Name: Vivek Vikram Pundkar

Roll no: 77

GR no: 11910860

Division: C

Batch: 3

--------------------------------------------------------------------------------------------------------------

**Matrix Chain Multiplication**

**Algorithm:**

Firstly we define the formula used to find the value of each cell. Matrix[i,j] equals the minimum cost for computing the sub-products A(i…k) and A(k+1…j), plus the cost of multiplying these two matrices together.

The basic algorithm of matrix chain multiplication:-

// Matrix A[i] has dimension dims[i-1] x dims[i] for i = 1..n

MatrixChainMultiplication(int dims[])

{

// length[dims] = n + 1

n = dims.length - 1;

// m[i,j] = Minimum number of scalar multiplications(i.e., cost)

// needed to compute the matrix A[i]A[i+1]...A[j] = A[i..j]

// The cost is zero when multiplying one matrix

for (i = 1; i <= n; i++)

m[i, i] = 0;

for (len = 2; len <= n; len++){

// Subsequence lengths

for (i = 1; i <= n - len + 1; i++) {

j = i + len - 1;

m[i, j] = MAXINT;

for (k = i; k <= j - 1; k++) {

cost = m[i, k] + m[k+1, j] + dims[i-1]\*dims[k]\*dims[j];

if (cost < m[i, j]) {

m[i, j] = cost;

s[i, j] = k;

// Index of the subsequence split that achieved minimal cost

}}}}}

**Steps:**

Step:1 Create a dp matrix and set all values with a big value(INFINITY).

Step:2 for i in range 1 to N-1:

dp[i][i]=0.

Step:3 for i in range 2 to N-1:

for j in range 1 to N-i+1:

ran=i+j-1.

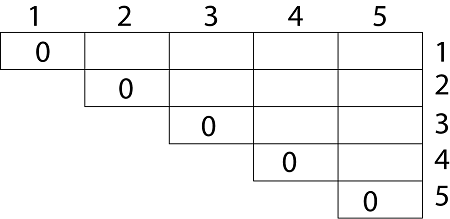
for k in range i to j:

dp[j][ran]=min(dp[j][ran],dp[j][k]+dp[k+1][ran]+v[j-1]\*v[k]\*v[ran]).

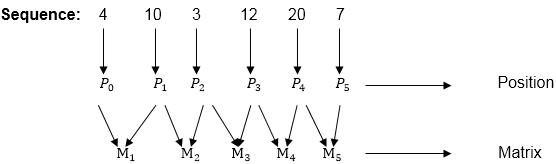
Step:4 Print dp[1][N-1].

**Example:**

We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.



Let us proceed with working away from the diagonal. We compute the optimal solution for the product of 2 matrices.



In Dynamic Programming, initialization of every method done by ‘0’.So we initialize it by ‘0’.It will sort out diagonally.

We have to sort out all the combination but the minimum output combination is taken into consideration.

Calculation of Product of 2 matrices:

1. m (1,2) = m1 x m2

= 4 x 10 x 10 x 3

= 4 x 10 x 3 = 120

2. m (2, 3) = m2 x m3

= 10 x 3 x 3 x 12

= 10 x 3 x 12 = 360

3. m (3, 4) = m3 x m4

= 3 x 12 x 12 x 20

= 3 x 12 x 20 = 720

4. m (4,5) = m4 x m5

= 12 x 20 x 20 x 7

= 12 x 20 x 7 = 1680

**Analysis:**

Time Complexity for Matrix Chain Multiplication:

