**Question 1:**

1. **Pseudo Code:**

Function readFile(filename, n, coordinates, prices)

Open file with filename

If file is open

Read first line to get n

For each remaining line in file

Parse and store coordinates and prices in given structures

Close file

Else

Print error message

Function coutData(n, coordinates, prices)

Print n

For each item in coordinates and prices

Print them

Function maxProfit(n, coordinates, prices)

Create a DP array of size n+1, initialize with 0

For each size from 1 to n

For each item in coordinates

Calculate area and update DP if profitable

Return value at DP[n]

Function maxProfitmemo(n, coordinates, prices)

If n is 0, return 0

If result is memoized, return it

For each item in coordinates

Calculate area and recursively call function

Update max profit

Store result in memo and return it

Function maxProfitmemo2(n, coordinates, prices)

Initialize memo1 if empty

If result is memoized, return it

For each item in coordinates

Calculate area and recursively call function

Update max profit

Store result in memo1 and return it

Function main

Read file and store data

Print data

Calculate max profit without memoization

Calculate max profit with memoization

Calculate max profit with improved memoization

1. **Time Complexity**

* maxProfit`: O(n \* m \* k), where n is the target area, m is the number of items, and k is the average number of coordinates per item.
* `maxProfitmemo` and `maxProfitmemo2`: O(n \* m \* k) in the worst case, but typically faster due to memoization.

1. **Algorithm Explanation**

* **File Reading**: The function `readFile` parses a file containing the number of items (n), followed by lines with coordinates and prices for each item.
* **Printing Data**: The function `coutData` simply prints the data read from the file.
* **Dynamic Programming (DP) Approach**: In `maxProfit`, a DP array is used to store the maximum profit for each area up to n. It iterates through each item, updating the DP array if a combination of items yields a higher profit.
* **Memoization:** `maxProfitmemo` and `maxProfitmemo2` use memoization to store previously computed results for specific areas, reducing redundant calculations.

1. **Dry Run Example**

Consider a simple file with the following content:

2

(3,4),{100}

(5,2),{150}

* `readFile` will parse this as `n=2`, coordinates as `[(3,4), (5,2)]`, and prices as `[100, 150]`.
* `coutData` will print `2`, then `"(3,4),{100}"` and `"(5,2),{150}"`.
* `maxProfit`, `maxProfitmemo`, and `maxProfitmemo2` will calculate the maximum profit for the given coordinates and prices. For instance, if `n=10`, the algorithm will look for the best combination of items (based on their area and price) that does not exceed an area of 10. The DP and memoization functions will optimize this process by storing intermediate results.

**Question 2:**

**Pseudo Code**

Function readTestCase(fileName)

Open file with fileName

Read matrix size, rows, and columns

Read matrix elements

Read pattern

Read expected result

Return testCase with read values

Function printMatrix(matrix)

For each row in matrix

Print all elements in the row

Function printTestCase(testCase)

Print matrix size, matrix, pattern, and expected result

Function convertPatternToMatrix(pattern)

Calculate matrix size based on pattern length

Create matrix and fill it with characters from pattern

Return the matrix

Function findDiagonalPattern(matrix, pattern)

Initialize count to 0

For each possible starting position in matrix

Check if pattern matches at this position

If match found, increment count

If count is at least 2, return count, else return 0

Function checkResult(testCase, actualResult)

Compare actualResult with testCase.expectedResult

Print test case status and results

Function main

Read and print the test case

Convert pattern to matrix

Find actual result using findDiagonalPattern

Check and print result

**Time Complexity**

* **`findDiagonalPattern**`: O((n-m)^2 \* m^2), where n is the size of the matrix and m is the size of the pattern matrix.

**Algorithm Explanation**

* **Reading Test Case:** The function `readTestCase` reads the input file and extracts the matrix, pattern, and expected result.
* **Printing Test Case:** `printTestCase` prints the details of the test case for verification.
* Converting Pattern to Matrix: `convertPatternToMatrix` converts the pattern string into a matrix, facilitating the pattern matching process.
* **Finding Diagonal Pattern:** `findDiagonalPattern` checks for at least two occurrences of the pattern in diagonal positions within the main matrix.
* **Checking Result**: `checkResult` compares the actual result with the expected result and prints whether the test passed or failed.

**Dry Run Example**

Matrix size: 5x5

Matrix:

A B C A B

F G H F G

K L M N O

P Q R S T

U V W X Y

Pattern: ABFG

Expected result: 2

* **`readTestCase**` reads this data from a file.
* `**printTestCase`** prints these details.
* **`convertPatternToMatrix`** converts `ABFG` into a 2x2 matrix.
* **`findDiagonalPattern**` searches for this 2x2 pattern in the 5x5 matrix. It finds two diagonal occurrences (top-left and middle) of the pattern.
* The function returns 2 as there are two diagonal occurrences.
* `**checkResult`** compares this with the expected result (2) and confirms the test case passes.

