Aula 28: Programação dinâmica

David Déharbe
Programa de Pós-graduação em Sistemas e Computação
Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada

Download me from `http://DavidDeharbe.github.io`

Plano

Introdução

Princípios gerais

Exemplo introdutório

Exemplo intermediário

Exemplo avançado

Exercícios

Referência: Cormen, cap 16.

Introdução

- qual forma ótima de escolher N objetos, cada um com peso $w_1, \ldots, w_n$, e valores $v_1, \ldots v_n$, para levar em uma mala de até K kilogramas?
- qual a maior subsequência comum a duas sequências?
- qual a quantidade mínima de edições (adição/remoção/troca) para passar de uma palavra à outra?
- qual a ordem ótima para multiplicar N matrizes $A_1, \ldots A_N$?
- como escolher valores dentre de um conjunto de tal forma que a soma deles seja nula?

PPgSC

Introdução

- qual forma ótima de escolher N objetos, cada um com peso $w_1, \ldots, w_n$, e valores $v_1, \ldots v_n$, para levar em uma mala de até K kilogramas?
- qual a maior subsequência comum a duas sequências?
- qual a quantidade mínima de edições (adição/remoção/troca) para passar de uma palavra à outra?
- qual a ordem ótima para multiplicar N matrizes $A_1, \ldots A_N$?
- como escolher valores dentre de um conjunto de tal forma que a soma deles seja nula?

programação dinâmica!

PPgSC

Origem

- 1940–1950 Richard Bellman
  - o mesmo do algoritmo de Bellman-Ford
- contexto: resolução de problemas envolvendo tomadas de decisões sucessivas
- *dynamic programming*: construção dinâmica de uma programação.

PPgSC

Princípios

- decomposição: a solução para uma instância I pode ser expressa a partir das soluções para instâncias menores $I_1, \ldots I_n$ derivadas de I
- aplicar sucessivamente esta decomposição leva a ter que solucionar várias vezes o mesmo sub-problema. sub-problemas sobrepostos
- a melhor solução para I pode ser obtida a partir das melhores soluções para $I_1, \ldots I_n$. sub-estrutura ótima

PPgSC

Exemplo 1

$$
\begin{aligned}
fib(0) &= 0 \\
fib(1) &= 1 \\
fib(n) &= fib(n-1) + fib(n-2) \text{para } n \geq 2
\end{aligned}
$$

1. sub-estrutura ótima: não há otimização
2. sub-problemas sobrepostos: não calcular duas vezes o mesmo resultado

- recursão + memorização das soluções dos sub-problemas

Exemplo 1

Resolução computational

- abordagem ascendente
  - iniciar dos sub-problemas mais simples possíveis
  - calcular sub-problemas cada vez maiores
  - $fib(0), fib(1), fib(2), \ldots, fib(n)$.
- abordagem descendente
  - quebrar o problema em sub-problemas mais simples
  - solucionar os sub-problemas gerados
  - $fib(n), fib(n-1), fib(n-2), fib(n-3) \ldots, fib(0)$.
  - memorizar as soluções já calculadas

PPgSC

Exercício

- Escrever um algoritmo para calcular $fib(n)$ utilizando a abordagem ascendente
- Escrever um algoritmo para calcular $fib(n)$ utilizando a abordagem descendente

PPgSC

Exemplo: um problema de tabuleiro

- É dado um tabuleiro $N \times M$ tal que há um custo $C_{i,j}$ associado à sub-divisão da linha i e da coluna j.
- Projetar um algoritmo que determina o menor custo para atravessar o tabuleiro da primeira até a última linha?

PPgSC

Análise do problema

- Se $M = 1$, há um único caminho possível
- Se $M \geq 2$, quantos caminhos são possíveis?

