K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment No. 7

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation Matrix Chain Multiplication of Dynamic Programming

CO to be achieved:

Sr. No	Objective
CO 1	Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations.
CO 2	Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.
CO 3	Analyze and solve problems for different string matching algorithms.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf**
4. **http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/**
5. **http://www.mafy.lut.fi/study/DiscreteOpt/tspdp.pdf**
6. **https://class.coursera.org/algo2-2012-001/lecture/181**

7. **http://www.quora.com/Algorithms/How-do-I-solve-the-travelling-salesman-problem-using-Dynamic-programming**
8. **www.cse.hcmut.edu.vn/~dtanh/download/Appendix_B_2.ppt**
9. **www.ms.unimelb.edu.au/~s620261/powerpoint/chapter9_4.ppt**

Pre Lab/ Prior Concepts:
Data structures, Concepts of algorithm analysis

Historical Profile:
Dynamic Programming (DP) is used heavily in optimization problems (finding the maximum and the minimum of something). Applications range from financial models and operation research to biology and basic algorithm research. So the good news is that understanding DP is profitable. However, the bad news is that DP is not an algorithm or a data structure that you can memorize. It is a powerful algorithmic design technique.

New Concepts to be learned:

Application of algorithmic design strategy to any problem, dynamic Programming method of problem solving Vs other methods of problem solving, optimality of the solution, Optimal Binary Search Tree Problems and their applications

Theory:

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

We have many options to multiply a chain of matrices because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same. For example, if we had four matrices A, B, C, and D, we would have: A(BCD), (AB)(CD), etc. However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency. Given an array p[] which represents the chain of matrices such that the ith matrix Ai is of dimension p[i-1] x p[i]. We need to write a function MatrixChainOrder() that should return the minimum number of multiplications needed to multiply the chain.

Algorithm:

The idea is to break the problem into a set of related subproblems which group the given matrix in such a way that yields the lowest total cost. Below is a recursive algorithm to find minimum cost:

1. Take the sequence of matrices and separate into two subsequences.
2. Find the minimum cost of multiplying out each susequence.
3. Add these costs together and then add the cost of muliplying the two resultant matrices.
4. Perform the same for each possible position at which the sequence of matrices can be split, and take the minimum over all of them.

Example :

Four matrices A,B,C and D can be multiplied as:

((AB)C)D = ((A(BC))D = (AB)(CD) = A((BC)D) = A(B(CD))

A: 10 x 30, B: 30 x 5, C: 5 x 60

(AB)C needs (10x30x5) + (10x5x60)=4500 operations.

A(BC) needs (30x5x60) + (10x3x60)=27000 operations.

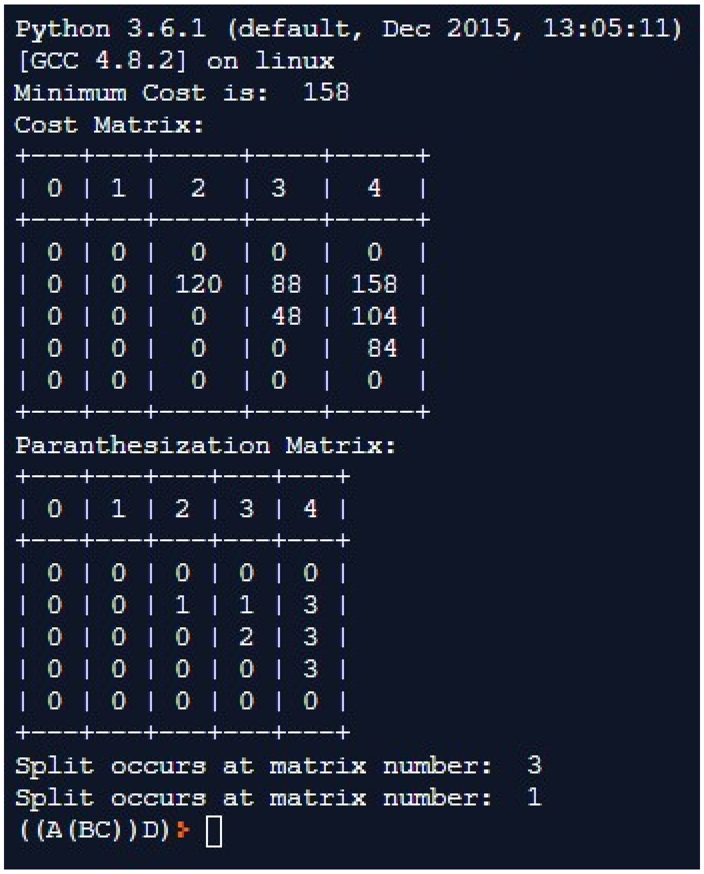
First method is more efficient.

Program:

```
from prettytable import PrettyTable
pr1,pr2=PrettyTable(), PrettyTable()
p,m,b,j,minimum,q,n, name=[5,4,6,2,7], [[0 for i in range(5)] for j in range
(5)], [[0 for i in range(5)] for j in range (5)],0,0,0,5, "A"
for d in range(1,n-1):
  for i in range(1,n-d):
    j=i+d
    minimum=float('inf')
    for k in range(i,j):
      q=m[i][k]+m[k+1][j]+(p[i-1]*p[k]*p[j])
      if q<minimum:
        minimum,b[i][j]=q,k
    m[i][j]=minimum
print("Minimum Cost is: ",m[1][n-1])
print("Cost Matrix: ")
pr1.field_names=[i for i in range(5)]
for row in m:
    pr1.add_row(row)
print (pr1.get_string(header=True, border=True))
print("Paranthesization Matrix: ")
pr2.field_names=[i for i in range(5)]
for row in b:
    pr2.add_row(row)
print (pr2.get_string(header=True, border=True))
for i in range(len(b[1])-1,2,-1):
  print('Split occurs at matrix number: ', b[1][i])
def Parenthesize(i,j,n,b):
  global name
  if i==j:
    print(chr(ord(name)),end='')
    name=chr(ord(name)+1)
```

```
        return
      print("(",end='')
      Parenthesize(i, b[i][j], n,b)
      Parenthesize(b[i][j] + 1, j,n, b)
      print(")",end='')
Parenthesize(1,n-1,n,b)
```

Output Screen:



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Minimum Cost is:  158
Cost Matrix:
+---+---+-----+----+-----+
| 0 | 1 |  2  | 3  |  4  |
+---+---+-----+----+-----+
| 0 | 0 |  0  | 0  |  0  |
| 0 | 0 | 120 | 88 | 158 |
| 0 | 0 |  0  | 48 | 104 |
| 0 | 0 |  0  | 0  |  84 |
| 0 | 0 |  0  | 0  |  0  |
+---+---+-----+----+-----+
Paranthesization Matrix:
+---+---+---+---+---+
| 0 | 1 | 2 | 3 | 4 |
+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 0 | 0 | 2 | 3 |
| 0 | 0 | 0 | 0 | 3 |
| 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+
Split occurs at matrix number:  3
Split occurs at matrix number:  1
((A(BC))D)
```

Analysis of algorithm:

Since we are using three nested for loops, then for large number of inputs, the time complexity of program will be O(n^3). The auxillary space complexity is O(n^2).

CONCLUSION: The program ran successfully for all test cases and the parenthesization was done accordingly.