CSCI 115 Lab

Week 12- Dynamic Programming_1

- Professor: Dr. Matin Pirouz
  Email: mpirouz@csufresno.edu

- TA: Shreeja Miyyar
  Email:shreejarao12@mail.fresnostate.edu

Table of Contents

Dynamic Programming

- Break up a problem into a series of overlapping sub-problems and build up solutions to larger and larger sub-problems.

- Unlike divide and conquer, sub-problems are not independent; Sub-problems may share sub-sub-problems.

- We solve the problem by solving sub-problems of increasing size and saving each optimal solution in a table (usually).

- The table is then used for finding the optimal solution to larger problems. Time is saved since each sub-problem is solved only once.

- Common Examples of Dynamic Programming:
    1. Matrix Chain Multiplication
    2. Longest Common Subsequence
    3. Assembly Line Scheduling

Elements of Dynamic Programming

DP is used to solve problems with the following characteristics:

1. Simple subproblems

   -We should be able to break the original problem into smaller subproblems that have same structure.

2. Optimal sub structure of the problems

   -The optimal solution to the problem contains within it an optimal solution to its subproblems.

3. Overlapping subproblems

   -There exist some places where we can solve the same sub problem more than once.

Matrix Chain Multiplication

The product **C=AB** of a *pxq* matrix *A* and a *qxr* matrix *B* is a *pxr* matrix given by :

$$c[i,j] = \sum_{k=1}^{q} a[i,k]b[k,j]$$

for 1<=*i*<=p and 1<=*j*<=r.

$$A = \begin{bmatrix} 1 & 8 & 9 \\ 7 & 6 & -1 \\ 5 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 8 \\ 7 & 6 \\ 5 & 5 \end{bmatrix},$$

then

$$C = AB = \begin{bmatrix} 102 & 101 \\ 44 & 87 \\ 70 & 100 \end{bmatrix}.$$

- Matrix multiplication is associative (parenthesization does not change result)

  i.e., A1A2A3=(A1A2)A3 = A1(A2A3)

- The order in which we multiply the matrices has a significant impact on the cost of evaluating the product.

  Example: Given matrix A(pxq),B(qxr),C(rxs), then ABC can be computed as (AB)C and A(BC).

  When p=5,q=4 ,r=6 and s=2, then cost of evaluating this product:

A:5x4 B:4x6 C:6x2

1. ((A.B).C)

   A.B = 5x4x6=120 (5x6)

   ((A.B).C) = 5x6x2=60

   Total 180 scalar multiplications

2. (A.(B.C))

   B.C = 4x6x2 = 48 (4x2)

   (A.(B.C)) = 5x4x2 =40

   Total 88 scalar multiplications.

A big difference!

Problem Statement:

Given a chain of matrices <A1,A2,...An> where Ai has dimensions $p_{i-1}xp_i$, fully parenthesize the product A1.A2...An in a way that minimizes the number of scalar multiplications.

A1 · A2 · · · · Ai · Ai+1 · · · An

p0 x p1 p1 x p2 pi-1 x pi pi x pi+1 pn-1 x pn

- Exhaustively checking all possible parenthesizations is not efficient!
- It can be shown that the number of parenthesizations grows as $\Omega(4^n/n^{3/2})$.
- Solution->DP!

1. **The Structure of an Optimal Parenthesization**

Suppose that an optimal parenthesization of Ai...j splits the product between Ak and Ak+1 , where i≤k<j

$A_{i..j} = A_i A_{i+1} \ldots A_j$

$= A_i A_{i+1} \ldots A_k A_{k+1} \ldots A_j$

$= A_{i..k} A_{k+1...j}$

2. Recursive Solution

- Consider all possible ways to split Ai through Aj into two pieces.
- Compare the costs of all these splits:
  - best case cost for computing the product of the two pieces
  - plus the cost of multiplying the two products
- Take the best one

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_k p_j\} & \text{if } i < j \end{cases}$$