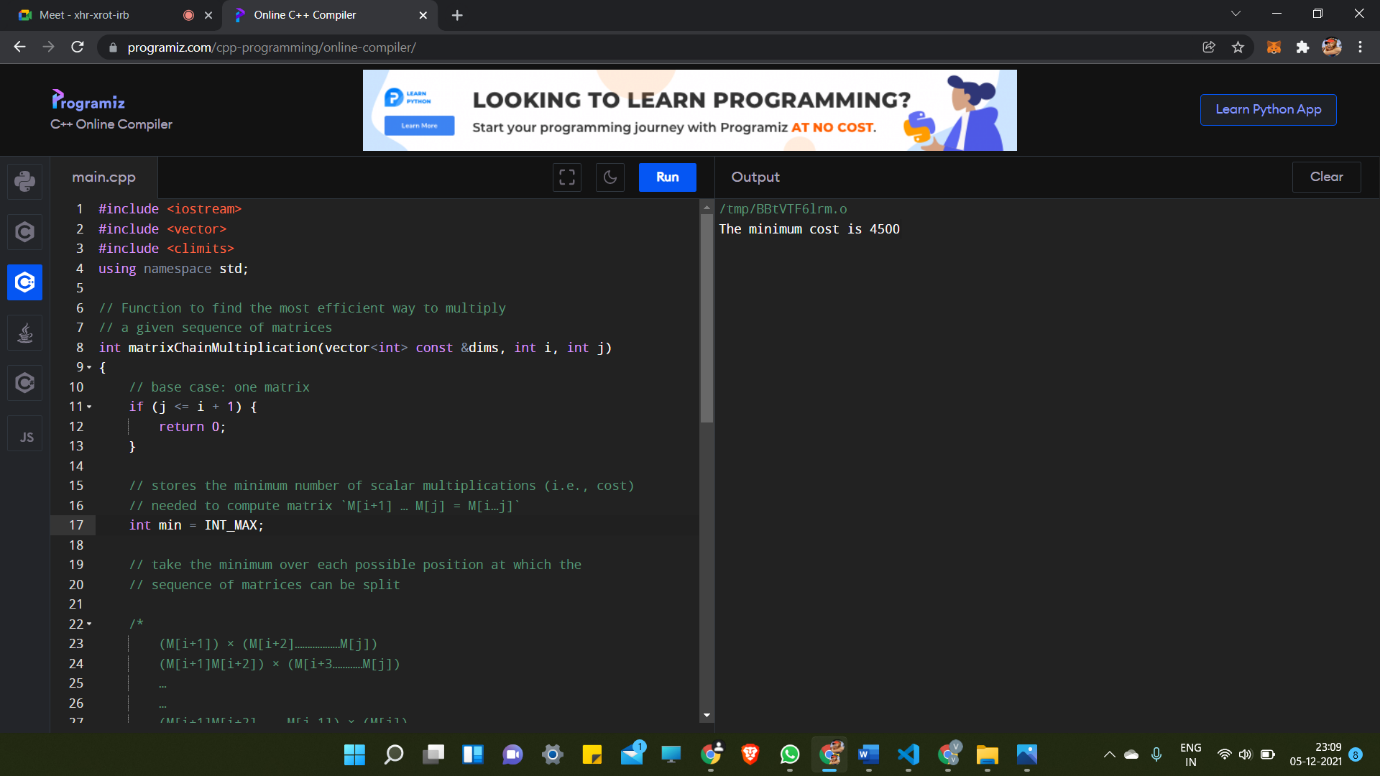
Time complexity = O(N\*N\*N) where N is the number present in the chain of the matrices.

Space Complexity:

We create a DP matrix that stores the results after each operation.

Space complexity = O(N\*N) where N is the number present in the chain of the matrices.

**OUTPUT:**



**Optimal Cost Binary Search Trees**

--------------------------------------------------------------------------------------------------------------

**Title:** Take an integer array and then count no of inversions in that array

**Algorithm:**

Here, the Optimal Binary Search Tree Algorithm is presented. First, we build a BST from a set of provided n number of distinct keys < k1, k2, k3, ... kn >. Here we assume, the probability of accessing a key Ki is pi. Some dummy keys (d0, d1, d2, ... dn) are added as some searches may be performed for the values which are not present in the Key set K. We assume, for each dummy key di probability of access is qi.

**Optimal-Binary-Search-Tree(p, q, n)**

e[1…n + 1, 0…n],

w[1…n + 1, 0…n],

root[1…n + 1, 0…n]

for i = 1 to n + 1 do

e[i, i - 1] := qi - 1

w[i, i - 1] := qi - 1

for l = 1 to n do

for i = 1 to n – l + 1 do

j = i + l – 1 e[i, j] := ∞

w[i, i] := w[i, i -1] + pj + qj

for r = i to j do

t := e[i, r - 1] + e[r + 1, j] + w[i, j]

if t < e[i, j]

e[i, j] := t

root[i, j] := r

return e and root

**Steps:**

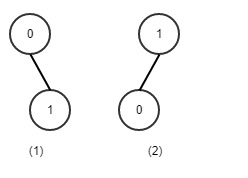
* Split our array into two halves just like merge sort
* Recursion will continue until we get one element.
* counting the number of inversions in the first half, second half and the number of inversions during the merge process.
* Use two pointers, i and j.  i will point to the starting element of the left half and j will point to the starting element of the second half. compare the elements at both the positions.

If ith element is smaller than jth element, add it to the new sorted list. Else, increment the count of inversions by (mid-i).

**Example:**

Keys: {0 ,1} and Freq: {10, 20}

Possible BSTs created from this set of keys are:



1) Total cost of BST = (level of key0 \* freq of key0) +

(level of key1 \* freq of key1)

= (1 \* 10) + (2 \* 20)

= 50

2) Total cost of BST = (level of key1 \* freq of key1) +

(level of key0 \* freq of key0)

= (1 \* 20) + (2 \* 10)

= 40

Hence, the minimum total searching cost for given set of keys is 40.

**Analysis:**

The algorithm requires O (n3) time, since three nested for loops are used. Each of these loops takes on at most n values.

**Output:**

