Assignment I — Structs and Arrays

TA: Jay(jayworking1117@gmail.com)

Deadline: October 17, 11:59 pm, 2024

Problem 1. Caeser Cipher Decryption with Zigzag Recovery (55%)

Description

You will be asked to implement a decryption process to extract the meaningful message from a ciphertext. This involves applying the Caesar cipher decryption and the zigzag pattern concept using structures and arrays to handle multiple test cases. The challenge requires you to perform Caesar cipher decryption first, followed by reading the decoded message using a zigzag pattern to recover the original message. Details on the Caesar cipher decryption process and how to apply the zigzag reading technique are provided in the following sections.

Step 1: Caesar Cipher Decryption

The Caesar cipher is one of the simplest and most well-known encryption techniques. It is a type of substitution cipher where each letter in the plaintext is shifted a fixed number of places forward (or backward) in the alphabet to produce the ciphertext. For example, in Figure 1, when the shift value is 3, every letter in the plaintext is replaced by the letter that is three positions ahead in the alphabet. As a result, the letter 'A' would be replaced with 'D', 'B' would become 'E', and so on.

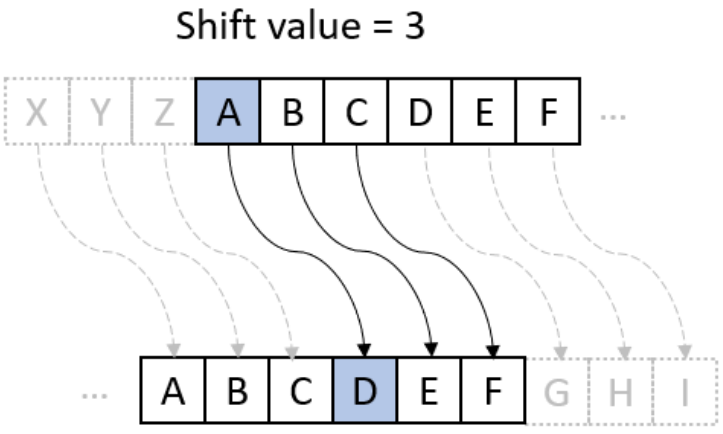

Figure 1: Caesar cipher encryption with shift value is 3

In this assignment, the problem involves a similar concept to the Caesar cipher. For example, in Figure 2, you are given a string (denoted by S) that has been encrypted using the Caesar cipher. Your task is to design a function to decrypt the string into the corresponding plaintext. The requirements for the decryption process are as follows:

- The function should take a string and a shift value (denoted by K) as inputs and return the decrypted string.
- It should handle both uppercase and lowercase letters, while leaving non-alphabetic characters unchanged.

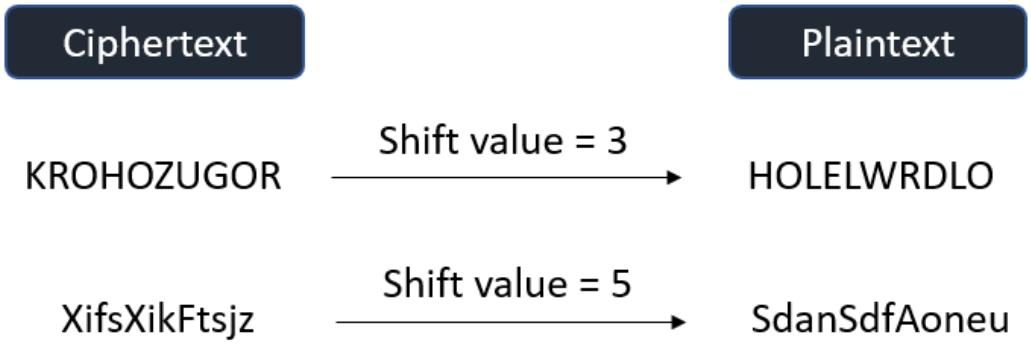

Figure 2: The corresponding plaintext processed by the Caesar cipher decryption function

Step 2: Zigzag Recovery

After completing Step 1, you will obtain a plaintext string. Next, you'll be provided with a row number R, which determines the dimensions of a 2-D array with R rows. The plaintext string should be arranged in this 2-D array. Your task is to create a function that reads and retrieves the data from this array in a zigzag pattern. The function should correctly output the result based on this zigzag reading order. Please see the examples of zigzang recovery process in Figure 3.

