Aula 11 - Programação Dinâmica

Escalonamento e Cadeia de Multipicação de Matrizes

Prof. Marco Aurélio Stefanes
marco em dct.ufms.br

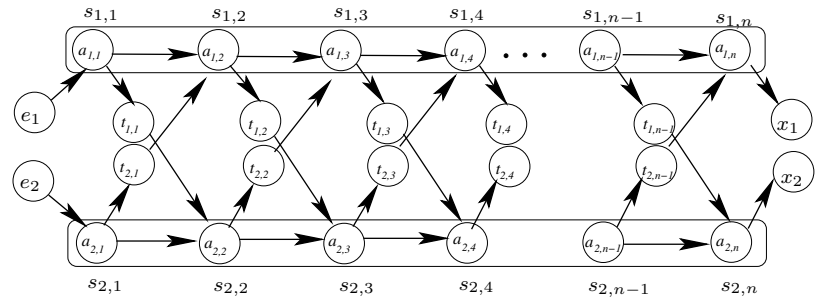
www.dct.ufms.br/ ~ marco

Método de Programação Dinâmica

- Aplicável a problemas de otimização
- Combina soluções de subproblemas
- Subproblemas não são independentes
- Método tabular
- Para desenvolver um algoritmos usando Programação Dinâmica devemos
  - Caracterizar a estrutura da solução ótima
  - Definir recursivamente o valor da solução ótima
  - Computar o valor da solução ótima da forma botton-up
  - Construir um solução ótima a partir dos valores computados

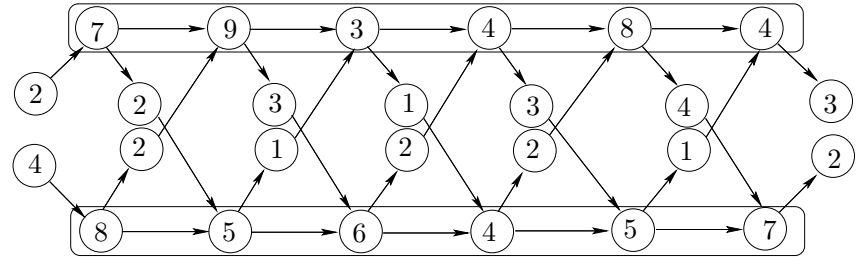
Escalonamento de Linha de Montagem

Um fábrica possui 2 linhas de montagem: Escolher caminho que minimiza o tempo de se produzir um carro.



Algoritmo Força Bruta gasta $\Omega(2^n)$

Escalonamento



Estrutura do caminho mais rápido

- Considere o caminho mais rápido do início até a estação $S_{1,j}$
  - Caso $j = 1$, determine quanto custa chegar até $S_{1,1}$ (Fácil: $e_1 + a_{1,1}$)
  - Caso $j \geq 2$, há duas possibilidades
    - Passar por $S_{1,j-1}$ e ir diretamente sem custo para $S_{1,j}$
    - Passar por $S_{2,j-1}$ e transferir para $S_{1,j}$ pagando o custo de transferência $t_{2,j-1}$

Estrutura do caminho mais rápido

- Suponha que o caminho mais rápido até $S_{1,j}$ passa por $S_{1,j-1}$
  Observação chave: devemos usar o caminho mais rápido para chegar até $S_{1,j-1}$. Se houver outro caminho mais rápido até $S_{1,j-1}$ o caminho escolhido até $S_{1,j}$ não seria o mais rápido.
- Suponha que o caminho mais rápido passa por $S_{2,j-1}$
  Similarmente devermos escolher o caminho mais rápido até $S_{2,j-1}$
- De forma geral: Uma solução ótima que passa por $S_{i,j}$ contém dentro dela soluções ótimas para $S_{1,j-1}$ ou $S_{2,j-1}$

Esta propriedade é chamada Subestrutura Ótima

Como usar substrutura ótima

Para construir a solução ótima devemos também saber como chegar até $S_{2,j}$. De forma similar

- Caminho mais rápido do início até $S_{2,j}$. Há duas possibilidades:
  - Caminho mais rápido até $S_{2,j-1}$ e ir direto para $S_{2,j}$
  - Caminho mais rápido até $S_{1,j-1}$ e ir para $S_{2,j}$ pagando o custo de transferência $t_{1,j-1}$

Portanto para escolher o caminho mais rápido até $S_{1,j}$ e $S_{2,j}$ devemos encontrar o caminho mais rápidp até $S_{1,j-1}$ e $S_{2,j-1}$

