**PART A**

(PART A : TO BE REFFERED BY STUDENTS)

**EXPERIMENT NO. 3**

**A.1 AIM: -** To perform Matrix Operation, find minimum cost path, find maximum in an integer array, and array sorting.

**A.2 Prerequisite**

* Different programming language (Python or Java), Understanding of Machine Learning Algorithms, Machine Learning Algorithms

**A.3 Outcome**

After successful completion of this experiment students will be able to understand working of matrix, find minimum and maximum cost paths

**A.4 Theory**

**Min Cost Path**

The minimum cost path problem in Java is one the most prominent problems that have been asked in the interview. In this problem, a matrix is provided (costMatrix[][]), which represents the cost of each of the cells present in the costMatrix[][]. The task is to go from the top left corner to the bottom right corner such that the cost is minimum. We have to return the minimum cost. The rule from going from one cell to another cell is that one can only go in the left or down or the diagonal direction, with one cell at a time. For example, from the current cell, say costMatrix[x][y], we can only go to one of these cells: costMatrix[x][y + 1] (the left direction), costMatrix[x + 1][y] (the downward direction), and costMatrix[x + 1][y + 1] (the diagonal direction).

For example, in the following matrix

Minimum Cost Path Problem in Java

There are the following paths to go from the top-left cell (of the cost 1) to the bottom-right cell (of the cost 7).

1 -> 6 -> 9 -> 5 -> 7 Total Cost = 1 + 6 + 9 + 5 + 7 = 28

1 -> 6 -> 15 -> 5 -> 7 Total Cost = 1 + 6 + 15 + 5 + 7 = 34

1 -> 6 -> 15 -> 3 -> 7 Total Cost = 1 + 6 + 15 + 3 + 7 = 32

1 -> 6 -> 15 -> 7 Total Cost = 1 + 6 + 15 + 7 = 29

1 -> 6 -> 5 -> 7 Total Cost = 1 + 6 + 5 + 7 = 19

1 -> 2 -> 15 -> 3 -> 7 Total Cost = 1 + 2 + 15 + 3 + 7 = 28

1 -> 2 -> 15 -> 5 -> 7 Total Cost = 1 + 2 + 15 + 5 + 7 = 30

1 -> 2 -> 15 -> 7 Total Cost = 1 + 2 + 15 + 7 = 25

1 -> 2 -> 2 -> 3 -> 7 Total Cost = 1 + 2 + 2 + 3 + 7 = 15

1 -> 2 -> 3 -> 7 Total Cost = 1 + 2 + 3 + 7 = 13

In all the above-mentioned paths, the last path (1 -> 2 -> 3 -> 7, total cost: 13) has the minimum cost. Therefore, 13 is the required answer of the above matrix.

**A5. Task**

Perform Following Operations

1.Write a Python program to find out when given an array of positive elements, you have to flip the sign of some of its elements such that the resultant sum of the elements of array should be minimum non-negative (as close to zero as possible). Return the minimum no. of elements whose sign needs to be flipped such that the resultant sum is minimum non-negative. Note that the sum of all the array elements will not exceed 10^4

Input: arr[] = [14, 10, 4]

Output: 1

Here, we will flip the sign of 14 and the resultant sum will be 0. Note that flipping the signs of 10 and 4 also gives the resultant sum 0 but the count of flipped elements is not minimum.

2. Write a Python program to find out when given a two dimensional grid, each cell of which contains integer cost which represents a cost to traverse through that cell. The task is to find the maximum cost path from the bottom-left corner to the top-right corner.

3. Write a Python program to find out when given an array of non-negative integers arr[], the task is to find a pair (n, r) such that nPr is maximum possible and r ≤ n.

Input: arr[] = {5, 2, 3, 4, 1}

Output: n = 5 and r = 4

5P4 = 5! / (5 – 4)! = 120

which is maximum possible. Input: arr[] = {0, 2, 3, 4, 1, 6, 8, 9} Output: n = 9 and r = 8

4. Write a Python program to find out when given an array of non-negative integers arr[], the task is to find a pair (n, r) such that nPr is maximum possible and r ≤ n.

Function to return the minimum number of given operations required to sort the array

Input: arr[] = {1, 2, 1, 4, 3}

Output: 2

Add 1 to the 3rd element(1) and subtract 1 from the 4th element(4) to get {1, 2, 2, 3, 3} Input: arr[] = {1, 2, 2, 100}

Output: 0 Given array is already sorted.

