Question 1.4 from CTCI: Palindrome Permutation

February 7, 2018

Question

Given a string, write a method to check if it is a permutation of a palindrome. A palindrome is a word or phrase that is the same forwards and backwards. A permutation is a rearrangement of letters. The palindrome does not need to be limited to just dictionary words.

Example:
Input: Tact Coa
Output: True (permutations: "taco cat", "atco cta", etc.)

Explanation and Algorithm

There are a few different solutions to the problem:

First, it is important to understand what is meant by a permutation of a palindrome. A palindrome is defined as a word or phrase that is the same forwards and backwards. In order to satisfy this property, a palindrome must have even counts of almost every character. At most, a single character can have an odd count. This would be the middle character in the palindrome. A permutation of a string that is a palindrome would have characters rearranged so that it would not read the same forwards and backwards. To check that a string is a permutation of a palindrome, we would only need to check that the number of occurences of characters satisfies the property of a palindrome since the order is unimportant.

The first solution uses an *int* array to track the number of occurrences for each character in the string. In this problem, we will only be concerned with alphabetical characters (case insensitive) so we will use a fixed size array. For each alphabetical character in the string, we will compute its index in the *int* array and increment its number of occurences, and ignore non-alphabetical characters. Once we have updated the occurences for all characters in the string, we then iterate through the *int* array and ensure that at most one character has an odd count.

The second solution doesn't improve the time complexity since any algorithm involves processing the entire string. However, this solution makes small improvements on the first solution. In the first solution, we check that the count property of a palindrome is satisfied *after* going through the string and counting the number of occurences. We can instead check that this property is satisfied as we update the number of occurences. Instead of tracking this with a boolean variable, we can count the number of characters with odd character counts after each update to the *int* array. After processing the entire string, we can then check that the number of characters with odd occurences in the string is at most 1 and thus determine if the string is a permutation of a palindrome.

Hints

1. Consider what property or properties must be satisfied for a string to be a permutation of a palindrome. This will help you formulate a solution to the problem.
2. A naive approach would be to determine all the permutations of the input string and check if one of the permutations is a palindrome. Such an approach would be impractical. Try to solve the problem without doing this.
3. A key approach to this problem involves keeping track of the number of occurences for each character in the input string. How might you do this?
4. If you tracked character occurrences in an *int* array and then iterated through the array to check the character counts, how might you modify the solution to improve it?
5. Try tracking the number of characters that have an odd number of occurences as you update the occurence counts.

Code

```
/* Solution 1 */

/* Maps each character to a number. a->0, b->1, c->2, etc.
   This is case insensitive. Non-letter characters map to -1.*/
public int GetIndex(char c) {

   if(c < 'a' || c > 'z') {
      return -1;
   }

   return c - 'a';
```

```
}

public boolean isPermutationOfPalindrome(String s) {

   String str = s.toLowerCase();
   boolean oneOdd = false;
   int[] letters = new int[26];

   for(char c : str.toCharArray()) {
      int index = GetIndex(c);
      if(index == -1) {
         continue;
      }
      letters[index]++;
   }

   for(int i = 0; i < 26; i++) {
      if(letters[i]%2 == 1) {
         if(oneOdd) {
            return false;
         }
         oneOdd = true;
      }
   }

   return true;
}
```

```
/* Solution 2 */

/* Maps each character to a number. a->0, b->1, c->2, etc.
   This is case insensitive. Non-letter characters map to -1.*/
public int GetIndex(char c) {

   if(c < 'a' || c > 'z') {
      return -1;
   }

   return c - 'a';
}

public boolean isPermutationOfPalindrome(String s) {

   String str = s.toLowerCase();
   int[] letters = new int[26];
   int numOdd = 0;

   for(char c : str.toCharArray()) {
```

```
      int index = GetIndex(c);
      if(index == -1) {continue;}
      letters[index]++;

      if(letters[index] % 2 == 1) {
         numOdd++;
      } else {
         numOdd--;
      }
   }

   return numOdd <= 1;
}
```

Big O Analysis

1. Solution 1: The running time is *O(n)* where n is the length of the input string.
2. Solution 2: The running time is *O(n)* where n is the length of the input string.

Sources

Question, answer and other material taken from Cracking the Coding Interview 6th edition by Gayle Laakmann McDowell.