cs3821 Assignment 3

Jennifer Xu (z5207930)

Question 2

2.1 Table

Sequence	***k***
[R, B, R, G, O, O, G, T, I, I, T]	3
[R, G, B, G, R]	1
[C, C, C, C, A, X, B, X]	3
[Z, X, Y, Z, X]	1
[A, BD, C, BD, A]	1
[A, O, A, B, X, Z, Y, X]	6
[A, B, B, A, C, A, B]	3

2.2 Dynamic Programming Algorithm

We need to keep track of two things: The number of partitions we created and whether string[i] - string[j] is a palindrome. Consider the following code.

```
strLen = len(string)
   nPartitions = [[-1 for i in range(strLen)] for i in range(strLen)]
   isPal = [[False for i in range(strLen)] for i in range(strLen)]
```

Subproblem

Our subproblem would be to look at the minimum number of palindromes in a substring denoted by *P(i, j)* where i and j are the index values in a given string and `0 <= i <= j <= len(string).`

Base Cases

The base case would be when the string/substring is of length 1, then it would be a valid palindrome.

```
   # Base cases for when length is 1
   for i in range(strLen):
       nPartitions[i][i] = 1
       isPal[i][i] = True
```

Recurrence Relation

We want to find every possible place to split our substring in order to find the minimum amount of partitions we would require. Consider the following code where we find the minimum number of partitions to form our substring by comparing `string[i] to string[i + k] and string[i + k + 1] to string[j].`

```
 nPartitions[head][tail] = infinite
 for i in range(head, tail):
     nPartitions[head][tail] = min(nPartitions[head][tail],
                             nPartitions[head][i] + nPartitions[i+1][tail] + 1)
```

Pseudo Code

```
def minPalindrome(string):
    strLen = len(string)
    nPartitions= [[0 for i in range(strLen)] for i in range(strLen)]
    isPal = [[False for i in range(strLen)] for i in range(strLen)]

    # Base cases for when length is 1
    for i in range(strLen):
        nPartitions[i][i] = 1
        isPal[i][i] = True

    for substringLength in range(2, strLen + 1):
        for head in range(strLen - substringLength + 1):
            tail = head + substringLength - 1

            # Check if palindrome based on string size
            if substringLength == 2:
                isPal[head][tail] = (string[head] == string[tail])
            else:
                isPal[head][tail] = ((string[head] == string[tail])
                                     and (isPal[head+1][tail-1]))

            # Cut the string further depending on if it's palindrome or not
            if isPal[head][tail] == True: nPartitions[head][tail] = 1
            else:
                minCost[head][tail] = infinite
                for i in range(head, tail):
                    nPartitions[head][tail] = min(nPartitions[head][tail],
nPartitions[head][i] + nPartitions[i+1][tail] + 1)

    return nPartitions[0][strLen - 1]
```

2.3 Worst Case Complexity

As we are using 3 nested for loops, the worst-case time complexity would be *O(n^3)*.