Dynamic Programming Matrix Chain Product

Design and Analysis of Algorithms

zeshan.khan@nu.edu.pk

Matrix Chain Multiplication

- Given some matrices to multiply, determine the best order to multiply them so you minimize the number of single element multiplications.
  - i.e. Determine the way the matrices are parenthesized.
- First off, it should be noted that matrix multiplication is associative, but not commutative. But since it is associative, we always have:
- ((AB)(CD)) = (A(B(CD))), or any other grouping as long as the matrices are in the same consecutive order.
- BUT NOT: ((AB)(CD)) = ((BA)(DC))

Matrix Chain Multiplication

- It may appear that the amount of work done won't change if you change the parenthesization of the expression, but we can prove that is not the case!
- Let us use the following example:
  - Let A be a 2x10 matrix
  - Let B be a 10x50 matrix
  - Let C be a 50x20 matrix
- But FIRST, let's review some matrix multiplication rules…

Matrix Chain Multiplication

- Let's get back to our example: We will show that the way we group matrices when multiplying A, B, C ***matters***:
  - Let A be a 2x10 matrix
  - Let B be a 10x50 matrix
  - Let C be a 50x20 matrix
- Consider computing **A(BC):**
  - # multiplications for (BC) = 10x50x20 = 10000, creating a 10x20 answer matrix
  - # multiplications for A(BC) = 2x10x20 = 400
  - Total multiplications = 10000 + 400 = 10400
- Consider computing **(AB)C**:
  - # multiplications for (AB) = 2x10x50 = 1000, creating a 2x50 answer matrix
  - # multiplications for (AB)C = 2x50x20 = 2000,
  - Total multiplications = 1000 + 2000 = 3000

Matrix Chain Multiplication

- Thus, our goal today is:
- Given a chain of matrices to multiply, determine the fewest number of multiplications necessary to compute the product.

Matrix Chain Multiplication

- Formal Definition of the problem:
  - Let $A = A_0 \bullet A_1 \bullet \ldots A_{n-1}$
  - Let $\mathbf{N_{i,j}}$ denote the minimal number of multiplications necessary to find the product:
    - $A_i \bullet A_{i+1} \bullet \ldots A_j$.
  - And let $\mathbf{d_i x d_{i+1}}$ denote the dimensions of matrix A_i.
- We must attempt to determine the minimal number of multiplications necessary($\mathbf{N_{0,n-1}}$) to find A,
  - assuming that we simply do each single matrix multiplication in the standard method.

Matrix Chain Multiplication

- Our final multiplication will ALWAYS be of the form
  - $(A_0 \bullet A_1 \bullet \ldots A_k) \bullet (A_{k+1} \bullet A_{k+2} \bullet \ldots A_{n-1})$
- There is exactly one value of k for which we should "split" our work into two separate cases so that we get an optimal result.
  - Here is a list of the cases to choose from:
  - $(A_0) \bullet (A_1 \bullet A_{k+2} \bullet \ldots A_{n-1})$
  - $(A_0 \bullet A_1) \bullet (A_2 \bullet A_{k+2} \bullet \ldots A_{n-1})$
  - $(A_0 \bullet A_1 \bullet A_2) \bullet (A_3 \bullet A_{k+2} \bullet \ldots A_{n-1})$
  - ...
  - $(A_0 \bullet A_1 \bullet \ldots A_{n-3}) \bullet (A_{n-2} \bullet A_{n-1})$
  - $(A_0 \bullet A_1 \bullet \ldots A_{n-2}) \bullet (A_{n-1})$
- Basically, count the number of multiplications in each of these choices and **pick the minimum**.
  - One other point to notice is that you have to account for the minimum number of multiplications in each of the two products.

