CS/ECE 374 Spring 2017
Homework 5 Problem 2

Lanxiao Bai (lbai5)
Renheng Ruan (rruan2)
Ho Yin Au (hoyinau2)

Recall that a *palindrome* is any string that is exactly the same as its reversal, like I, or DEED, or RACECAR, or AMANAPLANACATACANALPANAMA.

Any string can be decomposed into a sequence of palindrome substrings. For example, the string BUBBASEESABANANA ("Bubba sees a banana.") can be broken into palindromes in the following ways (among many others):

BUB • BASEESAB • ANANA

B • U • BB • A • SEES • ABA • NAN • A

B • U • BB • A • SEES • A • B • ANANA

B • U • B • B • A • S • E • E • S • A • B • ANA • N • A

Describe and analyze an efficient algorithm that given a string w and an integer k decides whether w can be split into palindromes each of which is of length at least k. For example, given the input string BUBBASEESABANANA and 3 your algorithm would answer yes because one can find a split BUB • BASEESAB • ANANA. The answer should be no if we set $k = 4$. Note that the answer is always yes for $k = 1$.

Solution: We can think of string w is an array of char, which is represented as $w[1..n]$ for the string w with length n. We define that the algorithm should return TRUE if the answer is yes, and should return FALSE if the answer is no. First, we check whether w itself has length at least k. If not, there is no way to split it with palindrome with length at least k. Then, we check whether w is a palindrome. If yes, w can be decomposed into palindrome set that only contains itself.
If not, we can decompose w into palindrome substrings by trying to find all the leftmost palindrome with length at least k and then recursively find decomposition of palindrome substrings on the remaining right substring, which definitely the right substring should also has length at least k. As we need to make sure the both sides of the string w after splitting still have length at least k, we first make sure that n is larger than $2k$. Then, we only check all the split position that satisfy both sides after splitting still have length at least k. We do the checking in ascending order according to the length of the right split. After that, we just check whether the left split is palindrome, and then recursively do finding decomposition on the right split.
We define a 1-dimension array ISCALCULATED$[1..n]$ to memoize whether the right split has already been calculated. We also define another 1-dimension array DECOMPOSABLE$[1..n]$ to memoize the return value of the right split, which is whether right split is decomposable into palindromes satisfying at least length k. Both arrays have initialized with value FALSE. For example, ISCALCULATED$[i]$ stores whether the right split of length i has already been calculated. If TRUE, DECOMPOSABLE$[i]$ stores whether the right split of length i is decomposable into palindromes. We can memoize the function like this because the right end of the right split is always the same as the right end of the original string w.

DECOMPOSEPALINDROMESUBSTRINGS is defined as follow:

```
DECOMPOSEPALINDROMESUBSTRINGS(w[1..n], k):

  if ISCALCULATED[n]
      return DECOMPOSABLE[n]
  ISCALCULATED[n] ← TRUE
  if n < k
      return FALSE
  if ISPALINDROME(w[1..n])
      return TRUE
  if n < 2k
      return FALSE
  DECOMPOSABLE[n] ← FALSE
  for i ← k to n − k
      if ISPALINDROME(w[1..i])
          if DECOMPOSEPALINDROMESUBSTRINGS(w[i + 1..n], k)
              CANDECOMPOSE ← TRUE
  return DECOMPOSABLE[n]
```

The algorithm use a subroutine ISPALINDROME to check whether the string passed into is a palindrome, which is defined as follow:

```
ISPALINDROME(w[1..n]):

  for i ← 1 to ⌊n/2⌋
      if w[i] = w[n − i + 1]
          return FALSE
  return TRUE
```

The algorithm runs in $O(\frac{n^3}{k})$). The time $T(n)$ needed for the algorithm is defined recursively by $T(n) = (n-2k)T(n-k) + O(n)$, which $O(n)$ comes from ISPALINDROME, and $(n-2k)T(n-k)$ comes from that we are searching in the range k to $n-k$ that each value in the range can possibly recurse into a new problem of size at most $n-k$, which is the length of right split. Although we already use memorization, we can only avoid the recalculation of special case that the left palindrome is composed by at least 2 smaller palindromes, which is very rare. After expanding the recursive definition of $T(n)$, we get $O(1) + kO(n)\frac{n^2}{k^2} = O(\frac{n^3}{k})$. ■