CHAITRA SAMANT

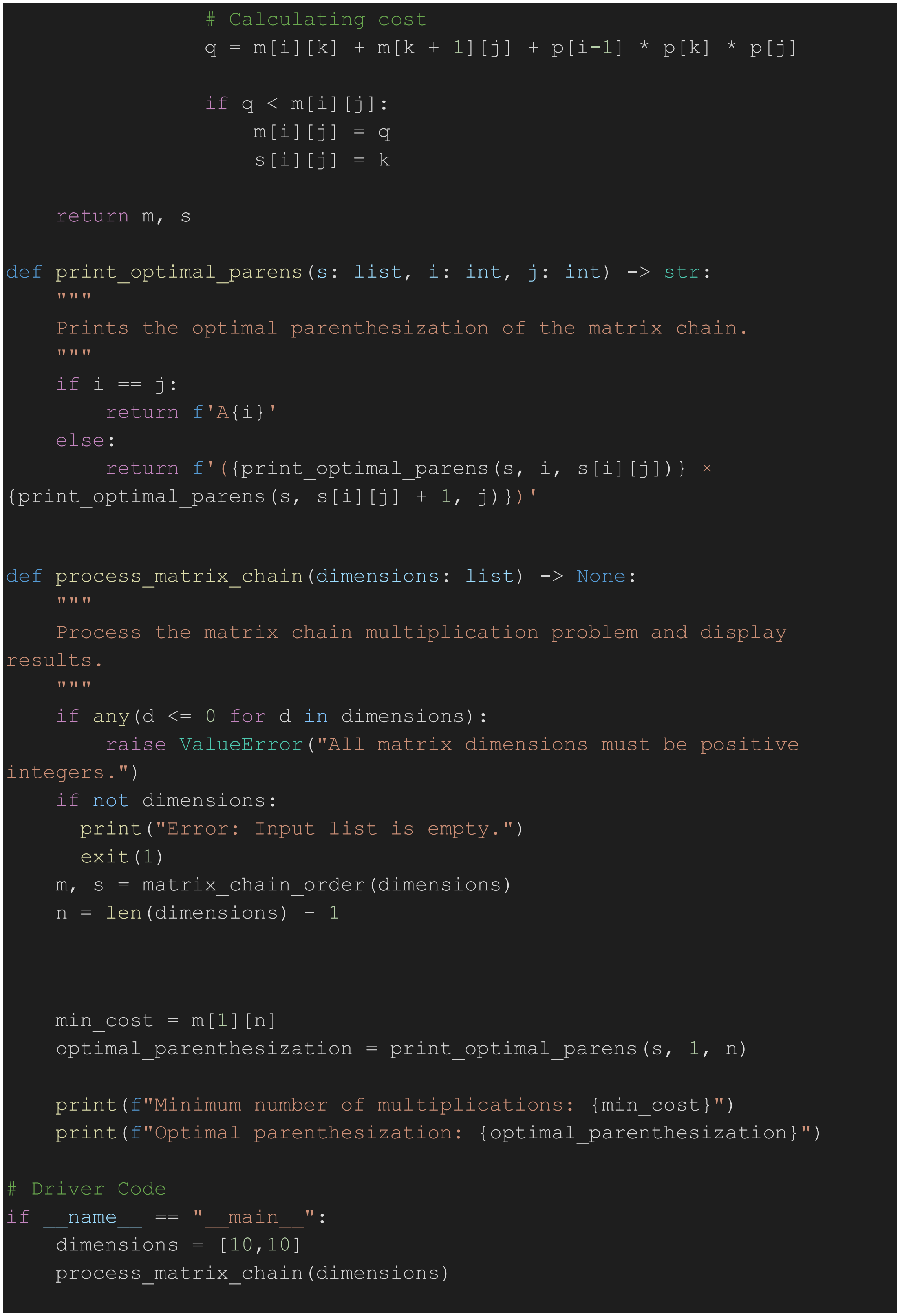
231070055

DAA LAB 6B

1. **AIM:**
   Consider meteorological data like **temperature, dew point, wind direction, wind speed, cloud cover, cloud layer(s)** for each city. This data is available in two dimensional array for a week. Assuming all tables are compatible for multiplication. You have to implement the matrix chain multiplication algorithm to find fastest way to complete the matrices multiplication to achieve timely predication.