1	1	1	1	1	1	1	1	1
2	3	3	3	3	3	3	3	2
5	8	9	9	9	9	9	8	2
13	22	26	27	27	27	26	22	13

- para cada sub-divisão: $\Omega(2^{N-1})$ e $O(3^{N-1})$ caminhos
- total: $\Omega(M \cdot 2^{N-1})$ e $O(M \cdot 3^{N-1})$ caminhos
- exponencial
- muitos caminhos possuem trechos comuns
  - sub-problemas sobrepostos

PPgSC

Análise do problema

- Como calcular o menor custo para uma sub-divisão?
- $P_{i,j}$: custo da sub-divisão (i, j)
- $C_{i,j}$: menor custo para chegar à divisão (i, j)
- $i > 1$, sub-divisão fora da borda:

$C_{i-1,j-1}$	$C_{i-1,j}$	$C_{i-1,j+1}$
	$P_{i,j}$	

- $C_{i,j} = P_{i,j} + \min\{C_{i-1,j-1}, C_{i-1,j}, C_{i-1,j+1}\}$
- $i > 1$, sub-divisão na borda:

$C_{i-1,1}$	$C_{i-1,2}$
$P_{i,1}$	

ou

$C_{i-1,M-1}$	$C_{i-1,M}$
	$P_{i,M}$

- $C_{i,1} = P_{i,1} + \min\{C_{i-1,1}, C_{i-1,2}$ ou
  $C_{i,M} = P_{i,M-1} + \min\{C_{i-1,1}, C_{i-1,M}$
  - sub-estrutura ótima

PPgSC

Algoritmo

- Queremos calcular $C[i, j]$ para cada i, j.
- Concluído este cálculo, basta determinar o menor valor entre os $C[N, j]$.

```
1   for j = 1 to M
2       C[1, j] = P[1, j]
3   for i = 2 to N
4       C[i, 1] = P[i, 1] + min{C[i − 1, 1], C[i − 1, 2]}
5       C[i, M] = P[i, M] + min{C[i − 1, M − 1], C[i − 1, M]}
6       for j = 2 to M − 1
7           C[i, j] = P[i, j] + min{C[i − 1, j − 1], C[i − 1, j], C[i − 1, j + 1]}
8   res = C[N, 1]
9   for j = 2 to N
10      if C[N, j] < res
11          res = C[N, j]
12  return res
```

Algoritmo

```
1   for j = 1 to M
2       C[1,j] = P[1,j]
3   for i = 2 to N
4       C[i,1] = P[i,1] + min{C[i-1,1], C[i-1,2]}
5       C[i,M] = P[i,M] + min{C[i-1,M-1], C[i-1,M]}
6       for j = 2 to M-1
7           C[i,j] = P[i,j] + min{C[i-1,j-1], C[i-1,j], C[i-1,j+1]}
8   res = C[N,1]
9   for j = 2 to N
10      if C[N,j] < res
11          res = C[N,j]
12  return res
```

Complexidade?

Algoritmo

```
1   for j = 1 to M // Θ(M)
2       C[1,j] = P[1,j]
3   for i = 2 to N // Θ(N) repetições
4       C[i,1] = P[i,1] + min{C[i-1,1], C[i-1,2]}
5       C[i,M] = P[i,M] + min{C[i-1,M-1], C[i-1,M]}
6       for j = 2 to M-1 // Θ(M) repetições
7           C[i,j] = P[i,j] + min{C[i-1,j-1], C[i-1,j], C[i-1,j+1]}
8   res = C[N,1]
9   for j = 2 to N
10      if C[N,j] < res
11          res = C[N,j]
12  return res
```

Complexidade? $\Theta(NM)$

PPgSC

Exercício

1. Adapte o algoritmo anterior para imprimir o caminho menos custoso para atravessar o tabuleiro.

Exemplo: o problema da multiplicação de matrizes

- É dada uma sequência de matrizes de tamanho $L_1 \times C_1$, $L_2 \times C_2, \ldots L_n \times C_n$, tal que $L_i = C_{i+1}$, para $1 \leq i < n$.
  - exemplo: $A_1 : 10 \times 100, A_2 : 100 \times 5, A_3 : 5 \times 50$
  - $(A_1 \cdot A_2) \cdot A_3$: $10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 7.500$
  - $A_1 \cdot (A_2 \cdot A_3)$: $100 \cdot 5 \cdot 50 + 10 \cdot 100 \cdot 50 = 75.000$
- Projetar um algoritmo que determina a ordem ótima de realizar as multiplicações (aquela que minimiza o número de operações entre elementos de matrizes).
- Notação: A_i tem dimensão $p_{i-1} \times p_i$.

