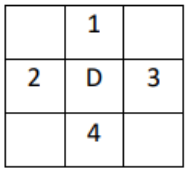
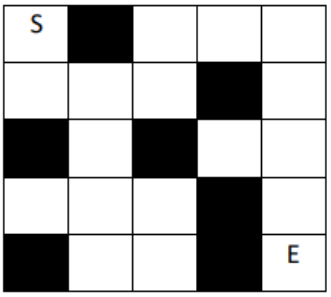
1. **[6 คะแนน]** กำหนดแผนที่ M มาให้เป็นตารางขนาด n x n หน่วย โดยมีจุดเริ่มต้นที่ M(1, 1) และมีสิ่งกีดขวางจำนวน k วัตถุ อยู่ในตำแหน่งต่าง ๆ K[a, b, c] โดยที่ $1 \le a \le k$, $1 \le b$, n และ $1 \le c \le n$ (a คือหมายเลขของวัตถุ K, b คือ row ของวัตถุ a และ c คือ column ของวัตถุ a) จงออกแบบอัลกอริทึมคำนวณเส้นทางและหาระยะทางที่สั้นที่สุดจากจุดเริ่มต้นไปยังจุดจบ M[k, k]. ทั้งนี้ การเคลื่อนที่ของหุ่นยนต์ สามารถเคลื่อนที่ด้วยระยะทาง 1 หน่วย ได้ 4 ทิศ ตามรูป ทั้งนี้คำตอบอยู่ในรูปของ ลำดับของทิศทางตามตัวอย่างต่อไปนี้

