Assignment 3 (due Monday, March 2, 6:00pm)

Instructions:

- Hand in your assignment using Crowdmark. Detailed instructions are on the course website.
- Give complete legible solutions to all questions.
- Your answers will be marked for clarity as well as correctness.
- For any algorithm you present, you should justify its correctness (if it is not obvious) and analyze the complexity.

In the TRAVEL-PACKING problem, you are given as input a set of $n$ items and their weights $w_i > 0$ of each item $i \in \{1, 2, \ldots, n\}$, along with the capacity $C > 0$ of the suitcases that will be used to pack all the items.

Design a greedy algorithm that determines the minimum number of suitcases required to pack the items, keeping in mind that since the items have numbers, you must place them in order in the suitcase, but that you can choose how many go on a suitcase.

Suppose Alice wants to see $n$ different concerts at a certain theater complex. Each concert is showing on specified dates. For $1 \leq j \leq n$, suppose that concert $C_j$ is showing from day $a_j$ until day $b_j$ inclusive (you can assume that $a_j$ and $b_j$ are positive integers such that $a_j \leq b_j$). The objective is to determine the minimum number of trips to the theater that will be required in order to view all $n$ concerts. For simplicity, assume that it is possible to view any number of concerts in a given day.

3. [10 marks] Dynamic programming
Given an array of non-negative integers $A[1 : n]$ of size $n$, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length (to the right) at that position. For example, $A[i] = 3$ means that you can jump from position $i$ to position $i + 1, i + 2$ or $i + 3$. Design an efficient dynamic programming algorithm to determine if it is possible to reach the last index.

For example, given $A = [3, 2, 0, 0, 4]$ the algorithm should output False, and for $A = [3, 2, 0, 1, 4]$ the algorithm should output True (since it is possible to reach the last index by jumping from index 1 to 4 and then 5).
In order to transform a source string of text $S[1..m]$ into a target string of text $T[1..n]$, we can perform various elementary “single character at a time” transformation operations. Given $S$ and $T$, our goal is to produce a series of transformation operations that change $S$ to $T$. We will use a “scratch pad” array $P$ assumed to be large enough to hold the intermediate results. Initially, $P$ is empty, and at the termination of the algorithm, we should have $P[j] = T[j]$ for $j = 1, 2, \ldots, n$.

We maintain current indices $i$ into $S$ and $j$ into $P$ and the elementary operations are allowed to alter $P$ and these indices. Initially, $i = j = 1$. We will be examining every character in $S$ during the processing, so at the end of the sequence of operations, $i$ will have gone through all values from 1 to $m$ inclusive. There will be four operations:

1. Copy a character from $S$ to $P$ by setting $P[j]$ to $S[i]$ and then incrementing both $i$ and $j$.
2. Replace a character from $S$ by another character $c$, by setting $P[j]$ to $c$, and then incrementing both $i$ and $j$.
3. Skip a character from the source $S$ by incrementing only $i$.
4. Insert a character $c$ into $P$ by setting $P[j]$ to $c$ and then incrementing only $j$.

Here is an example:

<table>
<thead>
<tr>
<th>Operation</th>
<th>$S$</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial string:</td>
<td>algorithm</td>
<td>_</td>
</tr>
<tr>
<td>Copy a</td>
<td>algorithm</td>
<td>a_</td>
</tr>
<tr>
<td>Copy l</td>
<td>algorithm</td>
<td>al_</td>
</tr>
<tr>
<td>Replace g with t</td>
<td>algorithm</td>
<td>alt_</td>
</tr>
<tr>
<td>Skip o</td>
<td>algorithm</td>
<td>alt_</td>
</tr>
<tr>
<td>Copy r</td>
<td>algorithm</td>
<td>altr_</td>
</tr>
<tr>
<td>Insert u</td>
<td>algorithm</td>
<td>altru_</td>
</tr>
<tr>
<td>Copy i</td>
<td>algorithm</td>
<td>altru_i</td>
</tr>
<tr>
<td>Insert s</td>
<td>algorithm</td>
<td>altruis_</td>
</tr>
<tr>
<td>Copy t</td>
<td>algorithm</td>
<td>altruist_</td>
</tr>
<tr>
<td>Replace h with i</td>
<td>algorithm</td>
<td>altruisti_</td>
</tr>
<tr>
<td>Replace m with c</td>
<td>algorithm</td>
<td>altruistic_</td>
</tr>
</tbody>
</table>

This is one way to transform $S = “algorithm”$ to $T = “altruistic”$. There are other ways. The underlined characters establish the value of $i$ and $j$ after an operation.

We will assume that each operation has a cost and the total cost of the transformation would be the sum of all these costs. We are interested in minimizing the cost of the transformation. In our example, the cost of the transformation would be: $5 \cdot \text{cost(Copy)} + 3 \cdot \text{cost(Replace)} + \text{cost(Skip)} + 2 \cdot \text{cost(Insert)}$. 

---

4. [30 marks] Dynamic programming with implementation.
In general the operation costs are established by the application and we would expect that
\( \text{cost}(\text{Skip}) + \text{cost}(\text{Insert}) > \text{cost}(\text{Copy}) \) and \( \text{cost}(\text{Skip}) + \text{cost}(\text{Insert}) > \text{cost}(\text{Replace}) \), for otherwise, we would never bother using \text{Copy} and \text{Replace}. Your program is to accept two
strings \( S \) and \( T \). It will calculate the “edit distance” between \( S \) and \( T \). This is the cost of
the least expensive sequence of transformation operations that transform \( S \) to \( T \). It should
print out the edit distance and the sequence of operations that were required. Analyze the
running time of your program.

**Input and output**

The input consists of three lines:
Line 1 contains 4 integers representing the costs of \text{Copy}, \text{Replace}, \text{Insert}, and \text{Skip}.
Line 2 will be the string \( S \).
Line 3 will be the string \( T \).

**Sample input**

\[
1 \ 2 \ 3 \ 4
\]
algorithm
altruistic

The output will be the edit distance, and the sequence of operations. For example, if the
transformation operations given above for “algorithm” and “altruistic” were, in fact, the ones
producing the minimal cost, then we would expect the following output:

**Sample output**

21
Copy a
Copy l
Replace g with t
Skip o
Copy r
Insert u
Copy i
Insert s
Copy t
Replace h with i
Replace m with c

Note, however, that correct output that corresponds to the given input is

17
Copy a
Copy l
Replace g with t
Replace o with r
Replace r with u
(a) [10 marks] Give an efficient dynamic programming algorithm to determine the edit distance of the given input strings. Justify correctness of your algorithm, provide a pseudocode and analyse the running time. What is the running time of your algorithm?

(b) [20 marks] Implement your algorithm. Please follow Piazza for implementation and submission details.