Exam of Algorithms and Data Management

G. Falquet, V. Daponte

January 24, 2019

Remarks.

- All documents on paper are allowed, all electronic devices are strictly forbidden.
- The algorithms must be written in Scala (minor syntax errors are allowed, provided the code remains non ambiguous).
- All questions have the same weight (20%)

Question 1 - Algorithm on arrays

- Write an algorithm that, given an array L of positive and negative numbers, constructs a new array N taht satisfies the following conditions:

  1. Ncontains the same elements as L
  2. Let f be the the first element of L. N begins with all the elements of L that are $\leq f$ (in any order), then come all the elements that are $> f$ (in any order).

  For example if $L = (-1, -2, 6, -6, 66, 4, 1, -5, -7, -9, 14)$ a possible content for N is $(-7, -9, -5, -6, -2, -1, 66, 4, 1, 6, 14)$.

- Compute and justify the time complexity of the algorithm produced.

Question 2 - Complexity

Consider the following algorithm expressed in Scala

```
def cntr(words: List[String], text: List[String]):
  Map[String, Int] = {
   var res = Map[String, Int]()
   for (w <- words) {
     var count = 0
     for (t <- text) {
        if (w == t) count += 1
```

```
      }
      res = res + (w -> count)
    }
    return res
}
```

1. If `someWords = List("x","bop","bip")` and `someText = List("a", "bop", "bip", "x", "bip", "bip", "bop")`, what would be the value (map) returned by `cntr(someWords, someText)`?
2. What is the time complexity of this algorithm as a function of the size N of `words` and the size M of `text`?

Reminder. Testing if an object belongs to the keyset of a map and adding, removing, or modifying a pair in a `Map` takes (average) constant time.

Question 3 - Recursion

Consider a recursive algorithm that takes two strings s_1 and s_2 as input and checks if these strings are the anagram of each other, hence if all the letters contained in the former appear in the latter the same number of times, and vice versa (i.e. s_2 is a permutation of s_1).

Example:

if $s_1 =$ "$elevenplustwo$" and $s_2 =$ "$twelveplusone$" the output is $true$

if $s_1 =$ "$amina$" and $s_2 =$ "$minia$" the output is $false$

For this exercise outline:

1. The base case of the algorithm;
2. The termination condition (for both positive and negative outcome);
3. The recursive step

   Hint: consider the first character $c = s_1(0)$ of s_1 and the rest of $r = s_1.substring(1, s_1.size)$ of s_1. What are the conditions that s_2 must (recursively) satisfy with respect to c and r?
4. The Scala code implementation of the algorithm.

Reminder.

- the function $s.indexOf(c)$ returns the position (index) of the character c in the string s, or -1 if c is not in s
- $s.substring(i, j)$ returns the substring of s that starts at position i and end at j-1

Question 4 - Graphs

For each of the weighted graphs G1, G2, and G3 below

1. What will be the shortest path from a to z found by applying Dijkstra's algorithm?
2. How many iterations (in the algorithm main loop) will be necessary to find this path?

   (motivate your answers, in no more than 3 lines).

graphs-for-dijkstra.pdf

Question 5. Databases

Given the following relational schema (the primary keys are underlined)

```
city(name, population, area)
trainStationIn(name, city)
trainLine(name, length)
stationOn(line, seqNo, station)
```

where

- a tuple (n, c) of *trainStation* represents the fact that the train station named n is in the city named c.
- a tuple (n, k) of *trainLine* represents the fact that the train line named n has k stations (thus $k - 1$ segments)
- a tuple (l, n, s) of *stationOn* represents the fact that s is the name of the nth station on the line named l.

1. Write SQL queries to answer the following questions:

(a) what are the cities that have at least one train stations and less than 20000 inhabitants?

(b) what are the lines that are circular, i.e. their first and last stations are the same?

(c) for each train line, what are the cities connected through this line?

(d) what is the first station of the "blue" line that is also on the "red" line (use subqueries)

2. For each table provide its foreign keys (if it has some)