Análise e Síntese de Algoritmos

Algoritmos Greedy

CLRS Cap. 16

Prof. Pedro T. Monteiro

IST - Universidade de Lisboa

2024/2025

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica [CLRS, Cap.15]
  - Algoritmos greedy [CLRS, Cap.16]
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Caminhos mais curtos [CLRS, Cap.22,24-25]
  - Árvores abrangentes [CLRS, Cap.23]
  - Fluxos máximos [CLRS, Cap.26]
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais
  - Emparelhamento de Cadeias de Caracteres [CLRS, Cap.32]
  - Complexidade Computacional [CLRS, Cap.34]

Resumo

Estratégia Greedy

Técnicas para Síntese de Algoritmos

- Dividir para conquistar
  - Exemplo: MergeSort
- Programação dinâmica
  - Exemplo: Floyd-Warshall
- Algoritmos greedy
  - Exemplo: Prim, Dijkstra

Estratégia Greedy

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?

Estratégia Greedy

Características 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

Ex 1 - Selecção de Actividades

Definição

- 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
- Cada 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

Ex 1 - Selecção de Actividades

Selecção de Actividades

- Admitir ordenação das actividades por tempos de fim
  $f_1 \leq f_2 \leq \ldots \leq f_n$
- Qual a escolha greedy?
  - Escolher (a cada passo) a actividade com o menor tempo de fim
- Porquê?
  - Maximizar espaço disponível para restantes actividades serem realizadas

Ex 1 - Selecção de Actividades

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	9	10	11	12	14	16

Actividades compatíveis

- $\{a_3, a_9, a_{11}\}$
- $\{a_1, a_4, a_8, a_{11}\}$ (maior subconjunto)
- $\{a_2, a_4, a_9, a_{11}\}$ (maior subconjunto)
- ...

Ex 1 - Selecção de Actividades

s - vector com os tempos de início
f - vector com os tempos de fim (ordenamos as actividades pelos tempos de fim)

Seleccionar-Actividades-Greedy(s, f)

```
n ← length[s]
A ← { 1 }                         (considera-se a actividade 1)
j ← 1            // last selected activity i
for i ← 2 to n do
  if s_i ≥ f_j then
    A ← A ∪ { i }                   (actividade i é compatível)
    j ← i          // update last activity i
  end if
end for
return A
```


Ex 1 - Selecção de Actividades

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: $B = A \setminus \{k\} \cup \{1\}$
  - $f_1 \leq f_k$
    - Actividades em B são mutuamente disjuntas e $|A| = |B|$
    - Logo, B é também solução óptima !

Ex 1 - Selecção de Actividades

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
  - $A' = A \setminus \{1\}$ 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 !

Ex 2 - Problema da Mochila Fraccionário

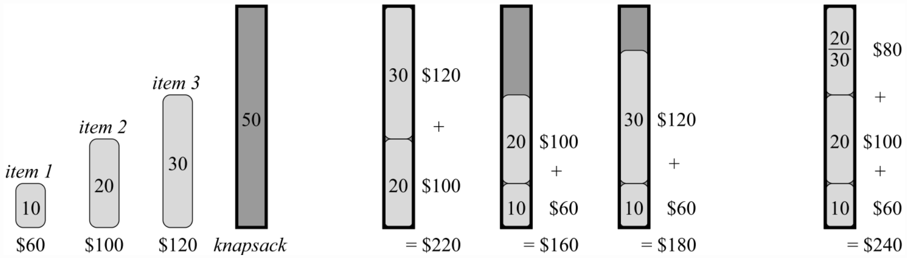
Definição

- 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

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

Ex 2 - Problema da Mochila Fraccionário



Items

Não Fraccionário

Fraccionário

Ex 2 - Problema da Mochila Fraccionário

Observações

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

Ex 2 - Problema da Mochila Fraccionário

Mochila-Fraccionario-Greedy(v[1..n], w[1..n], W)

```
weight ← 0
x ← [1..n]        // init 0
while weight < W do
  escolher proximo objecto i com v_i/w_i máximo
  if w_i + weight ≤ W then
    x_i ← 1                         // levar objecto i inteiro
    weight ← weight + w_i
  else
    x_i ← (W - weight)/w_i          // levar fracção do objecto i
    weight ← W
  end if
end while
return x
```


Ex 2 - Problema da Mochila Fraccionário

Optimalidade da Solução Greedy (1/3)

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

- Admitir ordenação $v_1/w_1 \geq \ldots \geq v_n/w_n$
- Solução calculada por algoritmo greedy: $X = (x_1, \ldots, x_n)$
  - Se $x_i = 1$ (fracção do objecto i) 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: $\sum_{i=1}^{n} x_i v_i = V(X)$