Solução recursiva

Definições

- $f_i[j] :=$ tempo mínimo para chegar até $S_{i,j}$
- $f^* := \min\{f_1[n] + x_1, f_2[n] + x_2\}$ Solução
- $f_1[1] := e_1 + a_{1,1}$
- $f_2[1] := e_2 + a_{2,1}$
- Para $j \geq 2$ temos
  - $f_1[j] := \min\{f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}$
  - $f_2[j] := \min\{f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}$
- $f_i[j] :=$ valor da solução ótima para chegar até $S_{i,j}$

Solução recursiva

Definições

- $l_i[j] :=$ linha (1 ou 2) cuja estação $j-1$ é usada no caminho mais rápido até $s_{i,j}$
- Assim: $S_{l_{i[j]},j-1}$ precede $S1, j$ para $j \geq 2$
- $l^* :=$ linha (1 ou 2) cuja estação n é usada
- Combinando temos a equação recursiva

$$f_{1,j} = \begin{cases} e_1 + a_{1,1} & \text{se } j = 1 \\ \min\{f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}\} & \text{se } j \geq 2 \end{cases}$$

$$f_{2,j} = \begin{cases} e_2 + a_{2,1} & \text{se } j = 1 \\ \min\{f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}\} & \text{se } j \geq 2 \end{cases}$$

Computação do valor de uma solução

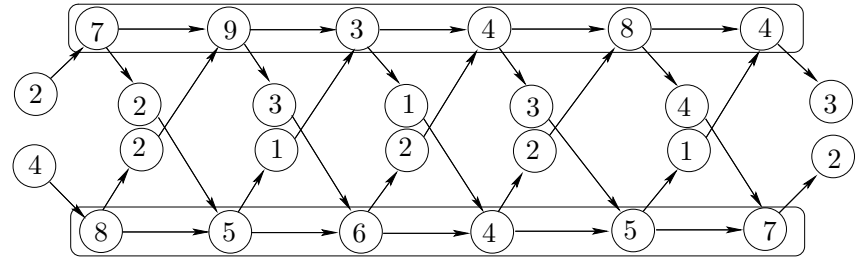
- Algoritmo recursivo: Gasta tempo exponecial
- $r_i(j) =$ referência a $f_i[j]$
- $r_1(n) = r_2(n) = 1$
- $r_1(j) = r_2(j) = r_1(j+1) + r_2(j+1)$ para $j = 1, .., n-1$

Afirmação: $r_i(j) = 2^{n-j}$
Prova: Por indução em j

- $f_1[1]$ é referenciada 2^{n-1} vezes
- Top-Down não é uma boa idéia
- OBS $f_1[j]$ depende apenas de $f_1[j-1]$ e de $f_2[j-1]$
- Abordagem Botton-up

Escalonamento de linha de montagem

Exemplo



Escalonamento de linha de montagem

j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

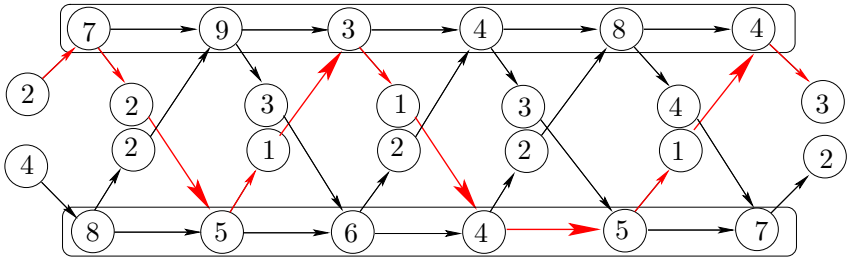
$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

Escalonamento de linha de montagem

Solução do Exemplo



fastest_way(a,t,e,x,n)

1: $f_1[1] = e_1 + a_{1,1}$
2: $f_2[1] = e_2 + a_{2,1}$
3: **for** $i = 1$ to n **do**
4: **if** $f_1[j-1] + a_{1,j} \leq f_2[j-1] + t_{2,j-1} + a_{1,j}$ **then**
5: $f_1[j] = f_1[j-1] + a_{1,j}$
6: $l_1[j] = 1$
7: **else**
8: $f_1[j] = f_2[j-1] + t_{2,j-1} + a_{1,j}$
9: $l_1[j] = 2$
10: **if** $f_2[j-1] + a_{2,j} \leq f_1[j-1] + t_{1,j-1} + a_{2,j}$ **then**
11: $f_2[j] = f_2[j-1] + a_{2,j}$
12: $l_1[j] = 2$
13: **else**
14: $f_2[j] = f_1[j-1] + t_{1,j-1} + a_{2,j}$
15: $l_1[j] = 1$
16: **if** $f_1[n] + x_1 \leq f_2[n] + x_2$ **then**
17: $f^* = f_1[n] + x_1$
18: $l^* = 1$
19: **else**
20: $f^* = f_2[n] + x_2$
21: $l^* = 2$

