Student Name: Duc Tu Luong
Last 3-digit ID: 122

# Homework #3

**Problem 1. Chapter 4, Exercise 4 (Data mining, subsequent problem)**

1. *Model of the problem*
   We want to detect a "pattern" in a sequence. This pattern is a set of events that occur over time. The pattern must be in order but not necessarily consecutively. We begin with a finite collection of possible events and a sequence *S* of *n* of these events. An event may occur multiple times in sequence *S*. we will say that a sequence *S'* is a subsequence of *S* if there is a way to delete certain events so that the remaining events, in order, are equal to the sequence *S'*.

   Give an algorithm that takes two sequences of events *S'* of length *m* and *S* of length *n*, each possibly containing an event more than once, and decides in time *O(m + n)* whether *S'* is a subsequence of *S*.

2. *Overall idea of the algorithm*
   Let *S* be the sequence which we need to find the pattern with *n* elements.
   Let *S'* be the sequence containing pattern with *m* elements.
   We will construct a greedy algorithm that finds the first element in *S* which is the same as the first element in *S'*, match these two elements. After that, we find the first element after first match which is the same as the second element in *S'*, also match these two elements, and continue until we run out of *S* or *S'*.
   Let *p1*, *p2*… denote the pattern match of elements in *S* and *S'*.
   Let *i* and *j* denote the current position of *S* and *S'* respectively.

3. *Pseudo code*
   $Let\ n\ be\ the\ length\ of\ S\ and\ m\ be\ the\ length\ of\ S'$
   $Let\ i = j = 1\ and\ let\ p\ be\ the\ set\ of\ indexes\ of\ matched\ elements$
   $While\ i \leq n\ and\ j \leq m$
   $\quad If\ s_i = s'_j$
   $\quad\quad Let\ p_j = i$
   $\quad\quad i++$
   $\quad\quad j++$
   $\quad Else$
   $\quad\quad i++$
   $If\ j > m$
   $\quad return\ (pattern\ found)$
   $Else$
   $\quad return\ (pattern\ not\ found)$

4. *Step-by-step*
   First we let both i and j, indexes of S and S' respectively, be 1 to start. Then we run the while loop to traverse through S, the condition to break the loop is that either i or j is greater than length of S and S' respectively. The algorithm will compare $s_i \; and \; s'_j$, if it matches, we will increase the indexes of both $i \; and \; j$ by 1, and assign $i \; to \; p_j$, otherwise, we only increase index of $i$ by 1, this will move to next element in S.
   At the end of the algorithm, we compare $j \; to \; m$, if $j > m$, we claim that the pattern S' appears in S, otherwise, it does not.

5. *Time complexity*
   The algorithm takes O(n) time to finish. Each iteration takes O(1) time, because it just needs to compare to the first element in sequence S'.

6. *Proof of correctness*
   We will use the STAY-AHEAD method to prove our algorithm is correct.
   *Case 1:* our algorithm finds the pattern, which in fact, no such pattern is found in the sequence. In this case, because our algorithm run linearly from $s_i \; to \; s_n$ of sequence S (sequence which we need to find pattern S' from) and don't skip any element, therefore, if there is no match between $s_i \; and \; s'_j$, j (index of S') will not increase, only i (index of S) increases. At the end of the $while$ loop (either $i > n$ or $j > m$ will break the loop), we base on the j index to indicate whether pattern S' is found or not, if no such pattern S' appears in S, then j will be less than m and our algorithm will return false.
   *Case 2:* our algorithm does not find the pattern, which in fact, the pattern appears in sequence. This only happens when we match $s'_j$ with $s_i$ with $i > k$ in S such that the pattern may appear between $s_i \; and \; s_k$. This is impossible because our algorithm runs linearly with index i increases by 1, so it makes sure we don't skip any element and always select the earliest match if there is any. Thus, our algorithm will return true if there is any pattern S' found in S.

**2. Chapter 4, Exercise 6 (Competition scheduling problem)**

1. *Model of the problem*
   We have n contestants from $i = 1 \ldots n$. Let $s_i, b_i, and \; r_i$ denote the swimming, biking, and running time of contestant $i$ respectively. The rule is only one contestant uses swimming pool at a time, biking and running can have multiple contestants at a time. Each contestant has projected swimming time, projected biking time and projected running time.
   Let's say that the completion time of a schedule is the earliest time at which all contestants will be finished with all three legs of the triathlon, assuming they each spend exactly their projected swimming, biking, and running times on the three parts. Give an efficient algorithm that produces a schedule whose completion time is as small as possible.

