

# ASSIGNMENT 3

ALGORITHMS & COMPLEXITY (CIS 522-01)

*Javier Arechalde*

February 20, 2018

# 1. Time-series data mining

## 1.1 Problem description

In this problem, we will be given two sequences of events, and we want to find out if the first sequence of events is a subsequence of the second given sequence. The events of the first sequence must appear in the same order in the second sequence too, but they don't necessarily need to be consecutive.

For example, we will be given the following sequence of events.

buy Yahoo, buy eBay, buy Yahoo, buy Oracle

We will also be given this other sequence, which may, or may not contain the first given sequence.

buy Amazon, buy Yahoo, buy eBay, buy Yahoo, buy Yahoo, buy Oracle

These two sequences of events will be named  $S'$  and  $S$ . Given these two sequences of events,  $S'$  of length  $m$  and  $S$  of length  $n$  each containing an event possibly more than once, we want to find in time  $O(m+n)$  if  $S'$  is a subsequence of  $S$ , in the fastest way possible.

## 1.2 Proposed solution

Our proposed solution is to iterate over the two sequences  $S'$  and  $S$  simultaneously, to find if the first sequence is a subsequence of the second one. We will start by taking the first element in  $S'$ , then we will iterate over the  $S$  sequence to see if we can find that element. If we find that element in  $S$ , we move to the next element in  $S'$ , and start iterating to search for this second element in the sequence  $S$ , starting from the  $S$  position right after the position on which we found the first element of  $S'$ . If we can't find that element in  $S$ , that means we iterated over all  $S$  and reached the final position in  $S$  without finding it. In that case we will return that  $S'$  is not a subset of  $S$ . If we find all the elements of  $S'$  in  $S$ , then we will return that  $S'$  is a subset of  $S$ .

### 1.3 Pseudo code

---

**Algorithm 1** Checking if  $S'$  subset of  $S$

---

```
1: We initialize  $i_{pos} = j_{pos} = 0$ 
2: while We didnt reach the end of  $S$  or  $S'$  do
3:   Take  $S(i_{pos})$ 
4:   for  $j$  in range  $j_{pos} \rightarrow length(S)$  do
5:     if  $S'(i_{pos}) == S(j_{pos})$  then
6:       if  $i_{pos} == length(S')$  then
7:          $S'$  is a subsequence of  $S$ 
8:       end if
9:       if  $i_{pos} \neq length(S')$  then
10:         $i_{pos}++$ 
11:         $j_{pos} = j$ 
12:        Break the loop
13:      end if
14:    end if
15:    if  $S'(i_{pos}) \neq S(j_{pos})$  then
16:       $j++$ 
17:    end if
18:  end for
19: end while
```

---

### 1.4 Example

We implemented the algorithm in *Python*, if we run the script, the algorithm will iterate over the two sequences to find if the first sequence is a subsequence of the second sequence. If it is it will print that  $S'$  is a subsequence of  $S$ , and the opposite otherwise.

```
#These are the two sets that we want to compare
s1 = [ 'buy_Yahoo', 'buy_eBay', 'buy_Yahoo', 'buy_Oracle' ]
s2 = [ 'buy_Amazon', 'buy_Yahoo', 'buy_eBay', 'buy_Yahoo', 'buy_Yahoo', 'buy_Oracle' ]

def comp(s1, s2):

    #Getting the length of the two sets
    l1 = len(s1)
    l2 = len(s2)

    #We initialize the operators
    ipos = 0
    jpos = 0
```

```

while(jpos<l2-1):
    el1 = s1[ipos]
    for j in range(jpos,l2):
        el2 = s2[j]

        #If the element is in that position of the list...
        if el1==el2:
            if(ipos==l1-1):
                ipos = ipos+1
                print("S1 is a subset of S2")
                return
            elif(ipos!=l1):
                ipos = ipos+1
                jpos = j
                break #We exit the for loop

        #If we cant find the element in that position of the list
        #We move to the next element
        elif(el1!=el2):
            jpos = j
            continue

    #If the loop above didn't return that S1 is a subset of S2
    # we print the opposite
    print('S1 is not a subset of S2')
    return

comp(s1,s2)

```

## 1.5 Time complexity

In worst case scenario, the last element of  $S'$  will be in the last position of  $S$ , therefore, we would have iterated over both lists to check if  $S'$  is a subset of  $S$ . As the length of  $S'$  is  $m$  and the length of  $S$  is  $n$ , the time complexity of our implementation will be  $O(m+n)$ .

## 2. Competition scheduling

### 1.1 Problem description

In this problem, we want to host a competition. In this competition, we planned to do a mini-thriathlon, in which we will have to swim 20 laps of a pools, then bike for 10 miles, and then run 3 miles.

The competition must follow this rule: no more than one person can be swimming in the pool at a time.

Each one of the contestants has a projected *swimming time*, a projected *biking time*, and a projected *running time*. These are the times that will take each one of the constestants to complete each one of de different sections of the thriatlon.

Our goal is to design an efficient algorithm that produces an schedule whose competition's completion time is as small as possible.

We will name each one of the contestants  $c_i$ , and each one of this contestants will have a  $\{st_i, bt_i, rt_i\}$  that will be the projected time for each of the sections of the thriatlon.

### 1.2 Proposed solution

### 1.3 Pseudocode

### 1.4 Example (Implementation)

Here we should prove that our algorithm is correct too.

### 1.5 Time complexity