**1. Sequential Search**

**Given an array arr of positive integers sorted in a strictly increasing order, and an integer k.**

**Return the kth positive integer that is missing from this array.**

**Example 1:**

**Input: arr = [2,3,4,7,11], k = 5**

**Output: 9**

def find(arr, k):

arr\_set=set(arr)

missing\_count=0

current\_num=1

while missing\_count<k:

if current\_num not in arr\_set:

missing\_count+=1

if missing\_count==k:

return current\_num

current\_num+=1

arr=[2,3,4,7,11]

k1=5

print(find(arr1,k1))

Output: 9

**2. Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.**

**Example 1:**

**Input: haystack = "sadbutsad", needle = "sad"**

**Output: 0**

**Explanation: "sad" occurs at index 0 and 6.**

def bruteforce(text,pattern):

n=len(text)

m=len(pattern)

found=False

for i in range(n-m+1):

j=0

while j<m and pattern[j]==text[i+j]:

j+=1

if j==m:

print(f"pattern is found at index:{i}")

found=True

if not found:

print("pattern not found")

text="sadbutsad"

pattern="sad"

bruteforce(text,pattern)

output:

pattern is found at index :0

pattern is found at index :6

**3. Given an array of strings words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.**

**You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.**

**Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.**

**For the last line of text, it should be left-justified, and no extra space is inserted between words.Example 1:**

**Input: words = ["This", "is", "an", "example", "of", "text", "justification."], maxWidth = 16**

**Output:**

**[ "This is an",**

**"example of text",**

**"justification. "**

**]**

def fullJustify(words,maxWidth):

def justify\_line(line,width,is\_last\_line):

if is\_last\_line:

return ' '.join(line).ljust(width)

if len(line)==1:

return line[0].ljust(width)

total\_spaces=width-sum(len(word) for word in line)

spaces\_between\_words=len(line)-1

min\_spaces=total\_spaces//spaces\_between\_words

extra\_spaces=total\_spaces%spaces\_between\_words

result=''

for i in range(spaces\_between\_words):

result+=line[i]+' '\*(min\_spaces+(1 if i < extra\_spaces else 0))

result+=line[-1]

return result

result, line, num\_of\_letters=[],[],0

for word in words:

if num\_of\_letters+len(word)+len(line)>maxWidth:

result.append(justify\_line(line,maxWidth,False))

line, num\_of\_letters=[],0

line.append(word)

num\_of\_letters+=len(word)

result.append(justify\_line(line, maxWidth, True))

return result

words = ["This","is","an","example","of","text","justification."]

maxWidth=16

justified\_text=fullJustify(words, maxWidth)

for line in justified\_text:

print(f'"{line}"')

output:

"This is an"

"example of text"

"justification. ‘’

**4. Word Break Problem**

**Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.**

**Note that the same word in the dictionary may be reused multiple times in the segmentation.**

**Example 1:**

**Input: s = "leetcode", wordDict = ["leet","code"]**

**Output: true**

def wordBreak(s,wordDict):

wordSet=set(wordDict)

dp=[False]\*(len(s)+1)

dp[0]=True

for i in range(1,len(s)1):

for j in range(i):

if dp[j] and s[j:i] in wordSet:

dp[i]=True

break

return dp[len(s)]

print(wordBreak("leetcode",["leet","code"]))

output:

True

**5. Warshall’s & Floyd’s Algorithm**

**There is a robot on an m x n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]). The robot tries to move to the bottom-right corner (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.**

**Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.**

**The test cases are generated so that the answer will be less than or equal to 2 \* 10 9.**

**Example 1:**

**Input: m = 3, n = 7**

**Output: 28**

**Example 2:**

**Input: m = 3, n = 2**

**Output: 3**

def uniquePaths(m,n):

dp=[[1]\*n for \_ in range(m)]

for i in range(1,m):

for j in range(1,n):

dp[i][j]=dp[i-1][j]+dp[i][j-1]

return dp[m-1][n-1]

print(uniquePaths(3,7))

print(uniquePaths(3,2))

output:

28

3