Análise e Síntese de Algoritmos

Programação dinâmica

CLRS Cap. 15

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

Programação dinâmica

Problema da Mochila s/rep

Problema da Mochila c/rep

Técnicas para Síntese de Algoritmos

Técnicas para Síntese de Algoritmos

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

Exemplo

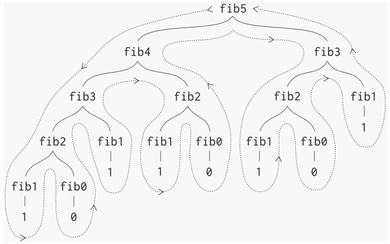
Calcular o n-ésimo elemento da sequência de Fibonacci, sabendo que:

$$fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ fib(n-1) + fib(n-2) & \text{caso contrário} \end{cases}$$

Solução Recursiva

fib(n)

if n $\leq$ 1 **then**
  return n
else
  return fib(n-1) + fib(n-2)
end if



Complexidade: $O(2^n)$

Solução Programação Dinâmica

fib(n)

if n $\leq$ 1 **then**
  return n
else
  Alocar vector f com $n+1$ posições
  f[0] = 0
  f[1] = 1
  i = 2
  while i $\leq$ n **do**
    f[i] = f[i-1]+f[i-2]
    i = i+1
  end while
  return f[n]
end if

Complexidade: $\Theta(n)$

Exemplo

Calcular as combinações de n, k a k

$$C^n_k = \begin{cases} 1 & \text{se } k = 0 \text{ ou } k = n \\ C^{n-1}_{k-1} + C^{n-1}_k & \text{se } 0 < k < n \\ 0 & \text{caso contrário} \end{cases}$$

Solução Recursiva

C(n,k)

if k = 0 or k = n **then**
  return 1
else
  return C(n-1, k-1) + C(n-1, k)
end if

- Cada chamada a $C(n, k)$ retorna 1 ou invoca o cálculo de dois sub-problemas
- Solução é calculada somando 1's !
- Tempo de execução é: $\Omega(C^n_k)$

Programação Dinâmica

Exemplo

- Número de $C(n, k)$ distintos é apenas $n \times k$
- Complexidade da solução recursiva deriva do cálculo repetido de sub-problemas
  - C(5,3) = C(4,2) + C(4,3)
  - C(4,2) = C(3,1) + C(3,2)
  - C(4,3) = C(3,2) + C(3,3)
- Solução: solução construtiva (bottom-up)
  - Preencher tabela $(n \times k)$ (triângulo de Pascal)

Programação Dinâmica

Características da Programação Dinâmica

- Solução óptima do problema composta por soluções óptimas para sub-problemas
- Solução recursiva resolve repetidamente os mesmos sub-problemas
  - Sobreposição de problemas

Tabulação vs Memoization

- Tabulação: preencher tabela de soluções bottom-up (forma iterativa)
- Memoization: guardar soluções calculadas top-down (forma recursiva)

Programação Dinâmica

Programação Dinâmica

Passos para a realização de um algoritmo baseado em programação dinâmica:

- Caracterizar estrutura de uma solução óptima
- Definir recursivamente o valor de uma solução óptima
- Calcular valor da solução óptima utilizando abordagem bottom-up
- Construir solução a partir do resultado de sub-problemas usando tabulação
  para evitar resolver repetidamente o mesmo (sub-)problema

Ex 1 - Problema da Mochila s/rep

Definição (sem repetição)

- 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
- **Objectivo:** maximizar o valor transportado pela mochila e respeitar a restrição de peso

Formalização

- $x_i = 1$ se objecto i incluído na mochila; 0 caso contrário

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

Tentativa de solução

- Algoritmo que a cada passo selecciona objecto com maior valor v_i/w_i
- Problema:
  - $v_1 = 8, w_1 = 6$
  - $v_2 = 5, w_2 = 5$
  - $v_3 = 5, w_3 = 5$
  - $W = 10$
- Primeiro objecto seleccionado (objecto 1) impede encontrar solução óptima (com objectos 2 e 3)

Solução

- $v[i,j]$: máximo valor que é possível transportar se:
  - se o peso limite é j, ($j \leq W$)
  - e se apenas podem ser seleccionados os objectos numerados de 1 a i
- Solução óptima encontra-se em $v[n, W]$
- Definição:

$$v[i,j] = max(v[i-1,j], v[i-1,j-w_i] + v_i)$$

$$\begin{aligned} v[0,j] &= 0, \quad j \geq 0 \\ v[i,j] &= -\infty, \ j < 0 \\ v[i,0] &= 0 \end{aligned}$$

Análise

- Solução óptima é composta por soluções óptimas para os sub-problemas:

$$v[i,j] = max(v[i-1,j], v[i-1,j-w_i] + v_i)$$

- Se $v[i,j]$ é solução óptima:
  - Se objecto i não é incluído, $v[i-1,j]$ é sub-solução óptima
  - Se objecto i é incluído, $v[i-1,j-w_i]$ é sub-solução óptima
- Caso contrário:
  seria possível obter solução com valor superior ao da solução óptima; uma contradição !

Análise

- Solução recursiva tem tempo de execução exponencial em n e W
  - Mas: número de sub-problemas distintos é apenas $n \times W$
  - **Conclusão:** resolução repetida dos mesmos sub-problemas !
- Utilizar solução construtiva (bottom-up)
  - preencher tabela ($n \times W$) da esquerda/cima para a direita/baixo

Complexidade: $\Theta(nW)$

- pseudo-polinomial (depende do valor, não do tamanho do input)

Ex 2 - Problema da Mochila c/rep

Definição (com repetição)

- 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
- **Objectivo:** maximizar o valor transportado pela mochila e respeitar a restrição de peso

Formalização

- $x_i = k$ se objecto incluído k vezes na mochila; 0 caso contrário

$$\begin{array}{ll} \text{maximizar} & \sum_{i=1}^{n} v_i x_i \\ \text{sujeito a} & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \mathbb{N} \end{array}$$

Ex 2 - Problema da Mochila c/rep

Solução

- $v[j]$: máximo valor que é possível transportar se:
  - se o peso limite é j, $(j \leq W)$
  - e se apenas podem ser seleccionados os objectos numerados de 1 a i
- Solução óptima encontra-se em $v[W]$
- Definição:

$$v[j] = max_{1 \leq i \leq n}\{v[j - w_i] + v_i\}, \quad j \leq W$$

$$v[0] = 0$$
$$v[j] = -\infty, \quad j < 0$$

Problema da Mochila s/ rep

Exercício: (casa / quadro)

Qual o valor máximo alcançado com uma mochila de capacidade W=10, para os seguintes objectos:

Obj 1	$w_1 = 2$	$v_1 = 3$
Obj 2	$w_2 = 3$	$v_2 = 4$
Obj 3	$w_3 = 4$	$v_3 = 5$
Obj 4	$w_4 = 5$	$v_4 = 6$

$$v[i,j] = max(v[i-1,j], v[i-1, j-w_i] + v_i)$$

Questões?

Dúvidas?