CISC2006: Algorithm Design and Analysis
Homework 3 **Due Date:** 23:59, 29 March 2023

Rules: Discussion of the problems is permitted, but writing the assignment together is not (i.e. you are not allowed to see the actual pages of another student). You can get at most 100 points if attempting all problems. Please make your answers precise and concise.

1. (15 pts) Complete the tables for dynamic programming for the Longest Common Sub-sequence (LCS) Problem on $X = CGATAC$ and $Y = ACGCTAC$.

   Let $c(i, j)$ represent the length of $\text{LCS}(X_i, Y_j)$, where $X_i = (x_1, x_2, \ldots, x_i)$ contains the first i letters of X and $Y_j = (y_1, y_2, \ldots, y_j)$ contains the first j letters of Y.

   Fill in the missing values of the entries.

rule
① $x_i = y_j$, then
$c(i,j) = c(i-1, j-1) + 1$
② $x_i \neq y_j$, then
$c(i,j) = \max\{c(i-1,j), c(i,j-1)\}$

c	•	A	C	G	C	T	A	C
•	0	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1	1
G	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
T	0	1	1	2	2	3	3	3
A	0	1	1	2	2	3	4	4
C	0	1	2	2	3	3	4	5

In the following, each entry $b(i, j) \in \{\uparrow, \nwarrow, \leftarrow\}$ represents the direction of recursion for $c(i, j)$. Specifically,$b(i, j) = \nwarrow$ if $x_i = y_j$; otherwise, $b(i, j) = \uparrow$ if $c(i-1, j) \geq c(i, j-1)$; $b(i, j) = \leftarrow$ if $c(i-1, j) < c(i, j-1)$.

Fill in the missing values of the entries.

① if same, $\nwarrow$
② if diff, $\leftarrow$ or $\uparrow$

b	•	A	C	G	C	T	A	C
•	0	0	0	0	0	0	0	0
C	0	↑	↖	←	↖	←	←	↖
G	0	↑	↑	↖	←	←	←	←
A	0	↖	↑	↑	↑	↑	↖	←
T	0	↑	↑	↑	↑	↖	↑	↑
A	0	↖	↑	↑	↑	↑	↖	←
C	0	↑	↖	↑	↖	↑	↑	↖

What is the LCS of X and Y?

⇒ C G T A C

value size

2. (20 pts) Complete the tables for dynamic programming for the Knapsack Problem with $C = 12$, on items $\{(0.74, 6), (0.84, 5), (0.37, 4), (0.57, 3), (0.25, 2), (0.10, 1)\}$, where each item i is represented by (value v_i, size s_i).

Let $f(i, b)$ represent the optimal objective on items $\{1, 2, \ldots, i\}$ and with capacity b. We have the following recursion for computing the value of $f(i, b)$ (suppose $b \geq s_i$):

$$f(i, b) = \max\{f(i-1, \underline{b - s_i}) + \underline{v_i}, f(i-1, \underline{b})\}.$$

Fill in the missing values of the entries.

f	1	2	3	4	5	6	7	8	9	10	11	12
1 (0.74, 6)	0.00	0.00	0.00	0.00	0.00	0.74	0.74	0.74	0.74	0.74	0.74	0.74
2 (0.84, 5)	0.00	0.00	0.00	0.00	0.84	0.84	0.84	0.84	0.84	0.84	1.58	1.58
3 (0.37, 4)	0.00	0.00	0.00	0.37	0.84	0.84	0.84	0.84	1.21	1.21	1.58	1.58
4 (0.57, 3)	0.00	0.00	0.57	0.57	0.84	0.84	0.94	1.41	1.41	1.41	1.58	1.78
5 (0.25, 2)	0.00	0.25	0.57	0.57	0.84	0.84	1.09	1.41	1.41	1.66	1.66	1.78
6 (0.10, 1)	0.10	0.25	0.57	0.67	0.84	0.94	1.09	1.41	1.51	1.66	1.76	1.78

In the following, each entry $d(i, b) \in \{Y, N\}$ represents the direction of recursion for $f(i, b)$. Specifically, $d(i, b) = Y$ if $f(i, b)$ is obtained by taking item i into the knapsack. Fill in the missing values of the entries.

d	1	2	3	4	5	6	7	8	9	10	11	12
1 (0.74, 6)	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y
2 (0.84, 5)	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
3 (0.37, 4)	N	N	N	Y	N	N	N	N	Y	Y	N	N
4 (0.57, 3)	N	N	Y	Y	N	N	Y	Y	Y	Y	N	Y
5 (0.25, 2)	N	Y	N	N	N	N	Y	N	N	Y	Y	N
6 (0.10, 1)	Y	N	N	Y	N	Y	N	N	Y	N	Y	N

What is the optimal solution (set of items) of the problem instance? (4) (3) (2)

size = 12, value = 1.78

What is the solution returned by the Greedy algorithm that

1) • picks items with maximum value first:
2) • picks items with minimum size first:
3) • picks items with maximum density first:

1) item {2, 1, 6}, size = 12, value = 1.68

2) item {6, 5, 4, 3}, size = 10, value = 1.29

3) item {4, 2, 5, 6}, size = 12, value = 1.76.

