

Apostila Introdutória de Algoritmos

Guilherme D. da Fonseca
Celina M. H. de Figueiredo

Versão rascunho para o curso de Análise de Algoritmos da Unirio
2009.

18 de Setembro de 2009

Programação Dinâmica

A técnica de programação dinâmica é uma técnica de decomposição que resolve um problema decompondo-o em subproblemas cujas soluções são armazenadas em uma tabela.

5.1. Ordem de Multiplicação de Matrizes

Imagine que você tem que multiplicar três matrizes A , B e C , na mão, usando apenas papel e lápis. Considere que A é uma matriz 10×2 (10 linhas e 2 colunas), B é uma matriz 2×20 e C é uma matriz 20×5 . Imagine que o tempo está correndo e quanto mais rápido você resolver o problema, maior será sua nota. O que você faz? Se a sua resposta é: ‘começo a multiplicar imediatamente’, então você provavelmente fez a escolha errada. Vamos contar quantas multiplicações você terá que fazer.

Para multiplicar A por B , você fará $10 \cdot 2 \cdot 20 = 400$ multiplicações. Em seguida, para multiplicar (AB) por C , você fará $10 \cdot 20 \cdot 5 = 1000$ multiplicações. No total, fará $400 + 1000 = 1400$ multiplicações.

Porém, se você olhar para o problema com um pouco mais de cuidado, poderá notar que vale mais a pena começar multiplicando B por C , fazendo $2 \cdot 20 \cdot 5 = 200$ multiplicações. Em seguida, você multiplica A por (BC) , fazendo mais $10 \cdot 2 \cdot 5 = 100$ multiplicações. No total, você faz $200 + 100 = 300$ multiplicações, enquanto quem começou multiplicando as matrizes na ordem fornecida fez 1100 multiplicações a mais!

Note que esta escolha da ordem da multiplicação é possível porque a multiplicação de matrizes, embora não seja comutativa, é associativa. Imagine que um computador tem que multiplicar uma seqüência de n matrizes. Não há dúvida que vale a pena, antes de iniciar a multiplicação, escolher a melhor ordem para fazê-lo. Isto é válido independente do algoritmo usado para fazer a multiplicação em si. Este é o problema estudado nesta sessão.

PROBLEMA 13. *Dada uma seqüência de n matrizes A_1, \dots, A_n , escolher a ordem para multiplicá-las que minimiza o tempo total gasto.*

O primeiro passo para resolvermos o problema é nos familiarizarmos com ele. Não precisamos nos preocupar com o conteúdo das matrizes que desejamos multiplicar, apenas com suas dimensões. Como, para multiplicarmos a matriz A pela matriz B , a largura de A tem que ser igual a altura de B , podemos condensar as dimensões das n matrizes que desejamos multiplicar em um vetor v com $n + 1$ posições, contendo as dimensões das matrizes, ou seja, M_i , a i -ésima matriz da multiplicação, tem dimensões $v_i \times v_{i+1}$. No nosso exemplo do início da sessão, o vetor seria $v = (10, 2, 20, 5)$. Nosso algoritmo não assumirá nada sobre o tempo gasto para multiplicar duas matrizes. Consideraremos que o tempo gasto para multiplicar uma matriz $a \times b$ por outra matriz $b \times c$ é $f(a, b, c)$. Esta função será considerada conhecida, e será avaliada pelo nosso algoritmo diversas vezes. Normalmente, porém, considerar $f(a, b, c) = abc$ é uma boa escolha, portanto, usaremos esta definição para os exemplos concretos.