PPgSC

Análise do problema

- Para n matrizes: quantas maneiras existem de calcular o produto delas?
- N matrizes podem ser divididas de $n-1$ formas diferentes: A_1 e $(A_2 \dots A_n)$, (A_1, A_2) e $(A_3 \dots A_n)$, etc.
- Seja $\pi(n)$ o número de maneiras que podemos calcular o produto de N matrizes.

$$
\begin{aligned}
\pi(1) &= 1 \\
\pi(n) &= \pi(1)\cdot\pi(n-1) + \pi(2)\cdot\pi(n-2) + \cdots + \pi(n-1)\cdot\pi(1) \\
&= \sum_{k=1}^{n-1} \pi(k)\cdot\pi(n-k) \\
&= \frac{1}{n}\begin{pmatrix} 2n-2 \\ n-1 \end{pmatrix} \\
&\in \Omega(4^n/n^{3/2})
\end{aligned}
$$

- crescimento exponencial

PPgSC

Análise da estrutura do problema

- Seja $A_1, \ldots A_n$ as matrizes a multiplicar.
- Seja $A_{i..j}$ o resultado da multiplicação de $A_i \ldots A_j$
- Um agrupamento ótimo para $A_1 \ldots A_n$ divide em dois grupos $A_1 \ldots A_k$ e $A_{k+1} \ldots A_n$.
- O custo do agrupamento ótimo é o custo de calcular de forma ótima $A_{1..k}$ e $A_{k+1..n}$, mais o custo de multiplicar essas duas matrizes.
- Um agrupamento ótimo para calcular $A_{1..n}$ deve incluir agrupamentos ótimos para calcular $A_{1..k}$ e $A_{k+1..n}$.
- e sub-estrutura ótima

PPgSC

Análise dos sub-problemas

- sub-problemas: qual o custo mínimo para calcular $A_i \dots A_j$, onde $1 \leq i \leq j \leq n$?
- seja $m[i, j]$ este custo
- queremos calcular $m[1, n]$
- caso de base: $i = j$
  - nenhuma operação: $m[i, i] = 0$, para $1 \leq i \leq n$
- caso geral:
  - se dividir em k: $m[i, j] = m[i, k] + m[k, j] + p_{i-1} p_k p_j$.
  - logo
    $m[i, j] = \min\{m[i, j] = m[i, k] + m[k, j] + p_{i-1} p_k p_j \cdot i \leq k < j\}$
    se $i < j$

Análise dos sub-problemas

- sub-problemas: qual índice k, tal que $i \leq k <$, dividir $A_i \ldots A_j$, em $A_{i..k}$ e $A_{k+1..j}$, onde $1 \leq i \leq j \leq n$?
- seja $s[i, j]$ este índice
- $s[i, j]$ é o índice que minimiza $m[i, k] + m[k, j] + p_{i-1} p_k p_j$.
- há $\Theta(n^2)$ sub-problemas no total
- um cálculo recursivo descendente necessita solucionar repetidas vezes o mesmo problema
- sub-problemas sobrepostos
- abordagem ascendente: cálculo de $m[i, j]$, $s[i, j]$ após ter calculado os sub-problemas contidos

PPgSC

Algoritmo

```
1   n = p.length − 1
2   for i = 1 to n
3         m[i, i] = 0
4   for l = 2 to n
5         for i = 1 to n − l + 1
6               j = i + l − 1
7               m[i, j] = ∞
8               for k = 1 to j − 1
9                     q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
10                    if q < m[i, j]
11                          m[i, j] = q
12                          s[i, j] = k
13  return m and s
```

Ilustração

matriz	dimensão
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

PPgSC

Algoritmo

Complexidade?

```
1   n = p.length − 1
2   for i = 1 to n
3         m[i, i] = 0
4   for l = 2 to n
5         for i = 1 to n − l + 1
6               j = i + l − 1
7               m[i, j] = ∞
8               for k = 1 to j − 1
9                     q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
10                    if q < m[i, j]
11                          m[i, j] = q
12                          s[i, j] = k
13  return m and s
```

Algoritmo

Complexidade?

