TÉCNICAS DE PROGRAMAÇÃO AVANÇADA

Série 6 – Programação Dinâmica

Introdução

- Programação Dinâmica – Também conhecida como recursão tabular
  - Método de resolver problemas combinando as soluções dos sub-problemas que formam o problema.
  - útil quando os sub-problemas compartilham sub-problemas.
    - Nestes casos algoritmos baseados no paradigma **dividir para conquistar** trabalham muito mais pois resolvem um mesmo sub-problema várias vezes.

Introdução

- Na solução baseada em programação dinâmica
  - o resultado dos sub-problemas são gravados em uma tabela
  - Esses resultados podem ser consultados evitando o recálculo toda vez que a solução de um sub-problema é necessária.
  - PD é muito utilizada na solução de problemas de otimização
    - Nesses problemas podem haver várias soluções com diversos valores e desejamos encontrar a solução com o valor ótimo.

Soluções usando PD

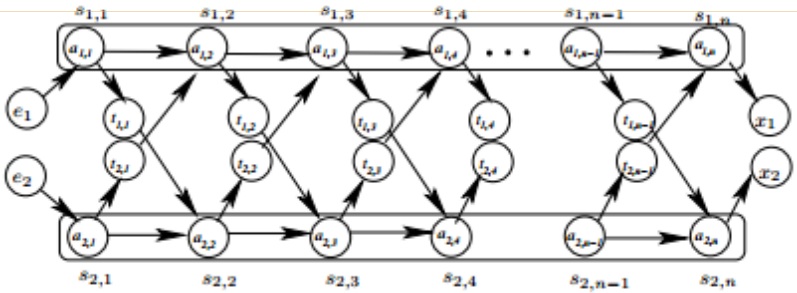
- Um algoritmo usando PD é obtido pelos seguintes passos:
  - Caracterização da solução ótima
  - Obtenção da solução ótima recursiva
  - Calcular o valor de uma solução ótima em um processo botton-up.
  - Construir a solução ótima a partir das informações calculadas.

Ex: Problema da linha de montagem flexível

- Situação:
  - Duas linhas de montagem
  - Cada linha com 6 estações
  - Cada estação com tempo a_{ij}
  - Tempo de mudar de uma estação para outra na mesma linha é desprezível
  - Tempo para mudar de uma estação para outra em linhas diferentes é t_{ij}
- Problema: Determinar que estações escolher para minimizar o tempo total de um único automóvel na fábrica

Ex: Problema da linha de montagem

- Nota-se que o espaço de solução tem tamanho 2^n
  - Logo, um algoritmo de força bruta teria complexidade de tempo igual a $\Omega(2^n)$



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

Caracterizando a estrutura da solução

- Seja fi[j] o tempo mais rápido possível para se chegar desde o ponto de partida até a estação sij
- O objetivo é descobrir o tempo mais rápido para fazer todo o percurso da fábrica (f*)
  - f* = min (f1[n] + x1, f2[n]+x2)
  - F1[1]= e1 + a1,1, e1 é o tempo de entrada em L1, a11 é o tempo de s11
  - F2[1]=e2 + a2,1, e2 é o tempo de entrada em L2, a2,1, o tempo de s2,1

Solução recursiva

- Temos então que
  - $f1[j] = f1[j-1] + a_{i,j}$ e
  - $f1[j] = f2[j-1] + t2,j-1 + a1,j$
- Desse modo
  - $f1[j] = \min(f1[j-1] + a1,j, f2[j-1] + t2,j-1 + a1,j)$

Para j = 2..n

E

- $f2[j] = \min(f2[j-1] + a2,j, f1[j-1] + t1,j-1 + a2,j)$

Solução recursiva

- Combinando as equações temos
  - f1[j] =
    - e1+a11, para j = 1
    - min(f1[j-1] + a1,j, f2[j-1] + t2,j-1 + a1,j), para j = 2..n
  - f2[j] =
    - e2+a21, para j = 1
    - min(f2[j-1] + a2,j, f1[j-1] + t1,j-1 + a2,j), para j = 2..n

Resultado

Resultados para a instância do problema (T. Commer pg. 261)

	1	2	3	4	5	6
F1[j]	9	18	20	24	32	35
f[2[j]	12	16	22	25	30	37

f* = 38

	2	3	4	5	6
L1[j]	1	2	1	1	2
L2[j]	1	2	1	2	2

l* = 1

Algoritmo em Programação Dinâmica

- Calcula-se os fi[j] valores em uma ordem diferente da recursiva:
  - Para j<=1 fi[j] depende apenas dos valores de f1[j-1] e f2[j-1]
  - Então calculando-se os fi[j] na ordem crescente de j, da esquerda para a direita teremos um algoritmo $\Theta(n)$.

