Xi'an Jiaotong-Liverpool University 西交利物浦大学

PAPER CODE	EXAMINER	DEPARTMENT	TEL
INT102	Jia WANG	Intelligent Science	9047

2nd SEMESTER 2023/24 EXAMINATIONS (FINAL)

BACHELOR DEGREE – Year 2

ALGORITHMIC FOUNDATIONS AND PROBLEM SOLVING

TIME ALLOWED: 2 Hours

INSTRUCTIONS TO CANDIDATES

READ THE FOLLOWING CAREFULLY:

1. The paper consists of Part I and Part II. Answer all questions in both parts.
2. Answer all questions in Part I using the Multiple-Choice Answer Sheet. Please read the instructions on the Multiple-Choice Answer Sheet carefully and use a 2B pencil to mark the Multiple-Choice Answer Sheet. If you change your mind, be sure to erase the mark you have made. You may then mark the alternative answer.
3. Answer all questions in Part II using the answer booklet.
4. Enter your name and student ID No. on BOTH the Multiple-Choice Answer Sheet and the answer booklet.
5. At the end of the examination, be absolutely sure to hand in BOTH the answer booklet AND the Multiple-Choice Answer Sheet.
6. All answers must be in English.

THIS PAPER MUST NOT BE REMOVED FROM THE EXAMINATION ROOM

PART II (30 Marks)

Question I (8 marks)

Given a string T of length n, represented as T[0] through T[n-1] , and another string P of length m, represented as P[0] through P[m-1] , we say that P is a substring of T if there exists at least one index i ($0 \leq i \leq n-m$) such that the segment T[i] through T[i+m-1] matches P[0] through P[m-1] .

For instance, if T = "ACGTACGGG", then "ACGG" is a substring that appears exactly once in T, "ACG" is a substring that appears twice, and "ACC" is not a substring of T at all.

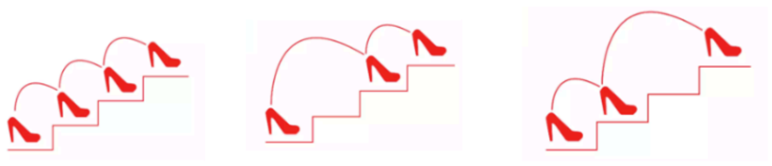
Ensure your algorithm is clear and concise, and that your explanation of the time complexity accurately reflects the computational resources required by your pseudocode.

1. Write pseudocode for an algorithm that determines if the string P appears as a substring in the string T exactly once. ***6***

2. Determine the worst-case time complexity of your algorithm in Big O notation. Provide a brief explanation for your analysis. ***2***

Question II (18 marks)

Stairs-climbing **problem**: there are **n** stairs, a person standing at the bottom wants to climb stairs to reach the **n-th** stair. The person can climb either **1** stair or **2** stairs at a time, the task is to count the number of ways that a person can reach at the top.



Example: As shown in the figure above, starting from the bottom. To reach to the top, we have several options: 1) taking three steps individually, 2) taking two initially and then one, or 3) taking one step and then two.

1. Briefly describe the idea of the dynamic programming technique. ***2***

2. Let F(n) be the number of ways that a person can reach the n-th stairs. For convenience, define F(0)=0, F(1)=1, F(2)=1. Set up a recurrence relation for F(n) (n > 0) that can be used by a dynamic programming algorithm. ***6***

3. For the amount *n* =6, solving the stairs-climbing problem using the relation set in a) **4**

n	0	1	2	3	4	5	6
F(n)	0	1	1				

4. Write pseudocode of the dynamic programming algorithm for solving this problem and determine its time complexity. **6**

Question III (4 marks)

Given any two decision problems *A* and *B*, what is a polynomial time reduction from *A* to *B*? Briefly explain how this technique can be used to prove certain problems are NP- hard **4**

END OF THE PAPER

Q1. 1.
```
Algorithm Find (T[0 ··· n-1] , P[0 ··· m-1] )
    if m > P then
        return False
    count ← 0
    for i ← 0 to n-m do
        number ← 0
        for j ← 0 to m-1 do
            if ~~T[i]~~ T[i+j] = P[j] then
                number++
        if number = m then
            count++
    if count = 1 then
        return True
    return False
```

2. $O(mn)$

If the string P isn't in string T or string P is at the end of string T, the algorithm will run (m-n) times and in each time we need to loop in string T for n times, which is $O((m-n) \cdot n)$ times; Since $m >> n$, $O(mn)$

Q2. 1. We define the sub-problems, and find the recurrence related to them, and use memorized method to solve the sub-problem then solve the original problem.

2. $$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1, 2 \\ F(n-1) + F(n-2) & n > 2 \end{cases}$$

3. $F(3) = 2$ $F(4) = 3$ $F(5) = 5$ $F(6) = 8$

4.
```
Algorithm F(n)
    // Input: n
    // Output: the number of ways
    Inalize A[n+1]
        A[0] ← 1   A[1] ← 1   A[2] ← 1
    for i ← 3 to n do
        A[i] = A[i-1] + A[i-2]
    return A[n]
```

$O(n)$