CSCI 570 Fall 2021
HW7 P5 Solution

1.4 Valid Palindrome if you can remove at most K characters

Given a string s and an integer k, find out if it can be transformed into a palindrome by removing at most k characters from it.

A) Let's first try to come up with a recursive solution *validPalindrome()*. Consider the bite sized question, "if I can remove $\leq$ K characters, can I make the string between indices i and j a palindrome?" Create a recursive solution that answers this bite sized question.

Solution.

bool validPalindrome(String s, int i, int j, int k):
if (k $<$ 0) return false // Return false if we removed too many characters.
if (i $\geq$ j) return true// Return true if we are looking at one character or an empty string.
if (s[i] == s[j]) return validPalindrome(s, i+1, j-1, k) // Look at next two characters if they match.
return helper(s, i+1, j, k-1) OR helper(s, i, j-1, k-1) // "remove" one of the chars and try again (dec. k)

B) Your *validPalindrome()* function likely takes 3 numbers as parameters: the starting index, ending index, and the remaining amount of characters you are allowed to remove. Think about how this solution would convert to a dynamic programming solution. You may be tempted to use a 3D array to save each result:

```
int answers[/*start index*/][/*end index*/][/*# of removable chars*/]
```
(2)

If you iterate over all three indices in order to fill up the array, the runtime and space complexity of your solution becomes O($n^2 \cdot k$). This is very not great. Let's use a different bite sized question: "HOW MANY characters do I have to remove in order to make the string between indices i and j a palindrome?" Create a recursive solution that answers this bite sized question.

Solution.

int validPalindrome(String s, int i, int j):

if (i $\geq$ j) return 0

// Return 0 if we are looking at one char or an empty string.

// Current answer = same answer as removing the two end characters.

if (s[i] == s[j]) return validPalindrome(s, i+1, j-1)

// Return 1 + minimum number of characters to make a palindrome with one of the end characters removed.

return 1 + MIN(helper(s, i+1, j), helper(s, i, j-1))

C) If you were to save the solutions in an array, the array would look like:

```
int answers[/*start index*/][/*end index*/]
```
(3)

This drastically cuts down on the space and time used by your solution. Now consider the order that you would need to fill up this array based on your recursive solution. Once you figure that out, write out your final solution. What is the new runtime and space complexity?

Solution.

ITERATIVE SOLUTION:

bool isValidPalindrome(String s, int k):

int answers[len(s)][len(s)]// Save answers here.

for i from len(s) - 2 to 0:// Iterate from end of the string to the beginning.

for j from i + 1 to len(s) - 1:// Start with smaller strings and end with larger ones.

// This is essentially just cut and paste from the recursive solution in part B.

if (s[i] == s[j]):

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

else:

answers[i][j] = 1 + MIN(answers[i][j - 1], answers[i + 1][j])

// Answer lies at the index that represents the string starting at 0 and ending at the end of the string.

return arr[0][len(s)] $\leq$ k

TIME AND SPACE COMPLEXITY

Time complexity is $O(n^2)$. Space complexity is $O(n^2)$ NOTE: There is a solution with a space complexity of $O(n)$, but we do not expect that answer.