**PART A**

**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

|  |  |
| --- | --- |
| Roll No. C009 | Name: Samarth Borade |
| Class : BTI SEM 10 | Batch : EB1 |
| Date of Experiment: 12/01/24 | Date of Submission |
| Grade : |  |

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

**Question 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

'''

def min\_sum(*a*):

temp = []

n = len(*a*)

for i in range(0, 2\*\*n): # basically 2^n

comb = []

sum = 0

flips = 0

for j in range(0, n):

if ((i & 2\*\*j) > 0):

sum += *a*[j]

comb.append(*a*[j])

else:

sum = sum + (-1 \* *a*[j])

comb.append(-*a*[j])

flips += 1

temp.append((sum, flips, comb))

temp2 = []

for i in range(len(temp)):

if temp[i][0] >= 0:

temp2.append(temp[i])

#print(temp2)

temp2.sort()

#print(temp2)

print("min sum:", temp2[0][0])

print("min flips:", temp2[0][1])

print("combination ", temp2[0][2])

a = [14, 10, 4]

min\_sum(a)

**Output:  
A blue background with white text

Description automatically generated**

**Question 2:**

def path(*grid*):

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

def generatePaths(*path*, *row*, *col*):

paths = []

if *row* == 0 and *col* == cols - 1:

paths.append(*path*)

return paths

if *row* > 0: #up

paths.extend(generatePaths(*path* + [(*row* - 1, *col*)], *row* - 1, *col*))

if *col* < cols - 1: #right

paths.extend(generatePaths(*path* + [(*row*, *col* + 1)], *row*, *col* + 1))

if *row* > 0 and *col* < cols - 1: #diagonal

paths.extend(generatePaths(*path* + [(*row* - 1, *col* + 1)], *row* - 1, *col* + 1))

return paths

return generatePaths([(rows - 1, 0)], rows - 1, 0)

arr = [

[20, -10, 0],

[1, 5, 10],

[1, 2, 3]

]

temp2 = path(arr)

print(temp2)

max\_sum=-9999999999

max\_sum\_path = None

for i, path in enumerate(temp2):

path\_values = [arr[row][col] for row, col in path]

path\_sum = sum(path\_values)

print("Sum of path {}: {}".format(i + 1, path\_sum))

if path\_sum > max\_sum:

max\_sum = path\_sum

max\_sum\_path = path

print("max sum:", max\_sum)

print("path:", max\_sum\_path)

**A computer screen shot of a computer

Description automatically generated**

**Question 3:**

def nPr(*arr*):

*arr*.sort(*reverse*=True)

print(*arr*)

n = *arr*[0]

r = *arr*[1]

print("n =", n, "and r =", r)

#simply reverse the array and pick the first 2 elems:

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

nPr(arr)

arr2 = [5, 2, 3, 4, 1]

nPr(arr2)

**A blue square with white text

Description automatically generated with medium confidence**

**Question 4:**

def nPr2(*arr*):

operations = 0

for i in range(len(*arr*) - 1):

if *arr*[i] > *arr*[i + 1]:

operations += *arr*[i] - *arr*[i + 1]

*arr*[i + 1] = *arr*[i]

return operations

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

print(arr,":",nPr2(arr))

arr2=[1, 2, 2, 100]

print( arr2,":",nPr2(arr2))

A blue square with white text

Description automatically generated with medium confidence

**B.2 Observations and learning:**

So I solved question 1 via combinations basically using 2^n and using for loops and if loops for rest . Question 2 was quite tricky wherein I was required to apply the logic to going right , up and diagonally in matrix and store all the combinations and later just find the path with max cost. Question 3 being the most easiest required just sorting the array and selecting the first two elements for the permutation since the max elements will give the max permutation. Question 4 being the same as 3rd just needed few adjustments regarding sorting and noting the count of changes .

**B.3 Conclusion:**

The experiment covered topics regardingMatrix Operation, find minimum cost path, find maximum in an integer array, and array sorting.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*