Matrix Chain Multiplication

- Consider the case multiplying these 4 matrices:
  - A: 2x4
  - B: 4x2
  - C: 2x3
  - D: 3x1
- 1. (A)(BCD) - This is a 2x4 multiplied by a 4x1,
  - so 2x4x1 = 8 multiplications, plus whatever work it will take to multiply (BCD).
- 2. (AB)(CD) - This is a 2x2 multiplied by a 2x1,
  - so 2x2x1 = 4 multiplications, plus whatever work it will take to multiply (AB) and (CD).
- 3. (ABC)(D) - This is a 2x3 multiplied by a 3x1,
  - so 2x3x1 = 6 multiplications, plus whatever work it will take to multiply (ABC).

Matrix Chain Multiplication

- The optimal way of matrices multiplication from our observations is as.
- Split matrix chain in to two chains by finding the splitting position as
  - Minimum number of multiplications needed to perform in both chains
  - Minimum number of multiplications needed to multiply final result of these two chains.

Matrix Chain Multiplication

- **Our recursive formula:**
  - $N_{i,j}$ = min value of $N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1}$, over all valid values of k.

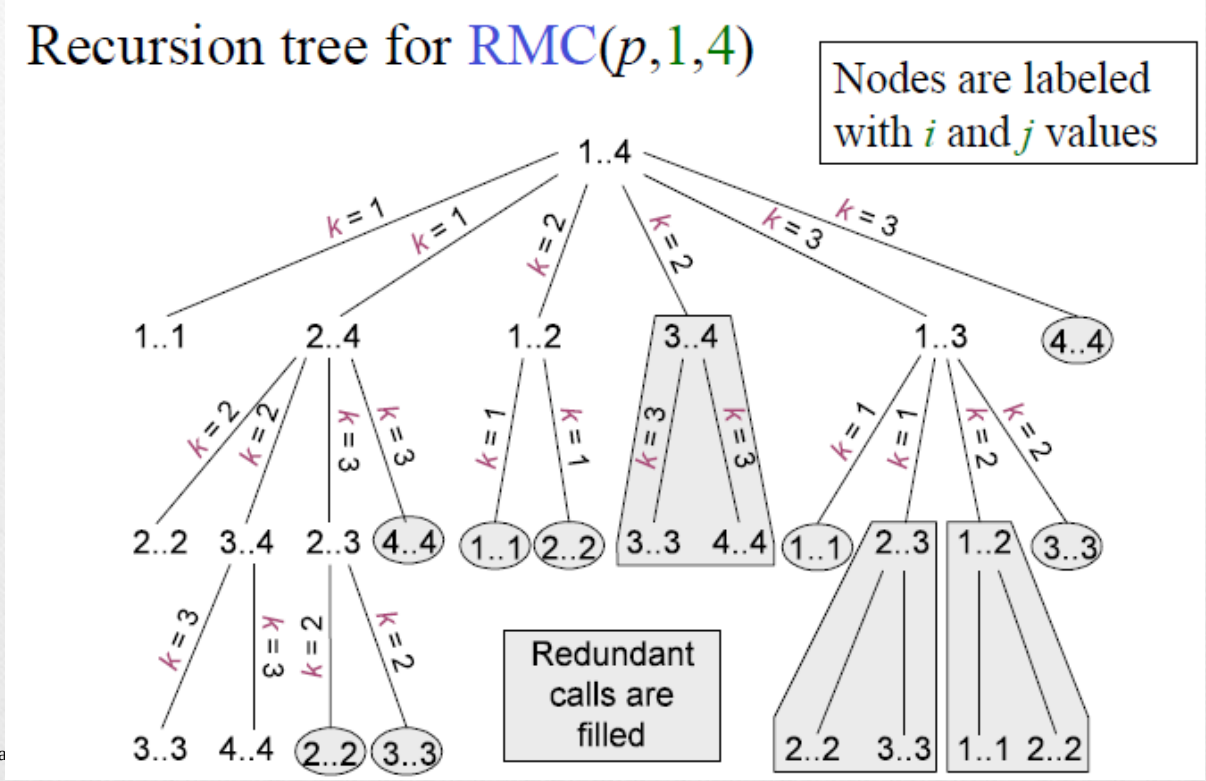
Matrix Chain Multiplication: Formal

- $N_{i,j} = \min_{i \leq k < j} (N_{i,k} + N_{k+1,j} + (d_i * d_{k+1} * d_{j+1}))$
- $N_{i,j} = 0 \; if \; i == j$