**Question 3(a):**

**Pseudo Code:**

Structure TestCase

Define rows, cols, matrix (2D char vector), pattern (string), expectedResult (int)

Function readTestCase(fileName)

Open file with fileName

Read matrix size (rows x cols)

Read matrix elements into 2D char vector

Read pattern string

Read expected result (int)

Return constructed TestCase object

Function printMatrix(matrix)

For each row in matrix

Print each element in row separated by space

Function printTestCase(testCase)

Print rows x cols, matrix, pattern, and expected result from testCase

Function convertPatternToMatrix(pattern)

Calculate size of square matrix to hold pattern

Fill square matrix with characters from pattern

Return the square matrix

Function findDiagonalPattern(matrix, pattern)

Initialize count of pattern occurrences to 0

For each possible starting position in matrix

Check if pattern is found at this position

If pattern found, increment count

Return count if it is at least 2, else return 0

Function checkResult(testCase, actualResult)

Compare actualResult with testCase.expectedResult

Print result and status of test case

Function main

Read and print testCase from file

Convert pattern to matrix form

Find diagonal pattern occurrences in main matrix

Check and print the result against expected result

**Time Complexity:**

* **`findDiagonalPattern`:** O((n-m)^2 \* m^2), where n is the size of the main matrix and m is the size of the pattern matrix.

**Algorithm Explanation:**

* **File Reading**: `readTestCase` reads data from a file, constructing a `TestCase` object with the matrix, pattern, and expected result.
* **Matrix Printing:** `printMatrix` and `printTestCase` are used for visual confirmation of the test case data.
* **Pattern Conversion:** `convertPatternToMatrix` converts the string pattern into a matrix form, making it easier to match against the main matrix.
* **Pattern Matching:** `findDiagonalPattern` searches for at least two diagonal occurrences of the pattern matrix in the main matrix. It iterates over possible start positions in the main matrix, checking if the pattern matrix fits.
* **Result Verification:** `checkResult` compares the actual result of pattern matching with the expected result and prints the outcome.

**Dry Run Example**

Assuming a test case file `test2.txt` with the following content:

```

T(5x5)

A,B,C,A,B

F,G,H,F,G

K,L,M,N,O

P,Q,R,S,T

U,V,W,X,Y

ABFG

2

* `readTestCase` reads and constructs a `TestCase` object with a 5x5 matrix, the pattern `ABFG`, and an expected result of 2.
* `printTestCase` prints the test case data for verification.
* `convertPatternToMatrix` turns `ABFG` into a 2x2 matrix: `[AB][FG]`.
* `findDiagonalPattern` searches for this 2x2 pattern in the 5x5 matrix. It finds two diagonal occurrences (top-left and middle-right) of the pattern.
* The function returns 2 as there are two diagonal occurrences.
* `checkResult` compares this with the expected result (2) and confirms the test case passes.

**Question 3(b):**

**Pseudo Code**

Class GraphData

Properties: numVertices, edgeList, expectedCycleLength, resultingCycle

Constructor(fileName)

Read numVertices, edgeList, expectedCycleLength, and resultingCycle from file

Function printData

Print numVertices

Call printEdges to print all edges

Print expectedCycleLength and resultingCycle

Function printEdges(index)

Recursive function to print all edges

Function computeShortestCycle

Initialize shortestPath matrix

Perform Floyd-Warshall algorithm

Find shortest cycle length

Print test result

Function printTestResult(result)

Print test status based on result and expectedCycleLength

Private Functions:

initializeMatrix(shortestPath, index)

floydWarshall(shortestPath, k)

updateShortestPath(shortestPath, k, i)

updatePath(shortestPath, k, i, j)

findShortestCycle(shortestPath, i, shortestCycle)

Function main

Create GraphData object from file

Print graph data

Compute and print shortest cycle length

**Time Complexity**

* Floyd-Warshall Algorithm: O(n^3), where n is the number of vertices.

**Algorithm Explanation**

**Data Reading:** The constructor of `GraphData` reads the graph data from a file, including the number of vertices, list of edges, expected cycle length, and resulting cycle.

**Data Printing:** `printData` and `printEdges` print the graph's properties for verification.

**Shortest Cycle Computation:**

- Initialize an adjacency matrix with infinite distances except for direct edges.

- Apply the Floyd-Warshall algorithm to find the shortest paths between all pairs of vertices.

- Identify the shortest cycle by finding the minimum distance where a vertex can reach itself.

**Test Verification: `**printTestResult` compares the computed shortest cycle length with the expected value and prints the test status.

**Dry Run Example**

Assume a file `testcase4.txt` with the following content:

4

0 1

1 2

2 3

3 0

3

0-1-2-3-0

* `GraphData` reads this as a graph with 4 vertices, edges (0-1, 1-2, 2-3, 3-0), an expected cycle length of 3, and a resulting cycle "0-1-2-3-0".
* `printData` prints these details.
* `computeShortestCycle` initializes the shortest path matrix and applies the Floyd-Warshall algorithm. It finds the shortest cycle length by checking each vertex's distance to itself.
* In this example, the shortest cycle is 3 (0-1-2-3-0).
* `printTestResult` compares this with the expected result (3) and confirms the test case passes.