Análise e Síntese de Algoritmos

Algoritmos Greedy

CLRS Cap. 16

Instituto Superior Técnico

2022/2023

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]
  - Fluxos máximos [CLRS, Cap.26]
  - Árvores abrangentes [CLRS, Cap.23]
- 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 ótima

- Estratégia permite obter solução ótima?

Estratégia Greedy

Caraterísticas Algoritmos Greedy

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

Seleção de Atividades

Definição

- Seja $S = \{1, 2, \ldots, n\}$ um conjunto de atividades que pretendem utilizar um dado recurso
- Apenas uma atividade pode utilizar o recurso de cada vez
- Cada atividade i:
  - tempo de início: s_i
  - tempo de fim: f_i
  - execução da atividade durante $[s_i, f_i[$
- Atividades i e j compatíveis apenas se $[s_i, f_i[$ e $[s_j, f_j[$ não se intersetam – atividades cujos intervalos não se sobrepõem

Objectivo: encontrar conjunto máximo de atividades mutuamente compatíveis

Seleção de Atividades

Seleção de Atividades

- Admitir ordenação $f_1 \leq f_2 \leq \ldots \leq f_n$
- Qual a escolha greedy?
  - Escolher atividade com o menor tempo de fim
- Porquê?
  - Maximizar espaço para restantes atividades serem realizadas

Seleção de Atividades

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

Atividades 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)
- ...

s - vector com os tempos de início
f - vector com os tempos de fim (ordenado)

Selecionar-Atividades-Greedy(s, f)

```
n ← length[s]
A ← {1}
j ← 1              // last selected activity i
for i ← 2 to n do
   if s_i ≥ f_j then
      A ← A ∪ {i}
      j ← i        // update last activity i
   end if
end for
return A
```


Otimalidade da Solução Greedy

- Algoritmo encontra soluções de tamanho máximo para o problema de Seleção de Atividades
- Existe uma solução ótima que começa com escolha greedy, i.e. atividade 1
  - Seja A uma solução ótima que começa na atividade k
  - Seja $B = A \setminus \{k\} \cup \{1\}$ uma solução que começa na atividade 1
  - Como as atividades estão ordenadas por ordem crescente de f, então $f_1 \leq f_k$
    - Se a atividade k é compatível com as atividades em $A \setminus \{k\}$, então a atividade 1 também é compatível com as atividades em $A \setminus \{k\}$, porque $f_1 \leq f_k$
    - Atividades em B são mutuamente disjuntas e $|A| = |B|$
    - Logo, B é também solução ótima!

Otimalidade da Solução Greedy

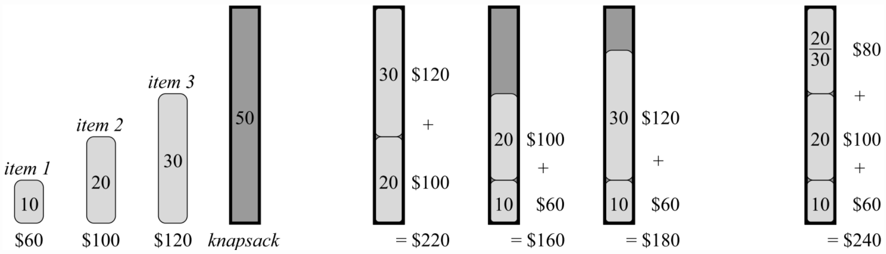
- Algoritmo encontra soluções de tamanho máximo para o problema de Seleção de Atividades
- Após escolha greedy, problema reduz-se a encontrar solução para atividades compatíveis com atividade 1
  - Seja A uma solução ótima que começa na atividade 1
  - $A' = A \setminus \{1\}$ é uma solução ótima para (o sub-problema com atividades em) $S' = \{i \in S : s_i \geq f_1\}$
  - Caso contrário, existiria uma solução B', tal que $|B'| > |A'|$, para S', que permitiria obter uma solução B para S com mais atividades do que A; A não seria ótima, o que é uma contradição !
- Aplicar indução no número de escolhas greedy
- Algoritmo calcula solução ótima !