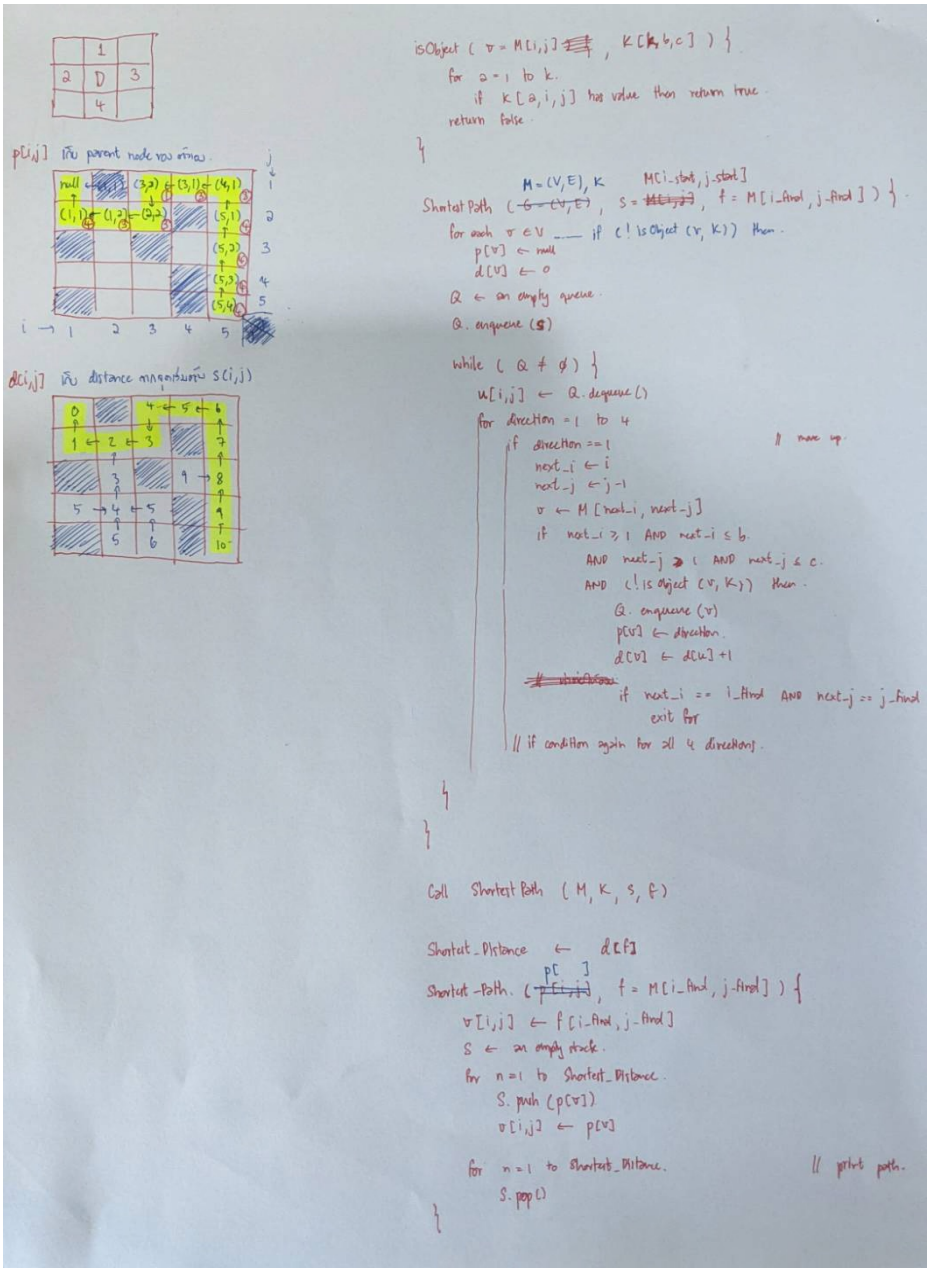


direction

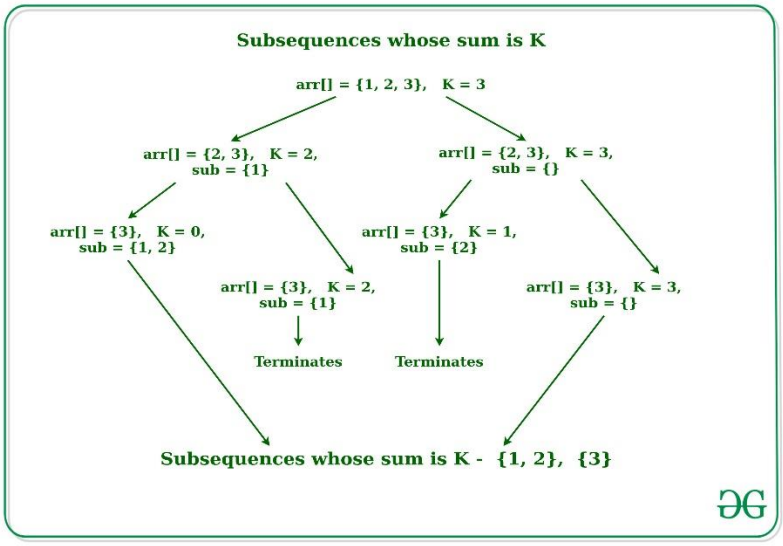


S: start point, E = end point

ระยะทางสั้นสุดคือ **2331224444** คิดเป็นระยะ 10 หน่วย



2. [6 คะแนน] จงออกแบบแลกอิทึมในการหาลำดับย่อย (subsequence) ของลำดับของตัวเลขขนาด n จำนวน ที่ลำดับย่อยนั้นมีผลบวกเท่ากับ K ที่กำหนดมาให้ ทั้งนี้ ในลำดับของตัวเลขอาจมีทั้งจำนวนเต็มบวก ศูนย์ และจำนวนเต็มลบได้



Link: [Find all subsequences with sum equals to K - GeeksforGeeks]

Time Complexity: O(2^n)

Auxiliary Space: O(h)

Here h is height of the tree and the extra space is used due to recursive function call stack. If we exclude the space used to store the result.

```
# Python3 implementation to find all the

# subsequence whose sum is K

# Utility function to find the subsequences

# whose sum of the element is K

def subsetSumToK(arr, n, output, k):


    # Base Case

    if (n == 0):

        if (k == 0):

            output[0][0] = 0;

            return 1;

        else:

            return 0;
```

```
# Array to store the subsequences
# which includes the element arr[0]
output1 = [[0 for j in range(50)] for i in range(1000)]

# Array to store the subsequences
# which not includes the element arr[0]
output2 = [[0 for j in range(50)] for i in range(1000)]

# Recursive call to find the subsequences
# which includes the element arr[0]
size1 = subsetSumToK(arr[1:], n - 1, output1, k - arr[0]);

# Recursive call to find the subsequences
# which not includes the element arr[0]
size2 = subsetSumToK(arr[1:], n - 1, output2, k)

# Loop to update the results of the
# Recursive call of the function
for i in range(size1):

        # Incrementing the length of
        # jagged array because it includes
        # the arr[0] element of the array
        output[i][0] = output1[i][0] + 1;

        # In the first column of the jagged
        # array put the arr[0] element
        output[i][1] = arr[0];

# Loop to update the subsequence
```