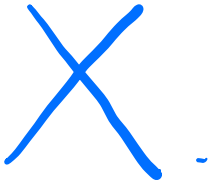
3. (15 pts) Consider the following Longest Increasing Subsequence (LIS) problem.

   Let $A = (a_1, a_2, \ldots, a_n)$ be a sequence of positive numbers. We would like to find the longest subsequence of A whose numbers are strictly increasing. We call this sequence the longest increasing subsequence (LIS) of A.

   For example, if $A = (10, 2, 9, 3, 8, 4, 7, 5, 6)$, then the LIS of A is $(2, 3, 4, 5, 6)$, and its length is 5; if $A = (5, 4, 3, 2, 1)$, then any LIS of A has length 1.

   Give an $O(n^2)$ time algorithm to compute the length of LIS of A. Present your algorithm in pseudo-code and analyze its complexity.

   (Hint: define $f(i)$ as the length of the LIS of $(a_1, \ldots, a_i)$ that ends with number a_i. For example, for $A = (2, 4, 5, 6, 3, 1)$, we have $f(1) = 1$, $f(2) = 2$, $f(3) = 3$, $f(4) = 4$, $f(5) = 2$, $f(6) = 1$.)



4. (10 pts) Compute a table representing the failure function in the Knuth-Morris-Pratt (KMP) algorithm, for the pattern string "CATCATGTTCATC".

   Recall that given a string P with m characters, the failure function f is defined as follows: for all $i \in \{0, 1, \ldots, m-1\}$, $f(i) \in \{0, 1 \ldots, i\}$ is the length of the longest prefix (excluding $P[0, i]$ itself) of $P[0, i]$ that is also a suffix of $P[0, i]$.

P = CATCATGTTCATC.
0 1 2 3 4 5 6 7 8 9 10 11 12

f(0) = 0
f(1) = 0
f(2) = 0
f(3) = 1
f(4) = 2
f(5) = 3
f(6) = 0
f(7) = 0
f(8) = 0
f(9) = 1
f(10) = 2
f(11) = 3
f(12) = 4

5. (10 pts) Let T be a text of length n, and let P be a pattern of length m. Design an algorithm for finding the longest prefix of P that is a sub-string of T and analyze its running time. You get 5 pts if your algorithm runs in $O(nm)$ time; 8 pts if it runs in $O(n \log m)$ time; 10 pts if it runs in $O(n+m)$ time.

len(T) = n, len(P) = m.

→ find longest prefix of P that is sub-string of T.

(note P might be/ not be sub-string of T).

→ pseudo code similar to KMP:

initialize i=0, j=0, max = 0

While (i< n)

- if (T[i] = P[j])
  - if (j= m-1) : return P[0, m-1] and terminate
  - else :
    - i = i+1
    - j = j+1
    - if (j > max) :
      - max = j.
- else :
  - IP (j= 0) : update i = i+1
  - else : j = f(j-1)

output : P[0, max]

6. (30 pts) **Dynamic Programming Algorithm for the Knapsack Problem**.

   In this problem you need to implement the dynamic programming algorithm for the Knapsack problem.

   In the main function, your algorithm need to read an input instance of the Knapsack problem from a file (each item has an ID i, a value v_i and size s_i). Then a function is called to compute a solution for the instance, and output the solution (the set of items) and its objective value (total value of items in the solution).

   You need to implement each of the following 4 algorithms as a function.

   - Algorithm that greedily picks items with maximum value first.
   - Algorithm that greedily picks items with minimum size first.
   - Algorithm that greedily picks items with maximum density first.
   - The Dynamic Programming algorithm the computes the optimal solution.

   Several test cases of input will be provided.

7. (20 pts) **Implementing the Knuth-Morris-Pratt (KMP) Algorithm**.

In this problem you need to implement the KMP algorithm for string matching.

In the main function, your algorithm need to read a text string T (of length n) and a pattern string P (of length m). Then a function is called to decide whether P is a sub-string of T or not. If it is, then output the position i for which $P = T[i, i+m-1]$; otherwise output "P is not a sub-string of T".

You need to implement the following 2 algorithms.

- The $O(nm)$ time Brute-Force algorithm.
- The $O(n+m)$ time KMP algorithm. In the implementation, you need to first compute the failure function in $O(m)$ time, then use the failure function to solve the problem in $O(n+m)$ time.

Several test cases of input will be provided. For each test case, the main function calls the two algorithms to solve the problem, and reports the running time of the algorithms.