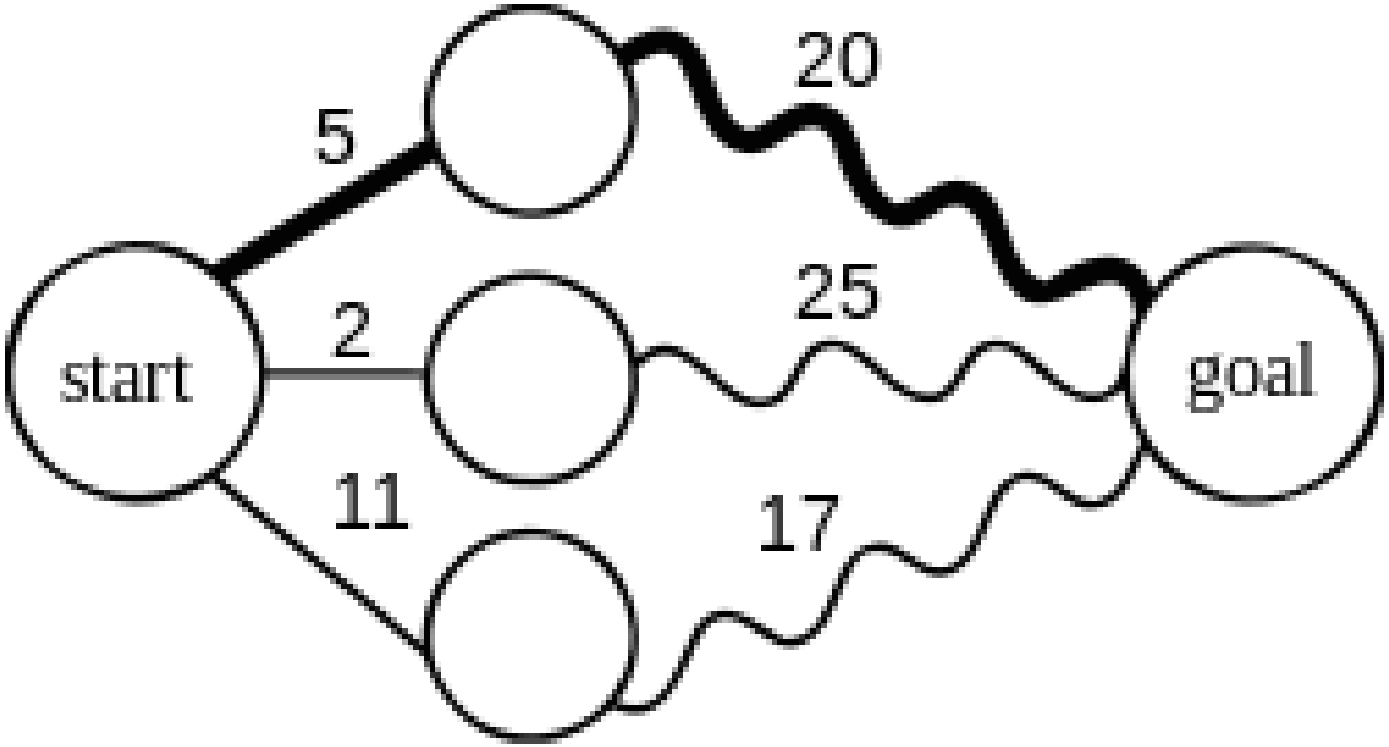
```
 1  n = p.length − 1
 2  for i = 1 to n
 3        m[i, i] = 0
 4  for l = 2 to n
 5        for i = 1 to n − l + 1
 6              j = i + l − 1
 7              m[i, j] = ∞
 8              for k = 1 to j − 1
 9                    q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
10                    if q < m[i, j]
11                          m[i, j] = q
12                          s[i, j] = k
13  return m and s
```

$\Theta(n^3)$

Concluindo: multiplicando matrizes de forma ótima

```
MATRIX-PRODUCT+(A, s, i, j)
1  if j > i
2        X = MATRIX-PRODUCT+(A, s, i, s[i, j])
3        Y = MATRIX-PRODUCT+(A, s, s[i, j] + 1, j)
4        return MATRIX-MULTIPLY(X, Y)
5  else return A_i
```

Menor caminho entre duas arestas



1. sub-estrutura ótima: em um grafo conectado com pesos, a menor distância de u até v pode ser calculada a partir da menor distância dos vértices adjacentes a u até v.
2. sub-problemas sobrepostos: não calcular duas vezes o mesmo resultado

- recursão + memorização das soluções dos sub-problemas

PPgSC

Exercício

1. Escrever um algoritmo para calcular o menor caminho de u até v utilizando a abordagem ascendente
2. Escrever um algoritmo para calcular o menor caminho de u até v utilizando a abordagem descendente

Sub-conjunto de soma nula

- É dada um conjunto de inteiros $\{v_1, v_2, \ldots v_n\}$.
- Projetar um algoritmo que determina se existe um sub-conjunto tal que a soma dos elementos é nula.