Construiremos nossa solução de baixo para cima, ou seja, partiremos de problemas menores até chegarmos ao problema total que desejamos resolver. Vamos criar um vetor bidimensional $T[1 \dots n, 1 \dots n]$ e preencheremos na posição $T[i, j]$ a melhor maneira de multiplicarmos as matrizes de A_i até A_j . Queremos, no final, obter $T[1, n]$, a solução para nosso problema. Para simplificarmos nossa explicação, computaremos apenas o tempo gasto na ordem ótima de multiplicação, e não a maneira explícita de fazê-lo. Porém, não é difícil usar o mesmo método para

	1	2	3	4	5
1	0	400	300	290	620
2		0	200	230	320
3			0	300	1200
4				0	225
5					0

TABELA 5.1. Tabela T tal que $T[i, j]$ é o custo de multiplicar as matrizes de M_i até M_j , onde M_k é uma matriz $v_k \times v_{k+1}$ segundo $v = (10, 2, 20, 5, 3, 15)$. Consideramos o custo de multiplicar uma matriz $a \times b$ por uma matriz $b \times c$ como sendo $f(a, b, c) = abc$.

obter realmente a maneira como as multiplicações devem ser realizadas. Vamos começar pelos casos triviais.

Quando temos apenas uma matriz, não há nada a fazer, portanto $T[i, i] = 0$, para $1 \leq i \leq n$. Quando temos apenas duas matrizes para multiplicar, há apenas uma maneira de fazê-lo, portanto $T[i, i+1] = f(v_i, v_{i+1}, v_{i+2})$, para $1 \leq i \leq n-1$. Quando temos três matrizes, A, B, C , podemos multiplicar primeiro AB ou BC . Assim, para minimizarmos o custo, fazemos $T[i, i+2] = \min(T[i, i+1] + f(v_i, v_{i+2}, v_{i+3}), T[i+1, i+2] + f(v_i, v_{i+1}, v_{i+3}))$. De um modo geral, para multiplicarmos as matrizes de M_i até M_j , podemos, para $i \leq k < j$, multiplicar primeiro as matrizes de M_i até M_k e também de M_{k+1} até M_j e depois multiplicarmos as duas matrizes obtidas. Temos então:

$$T[i, j] = \min_{k=i}^{j-1} (T[i, k] + T[k+1, j] + f(v_i, v_{k+1}, v_{j+1})).$$

Um exemplo da tabela T para a entrada $v = (10, 2, 20, 5, 3, 15)$ está na tabela 5.1. No exemplo, a função de custo usada foi $f(a, b, c) = abc$. Preenchemos as células $T[i, j]$ da tabela 5.1 em ordem não decrescente da diferença de subscrito, ou seja, primeiro preenchemos a diagonal principal com as células $T[i, i]$, em seguida a diagonal com as células $T[i, i+1]$, e assim por diante, até a última diagonal que consiste da célula $T[1, n]$. Note que para preencher uma célula da $T[i, j]$ tabela, basta consultar células $T[i, k]$ e $T[k, j]$ com k entre i e j . Com isto, é fácil escrever o pseudo-código da figura 5.1.

A complexidade de tempo do algoritmo é claramente $O(n^3)$, onde n é o número de matrizes a ser multiplicadas. Isto ocorre porque a tabela tem $O(n^2)$ posições e, para preencher uma posição, precisamos examinar outras $O(n)$ células da tabela.

Em muitos casos, uma complexidade de tempo cúbica no tamanho da entrada é inaceitável para propósitos práticos, porém, no caso da ordem de multiplicação de matrizes, esta complexidade é perfeitamente aceitável. Afinal, desejamos, após este pré-processamento, realmente multiplicar as matrizes e esta última fase do processo será provavelmente ainda mais demorada. Assim, não é provável que o número de matrizes seja grande a ponto de tornar a utilização de um algoritmo cúbico inviável.

5.2. Todos os caminhos mais curtos

Uma outra aplicação da técnica de programação dinâmica é o problema de todos os caminhos mais curtos num grafo direcionado.