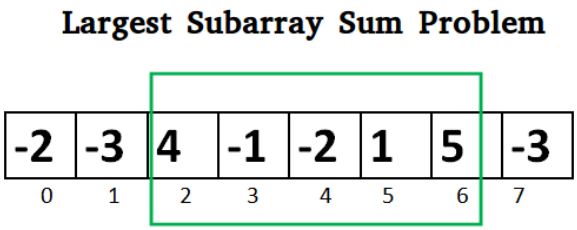
```
        # in the output array
        for i in range(size1):
                for j in range(1, output1[i][0]+1):
                        output[i][j + 1] = output1[i][j];

        # Loop to update the subsequences
        # which do not include the arr[0] element
        for i in range(size2):
                for j in range(output2[i][0] + 1):
                        output[i + size1][j] = output2[i][j];

        return size1 + size2;
# Function to find the subsequences
# whose sum of the element is K
def countSubsequences(arr, n, output, k):
        size = subsetSumToK(arr, n, output, k);
        for i in range(size):
                for j in range(1, output[i][0] + 1):
                        print(output[i][j], end = ' ')
                print()
# Driver Code
if __name__=='__main__':
        arr = [5, 12, 3, 17, 1, 18, 15, 3, 17]
        length = 9
        output = [[0 for j in range(50)] for i in range(1000)]
        k = 6;
        countSubsequences(arr, length, output, k);
# This code is contributed by rutvik_56.
```

3. [6 คะแนน] จงออกแบบอัลกอริทึมสำหรับปัญหาการหา Maximum Contiguous Subsequence Sum คือ กำหนดให้ $a_1, a_2, \ldots, a_n$ เป็นลำดับของจำนวนเต็ม (บวก ลบ หรือ ศูนย์) n จำนวน ต้องการหาว่า ค่ามากที่สุดที่เป็นไปได้ของ sum = $\Sigma_{k=i \text{ to } j} a_k$ สำหรับทุกค่าของ i ถึง j คือเท่าใด (ในกรณีที่ทุกจำนวนเป็นจำนวนเต็มลบ ให้ค่า sum เป็นศูนย์ ซึ่งหมายถึง empty subsequence)

Largest Sum Contiguous Subarray (Kadane's Algorithm)



4 + (-1) + (-2) + 1 + 5 = 7

Maximum Contiguous Array Sum is 7

Largest Sum Contiguous Subarray (Kadane's Algorithm) - GeeksforGeeks

Time Complexity: O(N)
Auxiliary Space: O(1)

Pseudocode

Initialize:
max_so_far = INT_MIN
max_ending_here = 0

Loop for each element of the array

(a) max_ending_here = max_ending_here + a[i]
(b) if(max_so_far < max_ending_here)
max_so_far = max_ending_here
(c) if(max_ending_here < 0)
max_ending_here = 0
return max_so_far

```
# Python program to find maximum contiguous subarray

# Function to find the maximum contiguous subarray

from sys import maxint


def maxSubArraySum(a, size):

        max_so_far = -maxint - 1

        max_ending_here = 0


        for i in range(0, size):

                max_ending_here = max_ending_here + a[i]

                if (max_so_far < max_ending_here):

                        max_so_far = max_ending_here


                if max_ending_here < 0:

                        max_ending_here = 0

        return max_so_far


# Driver function to check the above function

a = [-2, -3, 4, -1, -2, 1, 5, -3]

print "Maximum contiguous sum is", maxSubArraySum(a, len(a))

# This code is contributed by _Devesh Agrawal_
```

4. [6 คะแนน] กำหนดให้มีสายอักขระสองสาย A[n] และ B[m] ขนาด n และ m ตามลำดับ จงออกแบบอัลกอริทึมในการหา longest common substring ของ A และ B ที่กำหนดมาให้