Matrix Chain Multiplication

- **Recursive Algorithm:**
- N(i,j,d)
  - if(i==j) return 0;
  - else
    - min=infinity;
    - min=Loop_rec(min)
    - return min
- Loop_rec(min)
- for(int k=i+1;k<j;k++)
  - if(min>N(i,k)+N(k+1,j)
    +(d[i]*d[k+1]*d[j+1]))
    - min=N(i,k)+N(k+1,j)
      +(d[i]*d[k+1]*d[j+1])
- return min

Recursive Matrix-chain Order



Elements of Dynamic Programming

- When should we look for a DP solution to an optimization problem?
- Two key ingredients for the problem
  - Optimal substructure
  - Overlapping subproblems

Running Time of Matrix Chain

$$T(1) \geq 1$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} \left(T(k) + T(n-k) + 1 \right) \text{ for } n > 1$$

- For $i = 1, 2, \ldots, n$ each term $T(i)$ appears twice
  - Once as $T(k)$, and once as $T(n-k)$
- Collect $n-1$ 1's in the summation together with the front 1

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

- Prove that $T(n) = \Omega(2^n)$ using the substitution method

Running Time of RMC: Prove that $T(n) = \Omega(2^n)$

- Try to show that $T(n) \geq 2^{n-1}$ (by substitution)

Base case: $T(1) \geq 1 = 2^0 = 2^{1-1}$ for $n = 1$

IH: $T(i) \geq 2^{i-1}$ for all $i = 1, 2, \ldots, n-1$ and $n \geq 2$

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

$$= 2 \sum_{i=0}^{n-2} 2^i + n = 2(2^{n-1} - 1) + n$$

$$= 2^{n-1} + (2^{n-1} - 2 + n)$$

$$\Rightarrow T(n) \geq 2^{n-1}$$ Q.E.D.

Running Time of RMC: $T(n) \geq 2^{n-1}$

Whenever

- a recursion tree for the natural recursive solution to a problem contains the same subproblem repeatedly
- the total number of different subproblems is small

it is a good idea to see if DP can be applied

Memoized Recursive Algorithm

- Offers the efficiency of the usual DP approach while maintaining top-down strategy
- Maintains an entry in a table for the solution to each subproblem
- Each table entry contains a special value to indicate that the entry has yet to be filled in
- When the sub-problem is first encountered its solution is computed and then stored in the table
- Each subsequent time that the sub-problem encountered the value stored in the table is simply looked up and returned
- The approach assumes that
  - The set of all possible sub-problem parameters are known
  - The relation between the table positions and subproblems is established

Memoized Recursive Algorithm

```
MEMOIZED-MATRIX-CHAIN(p)
1  n ← length[p] − 1
2  for i ← 1 to n
3       do for j ← i to n
4            do m[i, j] ← ∞
5  return LOOKUP-CHAIN(p, 1, n)
```

```
LOOKUP-CHAIN(p, i, j)
1  if m[i, j] < ∞
2     then return m[i, j]
3  if i = j
4     then m[i, j] ← 0
5     else for k ← i to j − 1
6               do q ← LOOKUP-CHAIN(p, i, k)
                       + LOOKUP-CHAIN(p, k + 1, j) + p_{i−1} p_k p_j
7                  if q < m[i, j]
8                     then m[i, j] ← q
9  return m[i, j]
```

Matrix Chain Multiplication

- Algorithm:

Initialize N[i][i] = 0, and all other entries in N to ∞.

for i=1 to n-1 do the following

for j=0 to n-1-i do

for k=j to j+i-1

if (N[j][j+i-1] > N[j][k]+N[k+1][j+i-1]+$d_j d_{k+1} d_{i+j}$)

N[j][j+i-1]= N[j][k]+N[k+1][j+i-1]+$d_j d_{k+1} d_{i+j}$

- Basically, we're checking different places to "split" our matrices by checking different values of k and seeing if they improve our current minimum value.