PROBLEMA 14. *Dado um grafo direcionado com pesos positivos nas arestas, encontrar para cada par de vértices o caminho mais curto.*

Neste caso, é dado um grafo direcionado D , definido por dois conjuntos: o conjunto de vértices $V(D) = \{v_1, v_2, \dots, v_n\}$ e o conjunto de arestas $E(D)$, pares ordenados de vértices em $V(D)$. Também é dada uma matrix W de pesos associados às arestas do grafo direcionado. A diagonal da matrix W é composta de zeros, enquanto que para $i \neq j$, $w(i, j)$ é o peso da aresta

Entrada:

n : Número de matrizes M_1, \dots, M_n a serem multiplicadas.

v : Vetor com $n + 1$ posições onde a matriz M_i tem dimensões $v[i] \times v[i + 1]$.

Saída:

Custo total mínimo de multiplicar as matrizes de M_1 até M_n .

Observações:

$f(a, b, c)$: Custo de multiplicar uma matriz $a \times b$ por uma matriz $b \times c$.

OrdemMultMatrizes(n, v)

Para i de 1 até n

$T[i, i] \leftarrow 0$

Para Δ de 1 até $n - 1$

Para i de 1 até $n - \Delta$

$j \leftarrow i + \Delta$

$T[i, j] \leftarrow \infty$

Para k de i até $j - 1$

Se $T[i, j] > T[i, k] + T[k + 1, j] + f(v_i, v_{k+1}, v_{j+1})$

$T[i, j] \leftarrow T[i, k] + T[k + 1, j] + f(v_i, v_{k+1}, v_{j+1})$

Retorne $T[1, n]$

FIGURA 5.1. Solução do problema 13

(v_i, v_j) , caso $(v_i, v_j) \in E(D)$, e caso contrário, quando $(v_i, v_j) \notin E(D)$, $w(i, j)$ é definido como ∞ .

Um caminho no grafo direcionado D é uma seqüência $P = v_1, v_2, \dots, v_k$ de vértices tal que vértices consecutivos na seqüência são adjacentes no grafo direcionado, ou seja, $(v_i, v_{i+1}) \in E(D)$. O comprimento de um caminho P é $w(P) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$. Um menor caminho do vértice u ao vértice v em D é um caminho de u até v em D de comprimento mínimo e a distância de u a v em D é o comprimento do menor caminho de u até v em D .

Considere um menor caminho $P = v_1, v_2, \dots, v_k$ de v_1 até v_k . Chame de P_{ij} o subcaminho de P de v_i até v_j , definido pela seqüência de vértices v_i, v_{i+1}, \dots, v_j . Claramente, P_{ij} é um menor caminho de v_i até v_j . Chamamos esta observação de princípio de otimalidade. O algoritmo que descrevemos a seguir é consequência deste princípio.

O algoritmo considera uma seqüência W_0, W_1, \dots, W_n de n matrizes $n \times n$. Definimos W_0 como a matriz de pesos W de entrada. A matriz W_1 contém como entrada $w^1(i, j)$ o comprimento de um menor caminho de v_i até v_j , sujeito à condição de que os vértices intermediários pertencem a $\{v_1\}$. A matriz W_2 contém como entrada $w^2(i, j)$ o comprimento de um menor caminho de v_i até v_j , sujeito à condição de que os vértices intermediários pertencem a $\{v_1, v_2\}$. E assim por diante, finalmente a última matriz W_n contém como entrada $w^n(i, j)$ o comprimento de um menor caminho de v_i até v_j , sujeito à condição de que os vértices intermediários pertencem a $\{v_1, v_2, \dots, v_n\}$, ou seja, trata-se da matriz solução. Veja o pseudo-código correspondente na figura 5.2.

Podemos estabelecer a corretude deste algoritmo provando por indução que cada matriz W_k , onde $1 \leq k \leq n$, computada pelo algoritmo de fato contém na entrada $w^k(i, j)$ o comprimento de um menor caminho de v_i até v_j , sujeito à condição de que os vértices intermediários pertencem a $\{v_1, v_2, \dots, v_k\}$. Para isto, basta aplicar o princípio de otimalidade.