PPgSC

Elementos de resposta

- Quantos sub-conjuntos de $S = \{v_1, v_2, \ldots v_n\}$?
- Seja N a soma dos números negativos de S
- Seja P a soma dos números positivos de S
- Seja $Q(i, s)$ verdadeiro sse existe um sub-conjunto de $\{v_1, v_2, \ldots v_i\}$ de soma s.
- Representar Q por uma matriz $N + P + 1 \times n$
- Calcular $Q(n, 0)$ soluciona o problema desejado
- Temos que
  - $Q(1, s) \leftrightarrow v_1 = s$
  - para $i > 1$:

$$
\begin{array}{lll}
Q(i,s) \Leftrightarrow & Q(i-1,s) & v_i \text{ não é somado} \\
\vee & Q(i-1,s-v_i) & v_i \text{ é somado} \\
\vee & s = x_i & \text{a soma é apenas } v_i
\end{array}
$$

Exercício

- Projetar um algoritmo para solucionar este problema usando programação dinâmica.

O problema da mochila 0-1

- É dada um conjunto de N itens, cada um com seu valor v_i e seu peso w_i, e uma mochila, que tem uma capacidade máxima K.
- Projetar um algoritmo que determinar o valor máximo que pode ser transportado na mochila.

PPgSC

Elementos de resposta

- Quantas possibilidades temos?
- Porque o problema tem uma sub-estrutura ótima?
- Porque este problema tem sub-problemas sobrepostos?

PPgSC

Elementos de resposta

- Quantas possibilidades temos?
- Porque o problema tem uma sub-estrutura ótima?
- Porque este problema tem sub-problemas sobrepostos?
- Seja $C(i, m)$ o valor máximo carregável considerando
  1. apenas os i primeiros itens
  2. uma capacidade máxima de m
- Casos de base: $C(0, m) =$, $C(i, 0) =$
- Caso geral: se $w_i > m$, então $C(i, m) =$
- Caso geral: se $w_i \leq m$, então $C(i, m) =$

PPgSC

Elementos de resposta

- Quantas possibilidades temos?
- Porque o problema tem uma sub-estrutura ótima?
- Porque este problema tem sub-problemas sobrepostos?
- Seja $C(i, m)$ o valor máximo carregável considerando
  1. apenas os i primeiros itens
  2. uma capacidade máxima de m
- Casos de base: $C(0, m) = 0$, $C(i, 0) = 0$
- Caso geral: se $w_i > m$, então $C(i, m) =$
- Caso geral: se $w_i \leq m$, então $C(i, m) =$

PPgSC

Elementos de resposta

- Quantas possibilidades temos?
- Porque o problema tem uma sub-estrutura ótima?
- Porque este problema tem sub-problemas sobrepostos?
- Seja $C(i, m)$ o valor máximo carregável considerando
  1. apenas os i primeiros itens
  2. uma capacidade máxima de m
- Casos de base: $C(0, m) =$, $C(i, 0) =$
- Caso geral: se $w_i > m$, então $C(i, m) = C(i - 1, m)$
- Caso geral: se $w_i \leq m$, então $C(i, m) =$

PPgSC

Elementos de resposta

- Quantas possibilidades temos?
- Porque o problema tem uma sub-estrutura ótima?
- Porque este problema tem sub-problemas sobrepostos?
- Seja $C(i, m)$ o valor máximo carregável considerando
  1. apenas os i primeiros itens
  2. uma capacidade máxima de m
- Casos de base: $C(0, m) =$, $C(i, 0) =$
- Caso geral: se $w_i > m$, então $C(i, m) =$
- Caso geral: se $w_i \leq m$, então
  $C(i, m) = \max\{C(i-1, m), v_i + C(i-1, m-w_i)\}$

Exercício

- Utilizando programação dinâmica, projetar um algoritmo que soluciona o problema da mochila 0,1.
- Adapte o algoritmo para imprimir os itens selecionados para compor a solução ótima.