Construção da Solução ótima

Print_Station(l, n)
1: $i = l^*$
2: escreva linha i estação n
3: **for** $j = n$ downto n **do**
4: $i = l_i[j]$
5: escreva linha i estação $j - 1$

Complexidade:

O tempo total do problema é $\Theta(n)$

Produto de Cadeia de Matrizes

Multiplicação de duas Matrizes

$$C_{p\times r} = A_{p\times q} * B_{q\times r}$$

Tempo : $\Theta(pqr)$
Problema: Produto de Cadeia de Matrizes

- Computar $A = A_1 * A_2 * \ldots * A_n$
- A_i é de ordem $p_{i-1} \times p_i$
- O Objetivo é: encontrar uma parentização que minimiza o número total de múltiplicações
- Exemplo: Sejam $A_{10\times 100}$, $B_{100\times 5}$ e $C_{5\times 50}$
- $((A * B) * C) = 5000 + 2500 = 7500$ operações
- $(A * (B * C)) = 2500 + 5000 = 75000$ operações

Quantas posssibilidades?

Estrutura de um parentização ótima

Seja $A_{i..j} = A_i * A_{i+1} * .. * A_j$
Numa solução ótima exite um $k,\ 1 \leq k \leq n$ tq primerios os produtos $A_1 * .. * A_k$ e $A_{k+1} * .. * A_n$ são computados e depois multiplicados para obter $A_{1..n}$

- O Custo da Solução ótima é o custo de se obter $A_1 * .. * A_k$, $A_{k+1} * .. * A_n$ e $A_{1..k} * A_{k+1..n}$
- Seja $m[i,j]$ o número de multiplicações para se obter $A_{i..j}$ numa solução ótima. Assim $m[1,n]$ é o custo de uma solução ótima para o problema
- Podemos definir $m[i,j]$ recursivamente

$$m[i,j] = \begin{cases} 0 & \text{se } j = i \\ \min_{i \leq k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{i<j} \end{cases}$$

Algoritmo Recursivo

O Algoritmo baseado na definição recursica é simples (Exponencial)

Seja $V(n)$ o número de chamada recursivas necessárias para multiplicar n matrizes

$$V(1) = 1$$

$$V(n) = \sum_{i=1}^{n-1}(V(i) + V(n-i)) + 1$$

$$V(n) = 2\sum_{i=1}^{n-1} V(i) + 1$$

Vamos provar que $V(n) = 3^{n-1}$

Prova Indução em n

$n = 1 \quad V(1) = 1 = 3^{1-1}$

$n > 1\ V(n) = 1 + 2\sum_{i=1}^{n-1} V(i) = 1 + 2\sum_{i=1}^{n-1} 3^{i-1}$

$V(n) = 1 + 2.(3^{n-1} - 1)/2 = 3^{n-1}$

Algoritmo Recursivo

Idéia Aproveitar resultados anteriores para evitar repetições de cálculos dos $m[i,j]$
Exemplo: $A^1_{6\times 7}, A^2_{7\times 3}, A^3_{3\times 1}, A^4_{1\times 2}, A^5_{2\times 4}, A^4_{\times 5}$

Matriz_chain_order(p,n)

Saída: Matrizes m e s

```
1: for i = 1 to n do
2:     m[i, j] = 0
3: for l = 1 to n − 1 do
4:     for i = 1 to n − l do
5:         j = i + l
6:         m[i, j] = ∞
7:         for k = i to j − 1 do
8:             q = m[i, k] + m[k + 1] + p_{i−1} p_k p_j
9:             if q < m[i, j] then
10:                m[i, j] = q
11:                s[i, j] = k
```

Complexidade: $O(n^3)$

Construção da Solução Ótima

Multiplica_em_cadeia(A, s, i, j)

```
1: if i < j then
2:     x = Multiplica_em_cadeia(A, s, i, s[i, j])
3:     x = Multiplica_em_cadeia(A, s, s[i, j] + 1, j)
4:     devolva Multiplica(x,y)
5: else
6:     devolva A_i
```

Chamada Inicial:
Multiplica_em_cadeia($A, s, 1, n$)