CT-2:

07.03.2022

Continuous Assessment: 02

Algorithm

1. A 2D array is given. Find out a rectangle in the array so that the sum of the elements in the rectangle is maximum. How many algorithms do you know? Which one is the best algorithm? 10.00

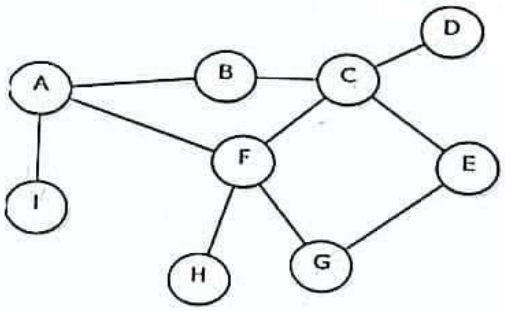
2. A graph is given. Answer the following questions 10.00

   i. What is the length (in number of edges) of the shortest path from A to D?

   ii. What is the length (in number of edges) of the longest cycle free path from G to H?

   iii. Write all nodes reachable from F?

   Show all the steps to find out the answer of above questions.



3. Prove that the time complexity of merge sort is O(nlogn). 5.00

1. Answer:

Given a 2D array, find the maximum sum subarray in it. For example, in the following 2D array, the maximum sum subarray is highlighted with blue rectangle and sum of this subarray is 29.

1	2	-1	-4	-20
-8	-3	4	2	1
3	8	10	1	3
-4	-1	1	7	-6

The **Naive Solution** for this problem is to check every possible rectangle in the given 2D array. This solution requires 6 nested loops –

- 4 for start and end coordinate of the 2 axis $O(n^4)$
- and 2 for the summation of the sub-matrix $O(n^2)$.

The overall time complexity of this solution would be $O(n^6)$.

Efficient Approach –

Kadane's algorithm for 1D array can be used to reduce the time complexity to O(n^3).

The idea is to fix the left and right columns one by one and find the maximum sum contiguous rows for every left and right column pair.

We basically find top and bottom row numbers (which have maximum sum) for every fixed left and right column pair.

To find the top and bottom row numbers, calculate the sum of elements in every row from left to right and store these sums in an array say temp[]. So temp[i] indicates sum of elements from left to right in row i.

If we apply Kadane's 1D algorithm on temp[], and get the maximum sum subarray of temp, this maximum sum would be the maximum possible sum with left and right as boundary columns. To get the overall maximum sum, we compare this sum with the maximum sum so far.

```
// Java Program to find max sum rectangular submatrix

import java.util.*;
import java.lang.*;
import java.io.*;

class MaximumSumRectangle
{

    // Function to find maximum sum rectangular
    // submatrix
    private static int maxSumRectangle(int[][] mat)
    {
        int m = mat.length;
        int n = mat[0].length;
        int preSum[][] = new int[m + 1][n];

        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                preSum[i + 1][j] =
                   preSum[i][j] + mat[i][j];
            }
        }

        int maxSum = -1;
        int minSum = Integer.MIN_VALUE;
        int negRow = 0, negCol = 0;
```

```
    int rStart = 0, rEnd = 0, cStart = 0, cEnd = 0;
    for (int rowStart = 0;
         rowStart < m;
         rowStart++)
    {
        for (int row = rowStart; row < m; row++)
        {
            int sum = 0;
            int curColStart = 0;
            for (int col = 0; col < n; col++)
            {
                sum += preSum[row + 1][col]
                       - preSum[rowStart][col];
                if (sum < 0) {
                    if (minSum < sum) {
                        minSum = sum;
                        negRow = row;
                        negCol = col;
                    }
                    sum = 0;
                    curColStart = col + 1;
                }
                else if (maxSum < sum)
                {
                    maxSum = sum;
                    rStart = rowStart;
                    rEnd = row;
                    cStart = curColStart;
                    cEnd = col;
                }
            }
        }
    }

    // Printing final values
    if (maxSum == -1) {
        System.out.println("from row - " + negRow
                           + " to row - " + negRow);
        System.out.println("from col - " + negCol
                           + " to col - " + negCol);
    }
    else {
        System.out.println("from row - " + rStart
                           + " to row - " + rEnd);
        System.out.println("from col - " + cStart
                           + " to col - " + cEnd);
    }
    return maxSum == -1 ? minSum : maxSum;
}

// Driver Code
public static void main(String[] args)
{
    int arr[][] = new int[][] { { 1, 2, -1, -4, -20 },
                                { -8, -3, 4, 2, 1 },
                                { 3, 8, 10, 1, 3 },
```

```
                                    { -4, -1, 1, 7, -6 } };

        // Function call
        System.out.println(maxSumRectangle(arr));
    }
}
```

Output

```
(Top, Left) (1, 1)

(Bottom, Right) (3, 3)

Max sum is: 29
```

Time Complexity: $O(n^3)$