2. *Overall idea of the algorithm*
   We arrange the contestants in descending order b + r, and send them out following this order. We say that this schedule will have the shortest completion time.

3. *Pseudo code*

$Let\ n\ be\ the\ number\ of\ contestants$
$Let\ s, b, and\ r\ be\ the\ swimming, biking, and\ running\ time\ of\ contestants$
$Let\ totalSwim\ be\ the\ total\ swimming\ time$
$Let\ total\ be\ the\ final\ total\ time\ to\ return$
$Let\ P\ be\ the\ sequence\ of\ to-be-send\ contestants\ and\ let\ P\ be\ empty$
$Let\ totalSwim\ be\ 0$
$Let\ total\ be\ 0$
$For\ i = 0\ to\ i = n - 1$
    $Sort\ i\ in\ descreasing\ order\ of\ (b_i + r_i)$
    $Push\ i\ to\ P$
$Endfor$
$For\ i = 0\ to\ n - 1\ in\ P$
    $Let\ out\ i$
    $Let\ totalSwim += s_i$
    $Let\ currentTotal = totalSwim + b_i + r_i$
    $If\ (currentTotal > total)$
        $total = currentTotal$
    $Endif$
$Endfor$
$Return\ P\ and\ total$

4. *Step-by-step*
   First we declare useful variables which we use throughout the whole process. Let $n$ be the total number of contestants. Let $s, b, r$ denote swimming time, biking time and running time for each contestant. Let $totalSwim$ be the total swimming time and $total$ be the total we need to find. Let $P$ be the array containing order of contestants.
   Next we sort our contestants in order of descending of $b + r$, push to $P$. Then, we loop through $P$, let each contestant in $P$ out, add current swimming time of $i$ to $totalSwim$.
   We argue that the minimum completion time possible must be total swimming time of all contestants (since only one contestant can use the pool at a time) plus biking and running time of the last contestant because the last person gets out of pool must complete his/her biking and running in his/her projected time. Thus, we compare $totalSwim + b_i + r_i$ to the current $total$ time we have, if it is greater, then we assign $total$ to the $totalSwim + b_i + r_i$ and go on. Finally, we return $total$ and $P$ as needed.

5. *Time complexity*
   The time complexity for sorting is $O(n^2)$, the time for sending contestants out is $O(n)$, total time complexity is $O(n^2)$.

6. *Proof of correctness*
   We prove this by an exchange argument method. Assume there is an optimal schedule which does not use the above order. Then that optimal schedule must have two contestants $i$ and $j$ such that $i$ is sent out right before $j$, but because it does not use our schedule, hence $b_i + r_i < b_j + r_j$.
   Now we consider the schedule which we swap the orders of $i \; and \; j$. In this swapped schedule, the swimming time for both contestants $i \; and \; j$ are not changed because only one contestant uses the swimming pool at a time, but since $b_i + r_i < b_j + r_j$, $i$ will finish sooner than $j$ finishes in the optimal schedule. Thus, our schedule does not have a greater finished time.
   We continue swapping for all contestants in the optimal schedule, we will not increase finished time. At the end of this swapping method, we will produce a schedule in the order of our algorithm, which does not have a greater finished time than the optimal schedule we assume. Thus, our algorithm is optimal and it finds the shortest finished time possible.

7. *Implementation*
   (Please see source code file)
   Assume we have the following contestants' data:

| | Swimming | Biking | Running |
|---|---|---|---|
| Contestant #1 | 8 | 4 | 10 |
| Contestant #2 | 6 | 7 | 11 |
| Contestant #3 | 7 | 8 | 12 |
| Contestant #4 | 5 | 9 | 10 |
| Contestant #5 | 9 | 10 | 9 |

   After running through Java implementation, the order of contestants to be sent out is $\#3 \rightarrow \#4 \rightarrow \#5 \rightarrow \#2 \rightarrow \#1$
   The shortest completion time is $49$