Ex 2 - Problema da Mochila Fraccionário

Optimalidade da Solução Greedy (2/3)

- 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:
  - $i < j \quad \Rightarrow \quad x_i = 1 \quad \wedge \quad x_i - y_i \geq 0 \quad \wedge \quad v_i / w_i \geq v_j / w_j$
  - $i = j \quad \Rightarrow \quad v_i / w_i = v_j / w_j$
  - $i > j \quad \Rightarrow \quad x_i = 0 \quad \wedge \quad x_i - y_i \leq 0 \quad \wedge \quad v_i / w_i \leq v_j / w_j$
- Verifica-se sempre que: $(x_i - y_i)(v_i / w_i) \geq (x_i - y_i)(v_j / w_j)$

Ex 2 - Problema da Mochila Fraccionário

Optimalidade da Solução Greedy (3/3)

- 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 !

"Experienced developer deploys hotfix on production"

Francisco Goya

Oil on canvas

circa 1788

https://classicprogrammerpaintings.com/

Ex 3 - Códigos de Huffman

Definição

Estratégia para construir uma representação compacta da string de caracteres, tendo em conta a frequência de cada caracter

Aplicação: Compressão de Dados

- Exemplo: Ficheiro com 100.000 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: $3 \times 100.000 = 300.000$ bits
- Código de largura variável pode ser melhor do que de largura fixa
  - Aos caracteres mais frequentes associar códigos de menor dimensão

Ex 3 - Códigos de Huffman

Aplicação: Compressão de Dados

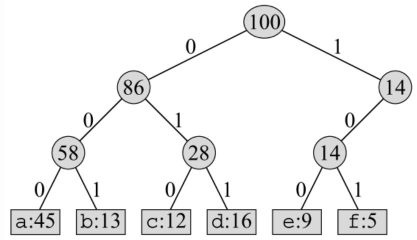
- Código de comprimento variável:

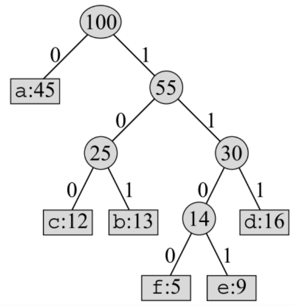
	a	b	c	d	e	f
Frequência (×1000)	45	13	12	16	9	5
Código Variável	0	101	100	111	1101	1100

- Número de bits necessário:
  - (45*1 + 13*3 + 12*3 + 16*3 + 9*4 + 5*4)*1000 = 224.000 bits
- Códigos livres de prefixo:
  - Nenhum código é prefixo de outro código (facilita descompressão)
    001011101 → 0.0.101.1101 → "aabe"
  - Código óptimo é representado por árvore binária

Ex 3 - Códigos de Huffman

Árvore binária





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

	a	b	c	d	e	f
Frequência (×1000)	45	13	12	16	9	5
Código Variável	0	101	100	111	1101	110

Ex 3 - 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

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

- **Problema**: 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

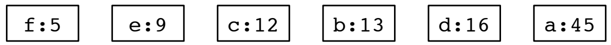
Ex 3 - Códigos de Huffman

```
Huffman(C)
  n ← |C|
  Q ← C
  for i ← 1 to n - 1 do
    z ← AllocateNode()
    x ← left[z] ← ExtractMin(Q)
    y ← right[z] ← ExtractMin(Q)
    f[z] ← f[x] + f[y]
    Insert(Q, z)
  end for
  return ExtractMin(Q)
```