2. **PROGRAM:**

```
def matrix_chain_order(p: list) -> tuple:
    """
    Implements the Matrix-Chain-Order algorithm as shown in the image
    using bottom-up dynamic programming.

    Args:
        p: List of dimensions where matrix i has dimensions p[i-1] x
p[i]

    Returns:
        Tuple of (m, s) tables where:
        m: Table containing optimal costs
        s: Table containing optimal split points
    """
    n = len(p) - 1

    # Initialize Memo Table
    m = [[float('inf')] * (n + 1) for _ in range(n + 1)]
    s = [[0] * (n + 1) for _ in range(n + 1)]


    for i in range(1, n + 1):
        m[i][i] = 0

    # l is the chain length
    for l in range(2, n + 1):

        for i in range(1, n - l + 2):
            j = i + l - 1
            for k in range(i, j):
```

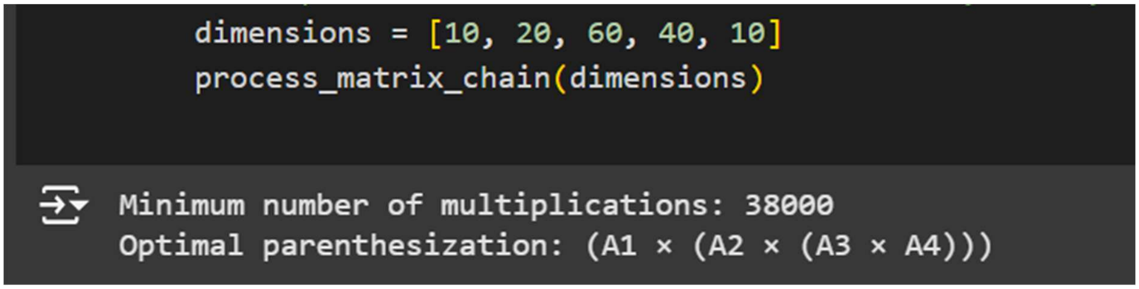


```
                # Calculating cost
                q = m[i][k] + m[k + 1][j] + p[i-1] * p[k] * p[j]

                if q < m[i][j]:
                    m[i][j] = q
                    s[i][j] = k

    return m, s

def print_optimal_parens(s: list, i: int, j: int) -> str:
    """
    Prints the optimal parenthesization of the matrix chain.
    """
    if i == j:
        return f'A{i}'
    else:
        return f'({print_optimal_parens(s, i, s[i][j])} ×
{print_optimal_parens(s, s[i][j] + 1, j)})'


def process_matrix_chain(dimensions: list) -> None:
    """
    Process the matrix chain multiplication problem and display
results.
    """
    if any(d <= 0 for d in dimensions):
        raise ValueError("All matrix dimensions must be positive
integers.")
    if not dimensions:
      print("Error: Input list is empty.")
      exit(1)
    m, s = matrix_chain_order(dimensions)
    n = len(dimensions) - 1



    min_cost = m[1][n]
    optimal_parenthesization = print_optimal_parens(s, 1, n)

    print(f"Minimum number of multiplications: {min_cost}")
    print(f"Optimal parenthesization: {optimal_parenthesization}")

# Driver Code
if __name__ == "__main__":
    dimensions = [10,10]
    process_matrix_chain(dimensions)
```

3. **TESTCASES:**

POSITIVE

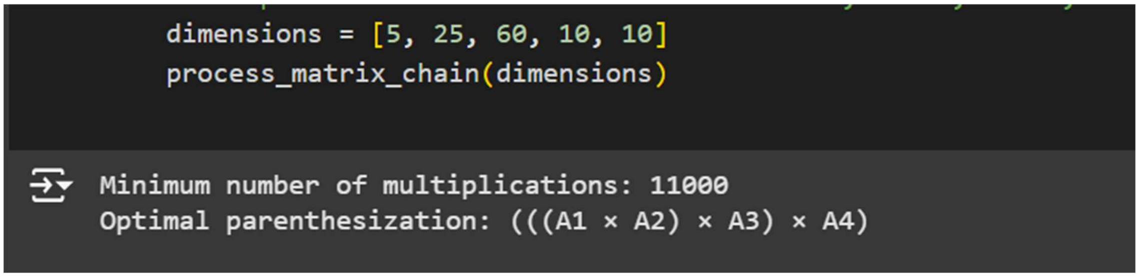
- Valid TC 1



```
dimensions = [10, 20, 60, 40, 10]
process_matrix_chain(dimensions)
```

```
Minimum number of multiplications: 38000
Optimal parenthesization: (A1 × (A2 × (A3 × A4)))
```

