + f[y]$
  - Se T' é não óptimo, então existe T'' tal que $B[T''] < B[T']$
  - Mas z é nó folha também em T'' (devido a facto 1)
    - Adicionando x e y como filhos de z em T''
    - Código livre de prefixo para C com custo:
    - $B[T''] + f[x] + f[y] < B[T]$
    - mas T é óptimo ($B[T''] + f[x] + f[y] \geq B[T]$); pelo que T' também é óptimo

Códigos de Huffman

Optimalidade da Solução Greedy

- O algoritmo Huffman produz um código livre de prefixo óptimo
- Ver factos anteriores
  - Propriedade da escolha greedy
  - Sub-estrutura óptima