Longest common subarray in the given two arrays - GeeksforGeeks

Time Complexity: O(N*M),

where N is the length of array A[] and M is the length of array B[].
Auxiliary Space: O(N*M)

```
# Python program to DP approach
# to above solution
# Function to find the maximum
# length of equal subarray

def FindMaxLength(A, B):
    n = len(A)
    m = len(B)
    # Auxiliary dp[][] array
    dp = [[0 for i in range(n + 1)] for i in range(m + 1)]
    # Updating the dp[][] table
    # in Bottom Up approach
    for i in range(n - 1, -1, -1):
        for j in range(m - 1, -1, -1):
            # If A[i] is equal to B[i]
            # then dp[j][i] = dp[j + 1][i + 1] + 1
            if A[i] == B[j]:
                dp[i][j] = dp[i + 1][j + 1] + 1
    maxm = 0
```

```
        # Find maximum of all the values
        # in dp[][] array to get the
        # maximum length
        for i in dp:
            for j in i:
                # Update the length
                maxm = max(maxm, j)
        # Return the maximum length
        return maxm

# Driver Code
if __name__ == '__main__':
        A = [1, 2, 8, 2, 1]
        B = [8, 2, 1, 4, 7]
        # Function call to find
        # maximum length of subarray
        print(FindMaxLength(A, B))
```

5. [6 คะแนน] กำหนดให้ B คือ อาเรย์ขนาด $n \times n$ ซึ่งเก็บเฉพาะเลข 1 กับ 0 โดยที่ในแต่ละแถวแนวนอนของ B เลข 1 ทุก ๆ ตัวต้องมาก่อนเลข 0 (จากซ้ายมาขวา)

	0	1	2	3	4	5
0	1	1	1	0	0	0
1	1	0	0	0	0	0
2	1	1	1	1	0	0
3	1	1	0	0	0	0
4	1	0	0	0	0	0
5	1	1	0	0	0	0

5.1 จงออกแบบอัลกอริทึมที่หาว่าแถวแนวนอนแถวใดใน B ที่มีจำนวนของเลข 1 มากที่สุด (หาเพียงแถวเดียวก็พอในกรณีที่มีจำนวนมากสุดเท่ากันหลายแถว) ตัวอย่างเช่นแถวแนวนอนหมายเลข 2 ของอาเรย์ B ข้างล่างนี้มีจำนวนเลข 1 มากที่สุด โดยมีความซับซ้อนเชิงเวลาเป็น $O(n)$ เมื่อ n คือจำนวนแถว (หรือคอลัมภ์)

5.2 ทดสอบอัลกอริทึมกับตัวอย่างที่ให้ และนับจำนวนการเปรียบเทียบ ว่ามีทั้งหมดเท่าไร (เทียบค่าในอาเรย์ว่าเป็น 0 หรือ 1)

[Find the row with maximum number of 1s - GeeksforGeeks]

Time Complexity: O(m+n)

where m is number of rows and n is number of columns in matrix

.

Auxiliary Space: O(1), as implicit stack is created due to recursion.

The worst case of the above solution occurs for a matrix like following.
0 0 0 … 0 1
0 0 0 ..0 1 1
0 … 0 1 1 1
….0 1 1 1 1

Following method works in O(m+n) time complexity in worst case.

- *Step1: Get the index of first (or leftmost) 1 in the first row.*
- *Step2: Do following for every row after the first row*
  - *…IF the element on left of previous leftmost 1 is 0, ignore this row.*
  - *…ELSE Move left until a 0 is found. Update the leftmost index to this index and max_row_index to be the current row.*
  - *The time complexity is O(m+n) because we can possibly go as far left as we came ahead in the first step.*

```
# Python3 program to find the row
# with maximum number of 1s
# Function that returns
# index of row with maximum
# number of 1s.
def rowWithMax1s( mat):

        # Initialize max values
        R = len(mat)
        C = len(mat[0])
        max_row_index = 0
        index=C-1;
        # Traverse for each row and
        # count number of 1s by finding
        # the index of first 1
        for i in range(0, R):
        flag=False #to check whether a row has more 1's than previous
        while(index >=0 and mat[i][index]==1):
                flag=True #present row has more 1's than previous
                index-=1
                if(flag): #if the present row has more 1's than previous
                max_row_index = i
        if max_row_index==0 and mat[0][C-1]==0:
                return 0;
        return max_row_index
# Driver Code
mat = [[0, 0, 0, 1],
        [0, 1, 1, 1],
        [1, 1, 1, 1],
        [0, 0, 0, 0]]
print ("Index of row with maximum 1s is",rowWithMax1s(mat)) # by Rishabh Chauhan
```