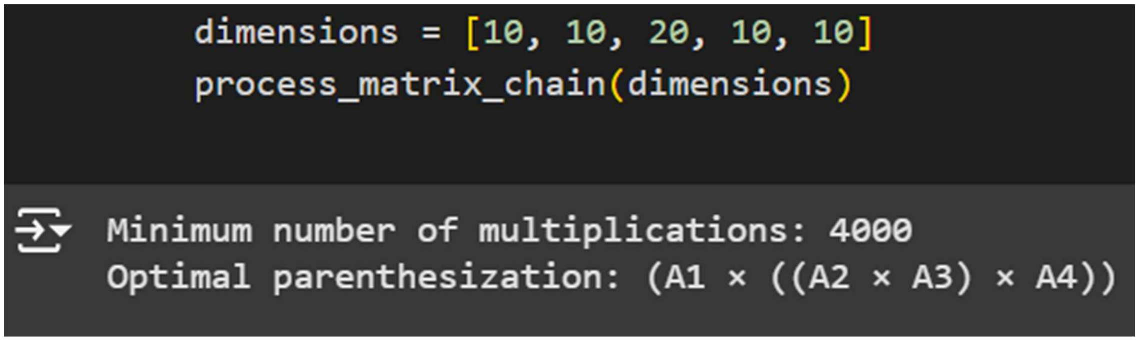
- Valid TC 2



```
dimensions = [5, 25, 60, 10, 10]
process_matrix_chain(dimensions)
```

```
Minimum number of multiplications: 11000
Optimal parenthesization: (((A1 × A2) × A3) × A4)
```

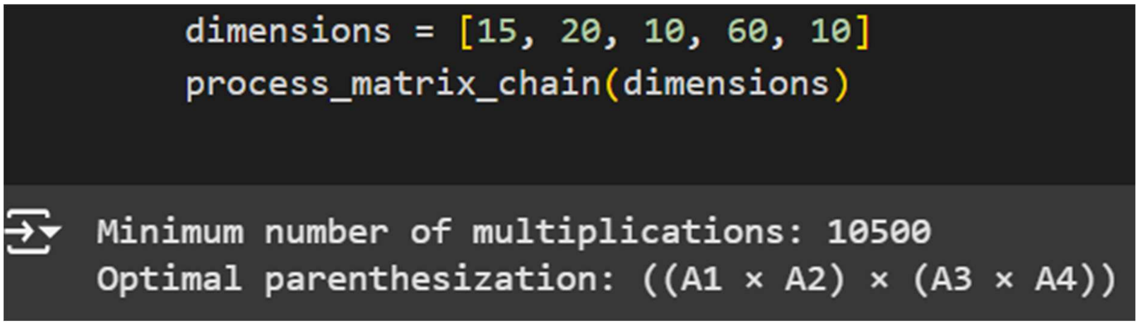
- Valid TC 3



```
dimensions = [10, 10, 20, 10, 10]
process_matrix_chain(dimensions)
```

```
Minimum number of multiplications: 4000
Optimal parenthesization: (A1 × ((A2 × A3) × A4))
```

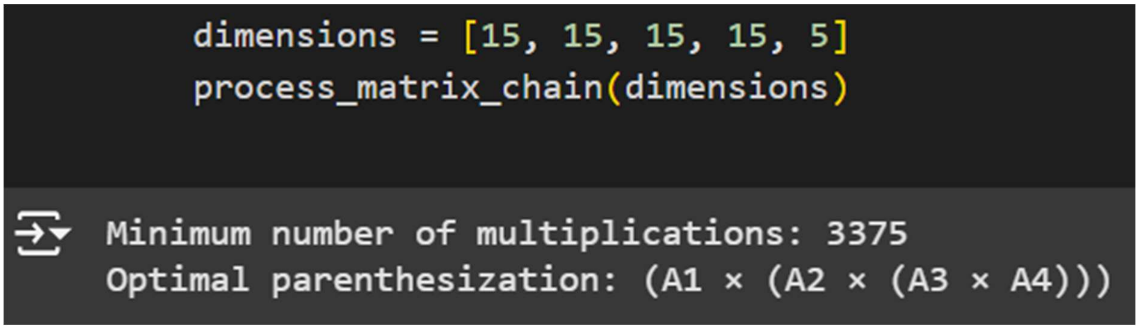
- Valid TC 4



```
dimensions = [15, 20, 10, 60, 10]
process_matrix_chain(dimensions)
```

```
Minimum number of multiplications: 10500
Optimal parenthesization: ((A1 × A2) × (A3 × A4))
```