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

Exemplo: Problema da Mochila



Items **Não Fracionário** **Fracionário**

Observações

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

Mochila-Fracionario-Greedy(v[1..n], w[1..n], W)
  weight ← 0
  $x \leftarrow [1..n]$ // init 0
  while *weight* < W **do**
    escolher proximo objecto i com v_i/w_i máximo
    if $w_i + weight \leq W$ **then**
      $x_i \leftarrow 1$; $weight \leftarrow weight + w_i$
    else
      $x_i \leftarrow (W - weight)/w_i$; $weight \leftarrow W$
    end if
  end while
  return x

Otimalidade da Solução Greedy

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

- 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 ótima
  - 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: $\sum_{i=1}^{n} x_i v_i = V(X)$

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

Problema da Mochila Fracionário

Otimalidade 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 ótima !

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

Códigos de Huffman

Aplicação: Compressão de Dados

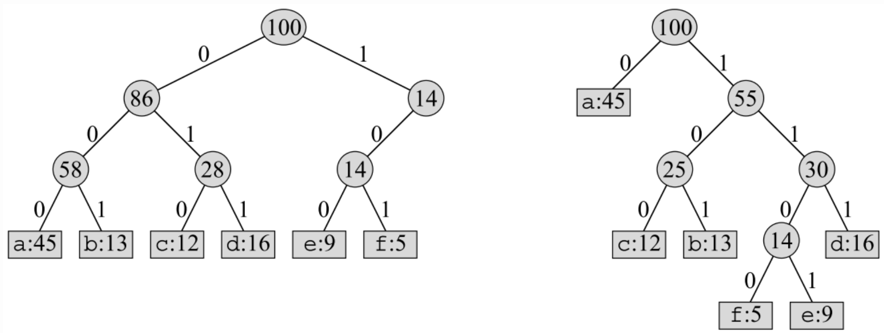
- Código de comprimento variável:

	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 = 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

Códigos de Huffman

Árvore binária



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

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

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

Códigos de Huffman

Huffman(C)
  $n \leftarrow |C|$
  $Q \leftarrow C$
  for $i \leftarrow 1$ **to** $n - 1$ **do**
    $z \leftarrow \text{AllocateNode}()$
    $x \leftarrow \mathit{left}[z] \leftarrow \text{ExtractMin}(Q)$
    $y \leftarrow \mathit{right}[z] \leftarrow \text{ExtractMin}(Q)$
    $f[z] \leftarrow f[x] + f[y]$
    $\text{Insert}(Q, z)$
  end for
  return $\text{ExtractMin}(Q)$

