CSC 317 & INF 307

Lab 2 – Discussion of Problem Analysis Tasks

Discuss and solve the following task in Java (for IT students) and Python (for CS students)

1. **Given an array of integers *arr* and an integer *k*, find the *kth* largest element.**

   Discussion

   Assuming arr = [18, 7, 0, 1, 15, 25, 5] and k = 3, then it means we are looking for the 3rd largest number in the array.

   Looking at the values in the array, we would identify "25" as the largest, "18" as the 2nd largest, and "15" as the third largest.

   Now, assuming we have an array of N unknown numbers, how will we go about this? Of course, everything become easier when the array is sorted. Do you remember sorting from the previous lab session?

   After the array has been sorted, it becomes easier to identify the kth largest element. If the array is sorted in descending order (from largest to smallest), then we start searching from the beginning of the array. On the other hand, if the array was sorted in ascending order (smallest to largest) then we start searching from the back/end of the array.

   Solution (Logical steps)

   a. Sort the array in either ascending or descending order (use bubble sort in the class, but any sorting algorithm works)
   b. If array sorted in descending order, the kth largest element would be in position k-1 (arrays are indexed from 0).
   c. If array sorted in ascending order, the kth largest element would be in position (arr_length -k).
      i. The largest element would be the last in the array => index = (arr_length – 1)
      ii. The 2nd largest would be the last-but-one in the array => index = (arr_length – 2)
      iii. If we continue on this path, the kth largest would be found at index = (arr_length – k)

2. **Given two string arrays *s1* and *s2*, write a function that checks if they are anagrams. Two strings are anagrams if they are made of the same characters with the same frequencies. E.g., danger and garden; salesmen and nameless**

Discussion

Based on the definition and conditions for two strings to be considered anagrams, they first have to be of the same length. If they are of different lengths, then they do not have the same frequencies of characters.

Having the same length is not enough, we need to check if they are made of the same characters. Every character in s1 should be in s2, and vice versa.

But is this enough? Consider “salesman” and “nameless”. Every character in s1 appears at least once in s2; same can be said of s2. But if we take a loser look, the character ‘a’ has different frequencies in both strings. So we need to also check the count.

Option 1:

One alternative is to use a data-structure like the HashMap (in Java) or Dict (in python) to capture the frequencies of each character in each string.

Example: salesman = {a:2, e:1, l:1, m:1, n:1, s:2} and nameless = { a:1, e:2, l:1, m:1, n:1, s:2}. We can easily spot the difference in frequencies here.

Option 2:

Another simply alternative would be to sort both strings and then compare. Example: salesman = {a, a, e, l, m, n, s, s} and nameless = { a, e, e, l, m, n, s, s}. We can easily spot the difference here as well.

Solution (Logical steps)

Option 1:

a. Check if the two string arrays are the same length. If no, print “s1 and s2 are not anagrams”. If yes, go to b.
b. Create a hashmap/dict from the characters in s. Do the same for s2.
c. Loop through the hashmap/dict and check if:
    i. the characters in s1_dict[i] and s2_dict[i] are equal.
        - If they are all equal, check if they have the same frequency.
    ii. If all characters are equal, and all have the same frequency, then we have an angram

Option 2:

a. Check if the two string arrays are the same length. If no, print “s1 and s2 are not anagrams”. If yes, go to b.
b. Sort s1 and s2.
c. Loop through the arrays and check if the characters in s1[i] and s2[i] are equal. If they are all equal, then we have an anagram.