- Valid TC 5

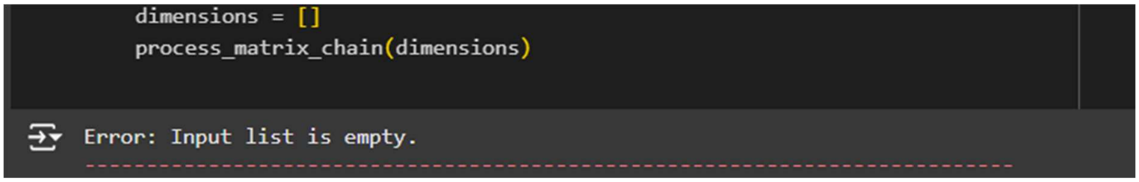


```
dimensions = [15, 15, 15, 15, 5]
process_matrix_chain(dimensions)
```

```
Minimum number of multiplications: 3375
Optimal parenthesization: (A1 × (A2 × (A3 × A4)))
```

NEGATIVE

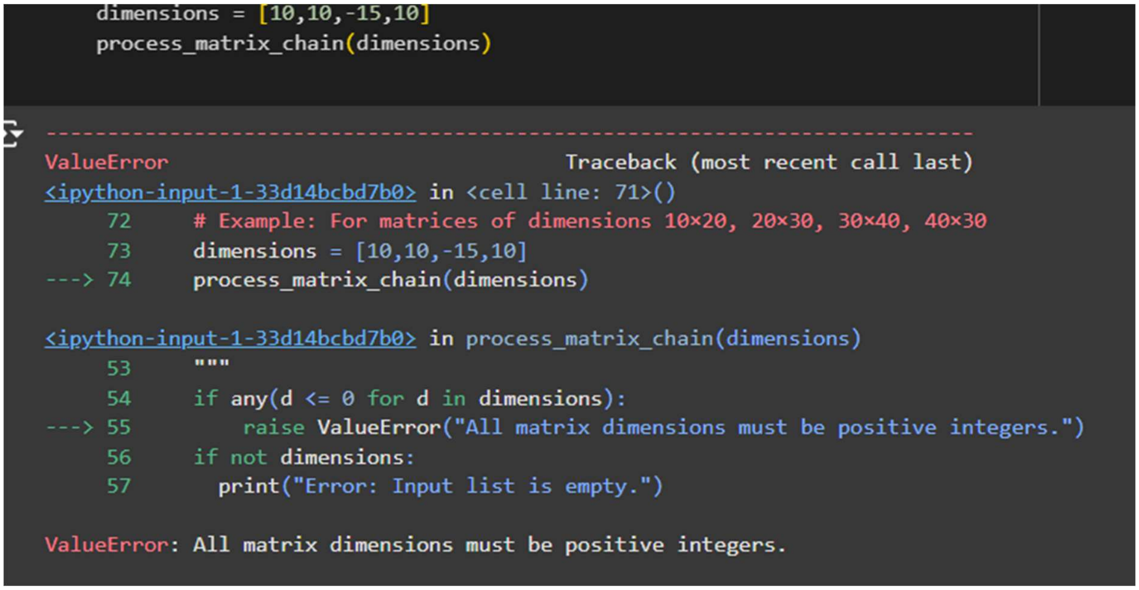
- Empty Input list



```
dimensions = []
process_matrix_chain(dimensions)
```

```
Error: Input list is empty.
---------------------------------------------------------------------------
```

