Algorithms and Data Structures, Academic Year 2016/2017

International Bologna Master in Bioinformatics

January 22, 2018

Please complete the following exercises by applying the concepts that have been illustrated to you during the classes. The score associated with each exercise and the expected time for completion is reported in the first line. Do NOT copy/exchange results (the parameters of each exercise are different). Time allowed: 3 hours.

Exercise 0 (2 points): write your name and surname in the first row of all the sheets you use.

Name:___________________________________ Surname:________________________________

Exercise 1 (43 points, 60 minutes): The command *diff* of the *Unix* operating system examines two text files and outputs their differences in terms of rows. For instance, given the two input files *P* and *T* shown in the first two colums, *diff(P, T)* gives the output file *U* shown in the third column:

Message P

```
Questo è il testo originale
alcune linee non dovrebbero
cambiare mai
altre invece vengono
rimosse
altre vengono aggiunte
```

Message T

```
Questo è il testo nuovo
alcune linee non dovrebbero
cambiare mai
altre invece vengono
cancellate
altre vengono aggiunte
come questa
```

Output U

```
- Questo è il testo originale
+ Questo è il testo nuovo
  alcune linee non dovrebbero
  cambiare mai
  altre invece vengono
- rimosse
+ cancellate
  altre vengono aggiunte
+ come questa
```

The problem of designing the pseudo-code for the *diff* command can be solved by a *dynamic programming* algorithm. One can assume that *P* and *T* have, respectively, *m* and *n* rows, and that two rows can be compared in O(1) time, since the number of characters in each row is upper bounded by a constant and each character is coded by a constant number of bits. Define first the recurrence relations giving the optimal sub-structure property of such a dynamic programming algorithm, and then write its corresponding pseudo-code and analyze its complexity.

Name:__________________________________ Surname:_______________________________

Exercise 2 (20 points, 20 minutes): Consider the string "b a b a c a r". Write (by hand) its corresponding:

1) Suffix trie;

2) Suffix tree;

3) Suffix array;

4) Burrows-Wheeler transform;

5) LF mapping.

Name:___________________________________ Surname:________________________________

Exercise 3 (15 points, 20 minutes): please provide the heap tree obtained by the insertion of the following values in this order: 40, 26, 22, 41, 20, 60, 46, 48. Provide the heap tree obtained after each insertion. Only for the final result, provide the heap tree in the array-based implementation. Then provide the ordered set of nodes visited in in-order and post-order visit of the obtained tree.

Final array-based heap:

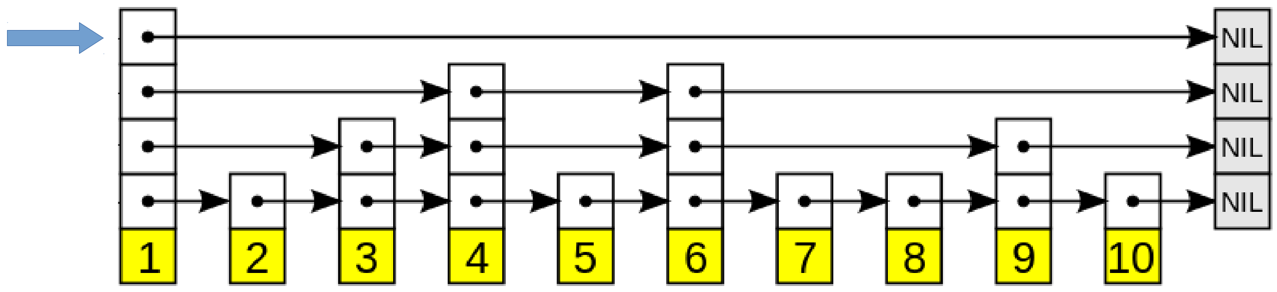
in-order visit:

post-order visit:

Name:___________________________________ Surname:________________________________

Exercise 4 (20 points, 20 minutes): A skip list is a data structure to hold a set of ordered elements. It is made of a list of lists. The last list is a simply linked, ordered list as usual. Each one of the previous lists only point to a subset of the elements of the next list, in the same order. Intuitively, they are fast lanes to go through the elements of the bottom list without traversing each element.

Consider, for example, the skip list below, that holds the values {1,…,10}. The handle for the data structure (the entry point) is a pointer to the topmost element of the stack of lists.



Answer the following questions:

- Execute s(SL,6) on the example above. Imagine that .next allows to move horizontally (in the same list), .down vertically (changes list) and .value reads in O(1) the number stored “all the way dow
  Show the execution by writing in the picture all the pointers L as small arrows.
- What does s(SL,N) do?
- What is the time complexity of s(SL,N) in the worst case? What about the best case? When does the best case happen?

```
s(skiplist SL, integer N) {
L := SL;
found := false;
while (L <> NIL and found == false) {
    if L.value == N then
    found := true else if L.next == NIL
        or L.next.value > N then
    L := L.down;
    else L := L.next;
    }
    return found;
}
```