Analisar a complexidade de tempo deste algoritmo é bastante simples. Basta notarmos que há três *loops* aninhados no algoritmo, cada um deles executando no máximo n repetições. Dentro desses *loops* todas as operações levam tempo constante. Assim, a complexidade de tempo do algoritmo é $\Theta(n^3)$.

Entrada:

Grafo direcionado D , com n vértices.
 Matrix de pesos positivos nas arestas W .

Saída:

Matriz de todos os caminhos mais curtos W_n .

TodosCaminhos(n, W)

$W_0 \leftarrow W$

Para k de 1 até n

 Para i de 1 até n

 Para j de i até n

 Se $w^{k-1}(i, k) + w^{k-1}(k, j) < w^{k-1}(i, j)$

$w^k(i, j) \leftarrow w^{k-1}(i, k) + w^{k-1}(k, j)$

Retorne W_n

FIGURA 5.2. Solução do problema 14

5.3. Resumo e Observações Finais

A idéia central do método de programação dinâmica consiste em decompor o problema a ser resolvido em subproblemas e armazenar a solução dos subproblemas evitando que um mesmo subproblema seja calculado repetidamente.

O algoritmo para ordem de multiplicação de matrizes encontra a parentização ótima de modo a minimizar o custo de se multiplicar uma seqüência de matrizes.

O algoritmo para todos os caminhos mais curtos calcula a matriz distância que contém cada uma das distâncias entre os diferentes pares de vértices num grafo direcionado.

Exercícios

- 5.1) A subsequência máxima comum (LCS) de duas seqüências de caracteres T e P é a maior seqüência L tal que L é seqüência de T e de P . A superseqüência mínima comum (SCS) de duas seqüências T e P é a menor seqüência L tal que T e P são seqüências de L .
 Descreva algoritmos que usam programação dinâmica para encontrar LCS e SCS de duas seqüências dadas.
- 5.2) O fecho transitivo de um grafo direcionado $G(V, E_1)$ é um digrafo $T_G(V, E_2)$ tal que se existe caminho de u a v em G , então $uv \in E_2$. Claramente temos $E_1 \subseteq E_2$. Descreva um algoritmo que constroi a matriz de adjacências $A(T_G)$ de T_G dada a matriz de adjacências de G . Esta matriz é chamada de matriz de alcançabilidade de G .
 O seu algoritmo usa programação dinâmica?
- 5.3) Dado o grafo direcionado D , onde $V(D) = \{v_1, v_2, v_3, v_4, v_5\}$ e $E(D) = \{(v_1v_2), (v_1v_3), (v_1v_5), (v_2v_4), (v_2v_5), (v_3v_2), (v_4v_1), (v_4v_3), (v_5v_4)\}$, com pesos nas arestas: $w(v_1v_2) = 9$, $w(v_1v_3) = 14$, $w(v_1v_5) = 2$, $w(v_2v_4) = 7$, $w(v_2v_5) = 13$, $w(v_3v_2) = 10$, $w(v_4v_1) = 8$, $w(v_4v_3) = 1$, $w(v_5v_4) = 12$. Pede-se a matriz M que tem como entradas $m(i, j)$, a distância do vértice v_i ao vértice v_j em D .
- 5.4) Um ladrão encontra um cofre com N tipos de objetos, de tamanho e valor variados, vários objetos de cada tipo, e carrega uma mochila com capacidade M . O objetivo do ladrão é encontrar a combinação de objetos que cabe na mochila e maximiza o valor.
 Descreva um algoritmo que usa programação dinâmica para resolver este problema da mochila.
- 5.5) São dadas n chaves s_1, s_2, \dots, s_n com correspondentes pesos p_1, p_2, \dots, p_n . Queremos encontrar a árvore binária de busca que minimize a soma, sobre todas as chaves, dos

pesos vezes a distância da chave à raiz (o custo de acessar a chave associada àquele vértice da árvore).

Descreva um algoritmo que usa programação dinâmica para resolver este problema da árvore binária de busca ótima.