CSCI 421 Design and Analysis of Algorithms Spring 2019

Midterm Exam

1. In Lecture 1, we use a model to analyze some specific problems and related algorithms to solve them (Theory of algorithms). Assume we are solving 1-sum problem, describe the meaning of “upper bound”, “lower bound”, respectively. Also, describe in which situation do we have optimal algorithm.
   1. Upper bound: O(N) runtime, basically it looks at every item in the array.
   2. Lower bound: Ω(N) runtime, Proof that no algorithm can do better.
   3. Optimal: Where lower bound is equal to upper bound(to within constant factor).
2. We say two strings are equivalent if one is an anagram of the other, that is, one string can be turned into the other by rearranging the characters. An equivalence class is a set in which any pair is equivalent. For example, in the array ["aaa", "aab", "aba"], there are two equivalence classes since the latter two strings are equivalent, but neither is equivalent to the first string.

The problem is: given an array of strings, find the number of equivalence classes among them. Fortunately, we have an idea on how to solve this problem as follows. You need to choose a sorting algorithm we learned in Lecture 2 and implement the idea to solve the aforementioned problem. Attach your code and screenshot here. Also, you need to give the theoretical running time of your solution in terms of N (size of the array) and m (maximum length of the strings). Note that the running time will be depending on the sorting algorithm you use.

Proposed solution:

The solution has three steps.

First step: iterate through the array, and for each element S of the array, sort the underlying character array of S using *your chosen sorting algorithm*. (i.e. convert it to a char[], sort the char[], then convert the char[] to a string again). Now, each string is in alphabetical order, so two strings are anagrams if and only if they are equal as strings.

Second step: sort the whole array, again using *your chosen sorting algorithm*. This ensures that equal strings (which are in 1-1 correspondence with equivalent strings in the original array) are next to each other.

Third step: iterate through the array and count how many distinct elements appear. (To be precise, count the number of values i such that a[i] != a[i-1], and add 1).

public class midterm {

public static void selectSort(String[] stringIN) {

char[] charArr;

for (int i = 0; i < stringIN.length; i++) {

charArr = stringIN[i].toCharArray();

for (int j = 0; j < charArr.length; j++) {

int min = j;

for (int k = j + 1; k < charArr.length; k++) {

if (less(charArr[k], charArr[min]))

min = k;

}

exch(charArr, j, min);

stringIN[i] = String.valueOf(charArr);

}

}

System.out.println("Array after sorting: ");

for (int i = 0; i < stringIN.length; i++) {

int min = i;

for (int j = i + 1; j < stringIN.length; j++) {

if (lessString(stringIN[j], stringIN[min]))

min = j;

}

exchStr(stringIN, i, min);

}

for (int i = 0; i < stringIN.length; i++)

System.out.println(stringIN[i] + "");

int count = 0;

for (int i = 0; i < stringIN.length; i++)

if (i == 0)

count++;

else if (!stringIN[i].equals(stringIN[i - 1]))

count++;

System.out.println("Found " + count + " Distinct elements");

}

private static boolean less(Character v, Character w) {

return v.compareTo(w) < 0;

}

private static boolean lessString(String v, String w) {

return v.compareTo(w) < 0;

}

private static void exch(char[] c, int i, int j) {

char swap = c[i];

c[i] = c[j];

c[j] = swap;

}

private static void exchStr(String[] s, int i, int j) {

String swap = s[i];

s[i] = s[j];

s[j] = swap;

}

public static void main(String[] args) {

String[] stringIN = { "acb", "ccc", "cba", "bca", "aaa", "bbb", "aab" };

System.out.println("Unsorted array: ");

for (int i = 0; i < stringIN.length; i++)

System.out.println(stringIN[i] + " ");

long startTime = System.currentTimeMillis();

selectSort(stringIN);

long endTime = System.currentTimeMillis();

long elapsed = endTime - startTime;

System.out.println("Elapsed time : " + elapsed);

}

}

A screen shot of a computer

Description automatically generated