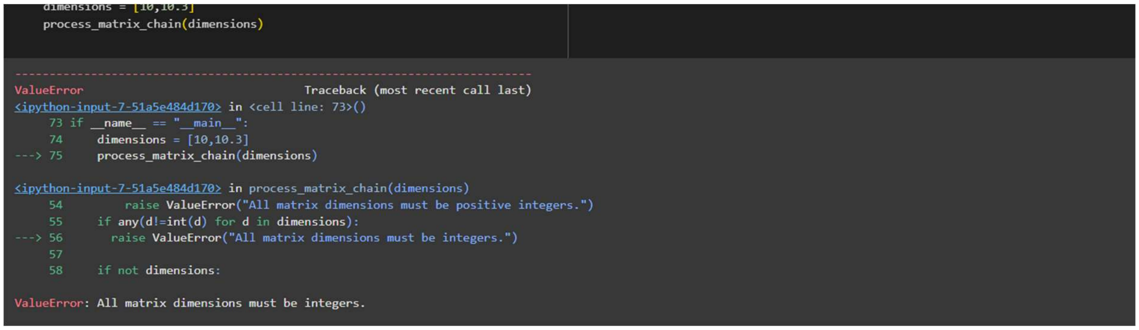
- Negative Dimensions



```
dimensions = [10,10,-15,10]
process_matrix_chain(dimensions)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1-33d14bcbd7b0> in <cell line: 71>()
     72     # Example: For matrices of dimensions 10×20, 20×30, 30×40, 40×30
     73     dimensions = [10,10,-15,10]
---> 74     process_matrix_chain(dimensions)

<ipython-input-1-33d14bcbd7b0> in process_matrix_chain(dimensions)
     53     """
     54     if any(d <= 0 for d in dimensions):
---> 55         raise ValueError("All matrix dimensions must be positive integers.")
     56     if not dimensions:
     57       print("Error: Input list is empty.")

ValueError: All matrix dimensions must be positive integers.
```

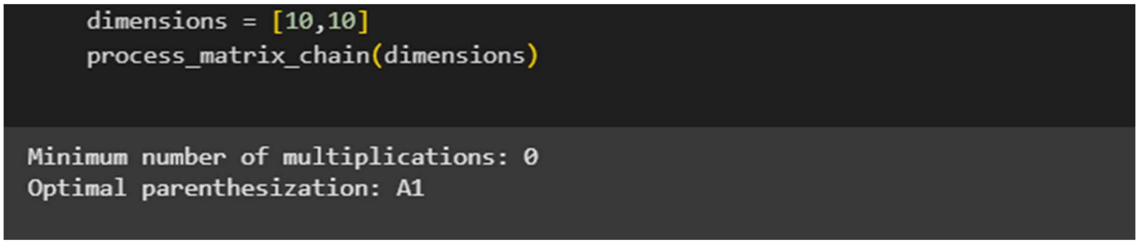
- Decimal Input of dimensions



```
dimensions = [10,10.3]
process_matrix_chain(dimensions)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-51a5e484d170> in <cell line: 73>()
     73 if __name__ == "__main__":
     74     dimensions = [10,10.3]
---> 75     process_matrix_chain(dimensions)

<ipython-input-7-51a5e484d170> in process_matrix_chain(dimensions)
     54         raise ValueError("All matrix dimensions must be positive integers.")
     55     if any(d!=int(d) for d in dimensions):
---> 56       raise ValueError("All matrix dimensions must be integers.")
     57
     58     if not dimensions:

ValueError: All matrix dimensions must be integers.
```

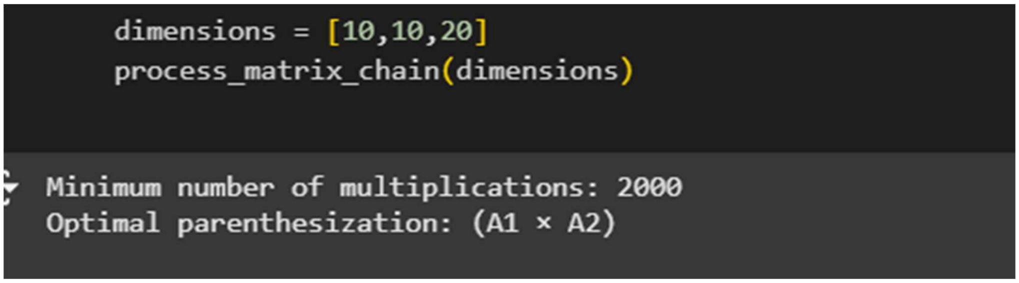
- Only one matrix



```
dimensions = [10,10]
process_matrix_chain(dimensions)
```

```
Minimum number of multiplications: 0
Optimal parenthesization: A1
```

- Two matrices



```
dimensions = [10,10,20]
process_matrix_chain(dimensions)
```

```
Minimum number of multiplications: 2000
Optimal parenthesization: (A1 × A2)
```

4. CONCLUSION

Hence, we implemented Matrix Chain Multiplication for matrices having meteorological data. We assumed all matrices are compatible and found out the number of operations as well as the optimal parenthesization of the matrices to have the most efficient multiplication. We used Dynamic Programming technique known as memorization for this problem.