Complexidade: $O(n \log n)$

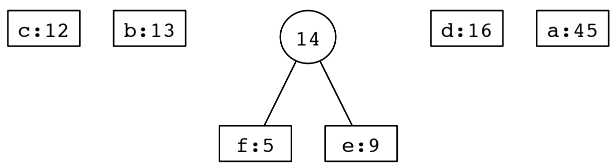
Códigos de Huffman

Exemplo

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

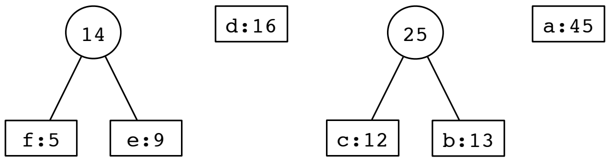
Códigos de Huffman

Exemplo



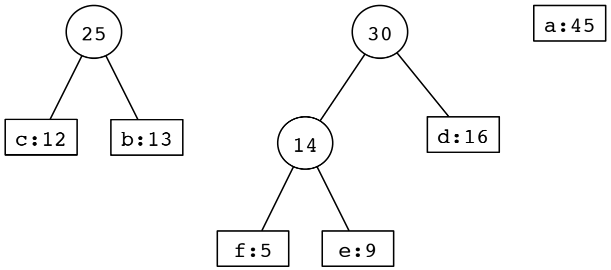
Códigos de Huffman

Exemplo



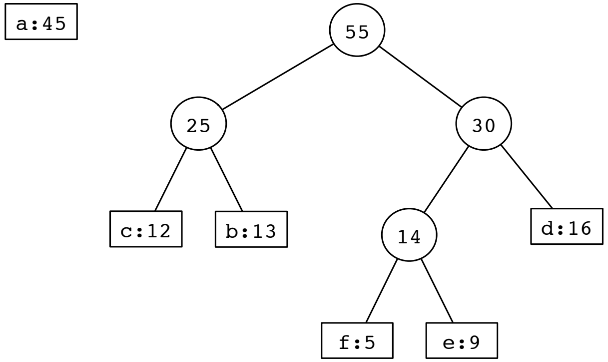
Códigos de Huffman

Exemplo



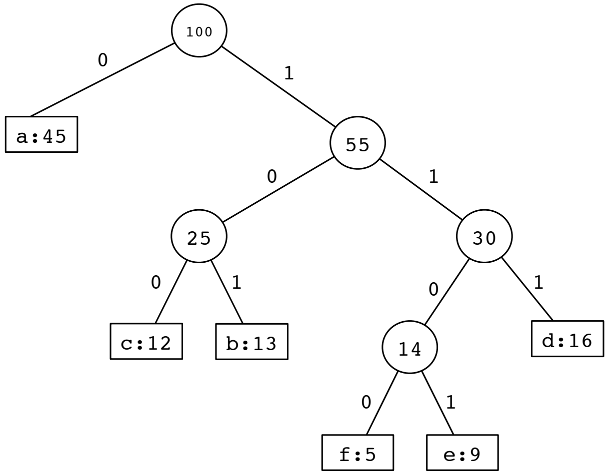
Códigos de Huffman

Exemplo



Códigos de Huffman

Exemplo

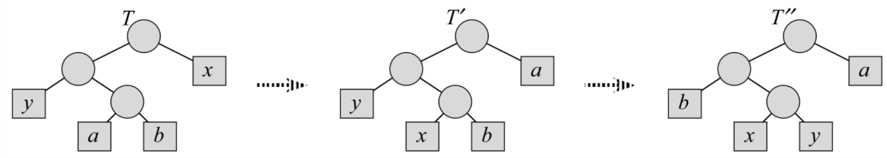


Otimalidade da Solução Greedy

Propriedade da escolha greedy

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

- Corresponde a dizer que existe uma árvore que representa um código ótimo em que x e y são os dois filhos do mesmo nó pai
- Vamos construir uma árvore T'' em que isto se verifica e provar que esta representa um código ótimo



Prova (propriedade da escolha greedy)

- T árvore que representa um código ótimo arbitrário
- Caracteres a e b são nós folha de maior profundidade em T
- Admitir, $f[a] \leq f[b]$, e $f[x] \leq f[y]$
- Se x e y têm as menores frequências: $f[x] \leq f[a]$, e $f[y] \leq f[b]$
- T': trocar posições de a e x em T
- T'': trocar posições de b e y em T'
- Assim, $B(T) \geq B(T')$ e $B(T') \geq B(T'')$, logo $B(T) \geq B(T'')$
- Mas, T é óptima, então $B(T) \leq B(T')$ e $B(T) \leq B(T'')$
- Então, $B(T'') = B(T)$, portanto T'' também é uma árvore ótima !

Otimalidade da Solução Greedy

Sub-estrutura ótima

- Sejam x e y os caracteres com menores frequências em C
- Seja $C' = C \setminus \{x, y\} \cup \{z\}$ o alfabeto onde x e y foram substituídos por z, tal que $f[z] = f[x] + f[y]$
- Admitir, T' representa um código ótimo para C'
- T é obtida a partir de T' substituindo o nó folha z por um nó interno que tem x e y como filhos
- Então, T representa um código ótimo para C

Otimalidade da Solução Greedy

Sub-estrutura ótima

- ...
- Então, T representa um código ótimo para C
  - $B(T) = B(T') + f[x] + f[y]$, porque a representação de x e y tem mais 1 *bit* do que a de z
  - Se T não é ótima, então existe um árvore ótima T'', tal que $B(T'') < B(T)$
  - Em T'', x e y são filhos do mesmo pai, devido à propriedade de escolha greedy
    - T''' obtida a partir de T'' substituindo o nó interno que tem x e y como filhos por um nó folha z, tal que $f[z] = f[x] + f[y]$
    - $B(T''') = B(T'') - f[x] - f[y] < B(T) - f[x] - f[z] = B(T')$
    - $B(T''') < B(T')$ contradiz a premissa de que T' é ótima para C'!

Códigos de Huffman

Otimalidade da Solução Greedy

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