m[i, j]= the minimum # of multiplications needed to compute $A_{i...j}$

m[i,k] -> min # of multiplications to compute $A_{i...k}$

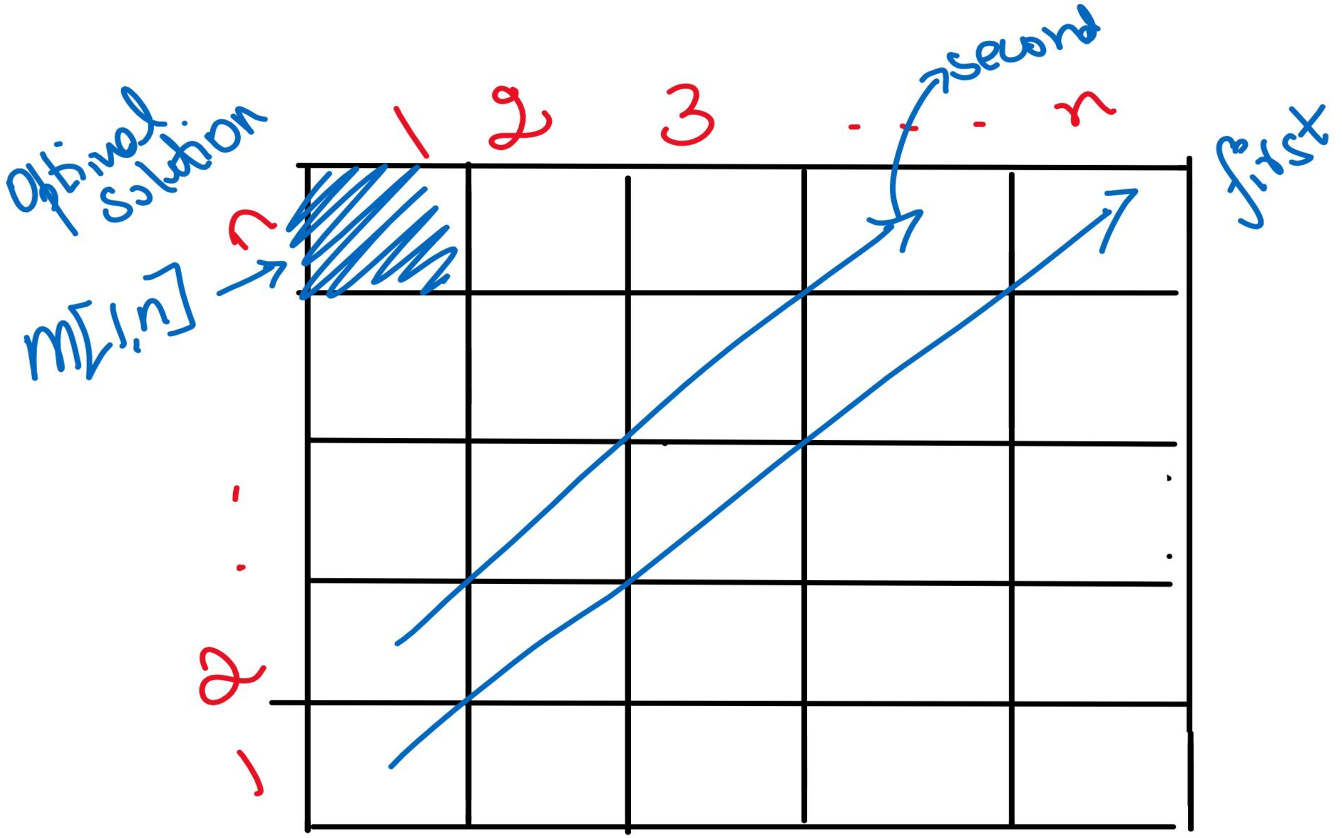
m[k+1,j] -> min # of multiplications to compute $A_{k+1...j}$

$p_{i-1}p_k p_j$ -># of multiplications to compute $A_{i...k}A_{k...j}$

3. Computing Optimal Cost

Computing the optimal solution recursively takes exponential time!

Therefore, use a table by computing rows from bottom to top and from left to right.



4. Construct Optimal Solution

s[i, j]= value of k such that the optimal parenthesization of AiAi+1 · · · Aj splits the product between Ak and Ak+1.

j \ i	1	2	3	4	5	6
6	3	3	3	5	5	-
5	3	3	3	4	-	
4	3	3	3	-		
3	1	2	-			
2	1	-				
1	-					

- $s[1, n] = 3 \Rightarrow A_{1..6} = A_{1..3}\ A_{4..6}$
- $s[1, 3] = 1 \Rightarrow A_{1..3} = A_{1..1}\ A_{2..3}$
- $s[4, 6] = 5 \Rightarrow A_{4..6} = A_{4..5}\ A_{6..6}$

Algorithm

Time Complexity: O(n^3)

```
matrix_chain_order(p, n){
        //Initialise matrix m and s
        for i<- 1 to n
           do m[i,i] <- 0
        for l<- 2 to n
              for i <- 1 to n-l+1
                do  j<- i+l-1
                 m[i,j]<- Infinity
                for k <- i to j-1
                    do q<-m[i,k]+m[k+1,j]+ p[i-1]*p[k]*p[j]
                    if q<m[i,j]
                       then m[i,j]<-q
                               s[i,j]<-k
        name <- 'A'
        print_opt_parens(s,1,n, name);
        cout << "\nOptimal Cost is : " << m[1][n ];
}
```

```
print_opt_parens(&s,i,j,&name){
if i==j
        then print name++
        return;
print "("
print_opt_parens(s,i,s[i,j],name)
print_opt_parens(s,s[i,j]+1,j,name)
print ")"
}
```

Lab Assignment

<u>Hints and Coding Guidelines:</u>

- Create a function to print the parenthesis using the algorithm in previous slide
- Create a matrix chain order function that takes the array and size of array as parameters.
- Create two matrix, one for cost and the other for parenthesis.
- Use the same logic as show in previous slide.
- Print the parenthesization and optimal cost.
- In main function, create an array which represents the layout of matrix , calculate n(size) and call the matrix chain order function.

For example, {10, 20, 30, 40, 30} -> There are 4 matrices of dimensions 10x20, 20x30, 30x40 and 40x30. You can keep the names of matrices as A, B, C and D.

Questions?