Algoritmo em Programação Dinâmica

```
Alg. CaminhoMaisRapido(a,t,e,c,n)
f1[1]← e1+a1,1
f2[1]← e2+a2,1
Para j← 2 até n faça
    se (f1[j-1] + a1,j <= f2[j-1] + t2, j-1 + a1,j)
    então
      f1[j]← f1[j-1] + a1,j
      l1[j] = 1
    senão
      f1[j] ← f2[j-1] + t2, j-1 + a1,j
      l1[j] ← 2
    Fim se  /*fazer condicional identica para
     determinar f2[j]*/
Fim para
```

```
Se  f1[n]+x1 <= f2[n]+x2
então
    f*=f1[n]+x1; l*=1
Senão
  f*=f2[n]; l* = 2
```

Apresentando a solução ótima

- Para apresentar a solução:

```
imprimeEstacoes(l,n)
    i=l*
    Imprime (“Linha: “ i, “estação”, n);
    Para j ← n até 2 faça
            i=li[j] Imprime (“Linha: “ i, “estação”, j-1);
Fim
```

Multiplicação seqüências de Matrizes

- Problema base: Dado um sequência de n Matrizes (A1, ..., An), com dimensão pi-1 x pi, ($1 \leq i \leq n$) obter o resultado da multiplicação das mesmas.
- Problema de otimização: Encontrar a ordem de multiplicação que minimize o número de multiplicações escalares.
  - Dado que, para multiplicar uma matriz de dimensões p x q, com uma matriz de dimensões q x r, pqr operações serão necessárias.

Multiplicação seqüências de Matrizes

- Exemplo: para as seguintes matrizes:
  1. $M1_{3x2}$
  2. $M2_{2x4}$
  3. $M3_{4x1}$
     - Temos que a multiplicação de (M1xM2) , produzirá (3x2x4) = 24 operações escalares e resultará em uma matriz R de dimensão 3x4.
     - Assim, a multiplicação da matriz R_{3x4} x $M3_{4x1}$ produzirá 12 operações escalares
     - Logo, a multiplicação considerando a ordem (M1xM2) x M3 irá realizar 24 +12 = 36 operações escalares.
  - Pergunta? Existe uma ordem de multiplicação para esta sequência que produza menos operações escalares?

Multiplicação seqüências de Matrizes

- Exemplo:
  - M= $M1_{10x20}$ x $M2_{20x50}$ x $M3_{50x1}$ x $M4_{1x100}$
  - Qual o número de multiplicações escalares para
    1. M1xM2xM3xM4?
    2. (M1xM2)x(M3xM4) ?
    3. M1x(M2xM3xM4)?
    4. (M1xM2xM3)xM4?
    5. M1x(M2xM3)xM4?

Multiplicação seqüências de Matrizes

- Tentar todas as possibilidades até encontrar a melhor ordem tem complexidade exponencial
  - $F(n) = O(2^{n-2})$, onde n é o número de matrizes.
- Já o algoritmo de programação dinâmica possui complexidade cúbica.
  - $F(n) = O(n^3)$

Modelo do Problema

- Seja $A_{i..j}$, $i<=j$, o resultado da multiplicação da sequência de matrizes a ser obtida.
- Qualquer colocação ótima de parênteses para $i<j$ divide o produto entre A_k e A_{k+1}, $i<=k<j$.
- Ou seja, para algum valor de k, computar primeiro $A_{i..k}$ e depois $A_{k+1..j}$ e depois multiplica estes dois resultados
- Exemplo: para $A_{1..4}$ e k=2 teremos a ordem (A1xA2) x (A3xA4).
  - Se $X_{i..k}$ é total de multiplicações de A1..k e $X_{k+1..j}$ é o total de multiplicações de $A_{k+1..j}$, então o número de multiplicações total é

$$X_{i..k} + X_{k+1..j} + d_{i-1} x d_k x d_j$$

Modelo do problema

- Seja m_{ij} o menor custo para computar o produto
- M_i x M_{i+1} x .... M_j, para $1 \leq i \leq j \leq n$
- Nesse caso:
  - $M_{ij} = 0$ se $i=j$
  - $M_{ij} = \text{Min}_{i \leq k < j} (m_{ik} + m_{k+1,j} + d_{i-1} x d_k x d_j)$ se $j>i$

Modelo do problema

- O termo m_{ik} representa o custo mínimo para calcular:
- $M' = M_i \times M_{i+1} \times \ldots . \times M_k$
- O termo $m_{k+1,j}$ representa o custo mínimo para calcular:
- $M'' = M_{k+1} \times M_{k+2} \times \ldots . \times M_j$
- O termo $d_{i-1} d_k d_j$ representa o custo de multiplicar $M'[d_{i-1}, d_k]$ por $M''[d_k, d_j]$

Modelo do problema

m11=0	m22=0	m33=0	m44=0
m12=?	m23=?	m34=?	
m13=?	m24=?		
m14=?			

Algoritmo (Cormen)

```
Matrix-chain-order(p)
N ← comprimento[p]-1
Para i ← 1 até n faça
    m[i,i]←0
Para l← 2 até n faça
  para i← 1 até n-l+1
   faça
       j← i+l-1
       m[i,j]← INF
Para k← i até j-1 faça
  q←m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
Se q < m[i,j] então
    m[i,j]← q
     s[i,j]←k
Fim para // l
Fim para  // i
Fim para  // k
Retorna m e s
```

Algoritmo Geral