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

|  |  |
| --- | --- |
| Roll No. C027 | Name: Vishesh Giyanani |
| Class : SEM IX | Batch : B1 |
| Date of Experiment: 12/1/2024 | Date of Submission |
| Grade : |  |

**B.1 Documentation written by student:**

1

def minimum\_flips(*array*):

n = len(*array*)

total\_sum = sum(*array*)

matrix = [[float('inf')] \* (total\_sum + 1) for \_ in range(n + 1)]

matrix[0][0] = 0

for i in range(1, n + 1):

for j in range(total\_sum + 1):

if j >= *array*[i - 1]:

matrix[i][j] = min(matrix[i][j], matrix[i - 1][j - *array*[i - 1]])

matrix[i][j] = min(matrix[i][j], matrix[i - 1][j] + 1)

min\_flips = float('inf')

for j in range(total\_sum // 2 + 1):

min\_flips = min(min\_flips, matrix[n][j])

return min\_flips

array = [14, 10, 4]

result = minimum\_flips(array)

print(result)

2

def max\_cost\_path(*grid*):

if not *grid*:

return 0

rows, columns = len(*grid*), len(*grid*[0])

matrix = [[0] \* columns for \_ in range(rows)]

matrix[rows - 1][0] = *grid*[rows - 1][0]

for i in range(rows - 2, -1, -1):

matrix[i][0] = matrix[i + 1][0] + *grid*[i][0]

for j in range(1, columns):

matrix[rows - 1][j] = matrix[rows - 1][j - 1] + *grid*[rows - 1][j]

for i in range(rows - 2, -1, -1):

for j in range(1, columns):

matrix[i][j] = max(matrix[i + 1][j], matrix[i][j - 1]) + *grid*[i][j]

return matrix[0][columns - 1]

grid = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

result = max\_cost\_path(grid)

print(result)

3

def maximum\_nPr\_pair(*arr*):

maximum\_n = -1

maximum\_r = -1

maximum\_nPr = -1

for n in *arr*:

for r in range(n + 1):

nPr = 1

for i in range(r):

nPr \*= (n - i)

if nPr > maximum\_nPr:

maximum\_nPr = nPr

maximum\_n = n

maximum\_r = r

return maximum\_n, maximum\_r

array1 = [5, 2, 3, 4, 1]

n1, r1 = maximum\_nPr\_pair(array1)

print(f"n = {n1} and r = {r1}")

array2 = [0, 2, 3, 4, 1, 6, 8, 9]

n2, r2 = maximum\_nPr\_pair(array2)

print(f"n = {n2} and r = {r2}")

4

def minimum\_operations\_to\_sort(*array*):

array\_length = len(*array*)

sorted\_array = sorted(*array*)

operations = 0

for i in range(array\_length):

diff = abs(*array*[i] - sorted\_array[i])

operations += diff

minimum\_operations = operations // 2

return minimum\_operations

array1 = [1, 2, 1, 4, 3]

result1 = minimum\_operations\_to\_sort(array1)

print(result1)

array2 = [1, 2, 2, 100]

result2 = minimum\_operations\_to\_sort(array2)

print(result2)

**B.2 Observations and learning:**

1. **minimum\_flips** function:

* This function takes an array of non-negative integers as input.
* It calculates the minimum number of flips needed to make the sum of all elements as close to half of the total sum as possible.
* It uses dynamic programming to fill a matrix, where each cell represents the minimum flips needed to reach a particular sum.
* The result is the value in the middle column of the last row of the matrix.

2. **max\_cost\_path** function:

* This function takes a two-dimensional grid as input, where each cell contains an integer cost.
* It calculates the maximum cost path from the bottom-left corner to the top-right corner of the grid.
* It uses dynamic programming to fill a matrix, where each cell represents the maximum cost path sum to reach that cell.
* The result is the value in the top-right corner of the matrix.

3. **maximum\_nPr\_pair** function:

* This function takes an array of non-negative integers as input.
* It finds a pair (n, r) such that nPr is maximum possible and r ≤ n.
* It iterates through all possible pairs (n, r) for each element in the array and calculates nPr for each pair.
* The result is the pair (n, r) with the maximum nPr value.

4. **minimum\_operations\_to\_sort** function:

* This function takes an array of non-negative integers as input.
* It calculates the minimum number of operations required to sort the array.
* It creates a sorted copy of the input array and computes the absolute differences between elements in both arrays.
* The result is half of the sum of these absolute differences, as each operation affects two elements.

**B.3 Conclusion:**

Implementation of minimum flips, maximum cost paths, maximum nPr pairs, and minimum operations to sort arrays.