Complexidade: $O(n \log n)$

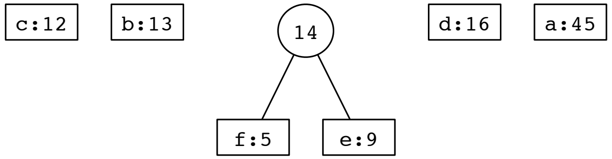
Ex 3 - Códigos de Huffman

Exemplo



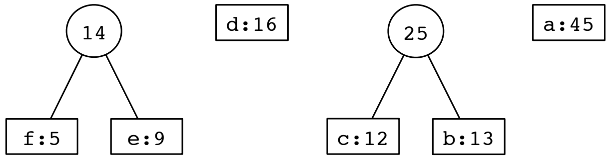
Ex 3 - Códigos de Huffman

Exemplo



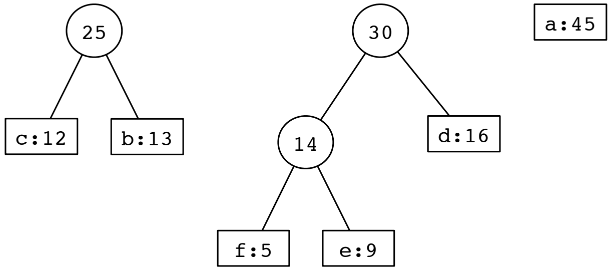
Ex 3 - Códigos de Huffman

Exemplo

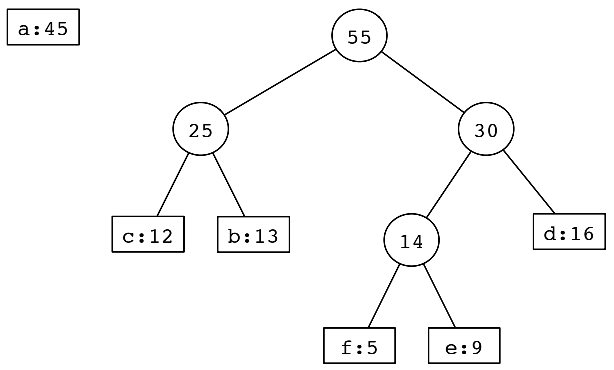


Ex 3 - Códigos de Huffman

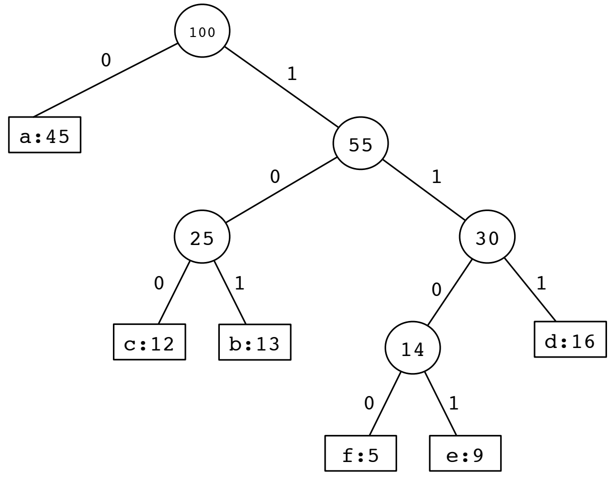
Exemplo



Exemplo



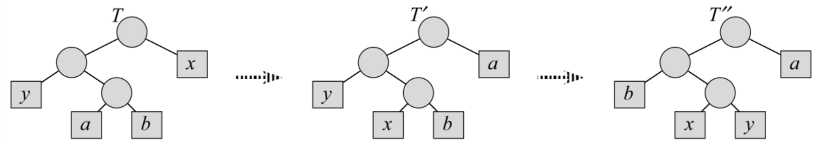
Exemplo



Optimalidade da Solução Greedy

Propriedade da escolha greedy

Lema: Existe código livre de prefixo para C tal que os códigos para caracteres x e y (com as menores frequências) têm o mesmo comprimento e diferem apenas no último bit



Prova (propriedade da escolha greedy)

- T árvore que representa código óptimo
- Caracteres a e b são nós folha de maior profundidade em T
- Admitir, $f[a] \leq f[b]$, e $f[x] \leq f[y]$
- Notar também que, $f[x] \leq f[a]$, e $f[y] \leq f[b]$
- T': trocar posições de a e x em T
- T'': trocar posições de b e y em T'
- Neste caso, $B(T) \geq B(T')$ e $B(T') \geq B(T'')$
- Mas, T é óptimo, então $B(T) \leq B(T'), B(T'')$
- Logo, T'' também é uma árvore óptima !

Optimalidade da Solução Greedy

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 $T' = T \setminus \{x, y\}$ é óptima para $C' = C \setminus \{x, y\} \cup \{z\}$
  - $B(T) = B(T') + f[x] + f[y]$
  - Se T' é não óptima, então existe T'' tal que $B[T''] < B[T']$
  - Mas z é nó folha também em T'' (devido propriedade anterior)
    - 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

Optimalidade da Solução Greedy

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

Exercício

ER 22/23 (casa / quadro)
Considere o problema de compressão de dados de um ficheiro usando a codificação de Huffman. Indique o código livre de prefixo óptimo para cada carácter num ficheiro com 300 caracteres com a seguinte frequência de ocorrências (dada em percentagem):

$$f(a) = 51, f(b) = 7, f(c) = 8, f(d) = 10, f(e) = 24$$

Quando constrói a árvore, atribua o bit 1 para o nó com menor frequência. Indique também o número total de bits no ficheiro codificado.

Questões?

Dúvidas?