1004. Max Consecutive Ones III

```
class Solution:
    def longestOnes(self, A: List[int], K: int) -> int:
        l, r = 0, 0
        max_length = 0
        while r < len(A):
            if A[r] == 0:
                K -= 1
            r += 1
            while K < 0:
                if A[l] == 0:
                    K += 1
                l += 1
            max_length = max(max_length, r - l)
        return max_length
```

Optimized version:

```
class Solution:
    def longestOnes(self, A: List[int], K: int) -> int:
        l, r = 0, 0
        max_length = 0
        while r < len(A):
            if A[r] == 0:
                K -= 1
            r += 1
            if K < 0:
                if A[l] == 0:
                    K += 1
                l += 1
        return r - l
```

139. Word Break

We will use dp. TC is O(n**2), SC is O(n)

```
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        mem = {}
        mem[0] = True
        length = len(s)
        wordDict = set(wordDict)
        for i in range(1, length + 1):
            for j in range(i):
                if j in mem and s[j:i] in wordDict:
                    mem[i] = True
```

```
                break
        return length in mem
```

140. Word Break II

```
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> List[str]:
        wordDict = set(wordDict)
        mem = {}
        def dfs(cur_i):
            if cur_i == len(s):
                return ['']
            if s[cur_i:] in mem:
                return mem[s[cur_i:]]
            result = []
            for w in wordDict:
                idx = len(w) + cur_i
                if s[cur_i:idx] == w:
                    sub = dfs(idx)
                    for l_w in sub:
                        if l_w:
                            result.append(w + ' ' + l_w)
                        else:
                            result.append(w)
            mem[s[cur_i:]] = result
            return result
        return dfs(0)
```

490. The Maze

We will use bfs. TC is O(m*n)

```
class Solution:
    def hasPath(self, maze: List[List[int]], start: List[int], destination: List[int]) -> bool:
        directions = [[0, -1], [0, 1], [1, 0], [-1, 0]]
        q = [start]
        rows = len(maze)
        cols = len(maze[0])
        while q:
            x, y = q.pop()
            maze[x][y] = 2
            if destination == [x, y]:
                return True
            for d_x, d_y in directions:
                new_x, new_y = x + d_x, y + d_y
                while 0 <= new_x < rows and 0 <= new_y < cols and maze[new_x][new_y] != 1:
```

```
            new_x += d_x
            new_y += d_y
        new_x -= d_x
        new_y -= d_y
        if maze[new_x][new_y] == 0:
            q.append([new_x, new_y])
return False
```