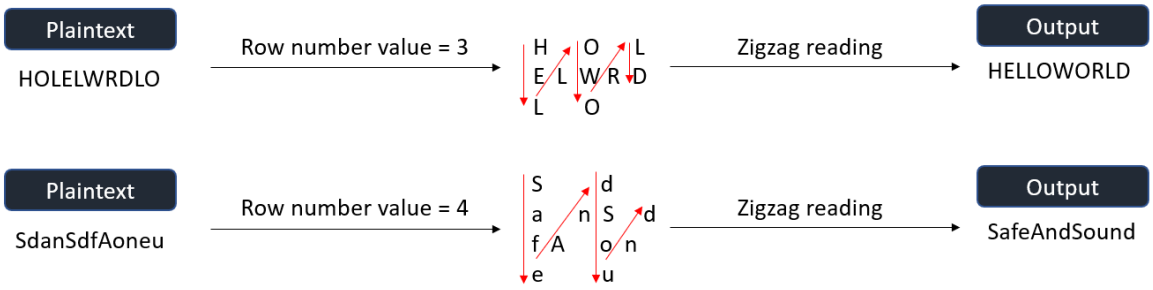

Figure 3: The output of reading the array in a zigzag manner.

Input:

- The first line contains the number of test cases.
- For each test case:
    1. A string S represents the encrypted message, where $1 \leq |S| \leq 500$. The encrypted message S contains uppercase and lowercase english letters [A Za-z], and other non-alphabetic characters(no spaces included).
    2. An integer K denotes the shift value for the Caesar cipher, where $0 \leq K < 26$.
    3. An integer R indicates the number of rows used in the Zigzag cipher, where $1 \leq R \leq 1000$.

Output:

For each test case, output a single line containing the fully decrypted message.

Test case examples

Sample Input

```
3
KROHOZUGOR
3
3
XifsXikFtsjz
5
4
MxByIERYCE
10
4
```

Sample Output

```
HELLOWORLD
SafeAndSound
CRUSHonYOU
```

Problem 2. Optimal Starting Points for Urban Delivery (40%)

Description

Design a solution to identify the possible starting cities for deliveries, ensuring that a delivery driver can complete the delivery route across the entire city and safely return to the starting point. The input will provide the number of cities (denoted by n), the number of goods that need to be collected, and delivered in each city and the maximum capacity of the delivery vehicle (denoted by capacity). Each city is numbered from 0 to n-1 (for example, if n=3, there will be City 0, City 1, and City 2).

If starting with City 0, the driver proceeds in a clockwise direction (City 0 -> City 1 -> … -> City n-1 -> City 0) to check if all goods can be successfully collected and delivered and if they can return to City 0. Then, the process is repeated starting from City 1, and so on for each city.

Since there may be multiple possible starting cities for solution, please return all possible indices of the starting cities in ascending order. If no such starting cities exist, output -1.

Task

Implement a function that takes the following parameters:

- **n**: The number of cities (integer)
- **Capacity**: The maximum capacity of the truck (integer)
- **Collection**: An array of integers representing the number of goods to be collected at each city.
- **Delivery**: An array of integers representing the number of goods to be delivered at each city.

The function should meet the following requirements:

1. When starting from a given starting city, the first city should only be used for collecting goods. Only after completing a full clockwise traversal and returning to the original starting city can the delivery goods be processed.
2. For all other remaining cities, goods should be collected first and checked if the total load exceeds the capacity, before performing the delivery goods.

Input:

- Your program should handle multiple test cases. For each test case, there are following constraints.

    1. The first line contains two space-separated integers, n and capacity, where n is the number of cities (1 ≤ n ≤ 1000) and capacity is the maximum capacity (0 ≤ capacity ≤ 10000).
    2. The second line contains n space-separated integers, representing the collecting array, where 0 ≤ collect[i] ≤ capacity for all i.
    3. The third line contains n space-separated integers, representing the delivery array, where 0 ≤ deliver[i] ≤ capacity for all i.

- The input ends when both n and capacity are 0.

Output:

For each test case, output a single line containing space-separated integers, representing the valid starting points in ascending order.

Test case examples

Sample Input

```
4 6
5 1 3 2
4 3 2 2
5 7
3 2 4 3 5
2 3 1 4 3
4 10
5 5 5 5
5 5 5 5
0 0
```

Sample Output

```
0
-1
0 1 2 3
```

Example:

Given the first test case above, Figures 1-3 show the illustration of the delivery route, accordingly.

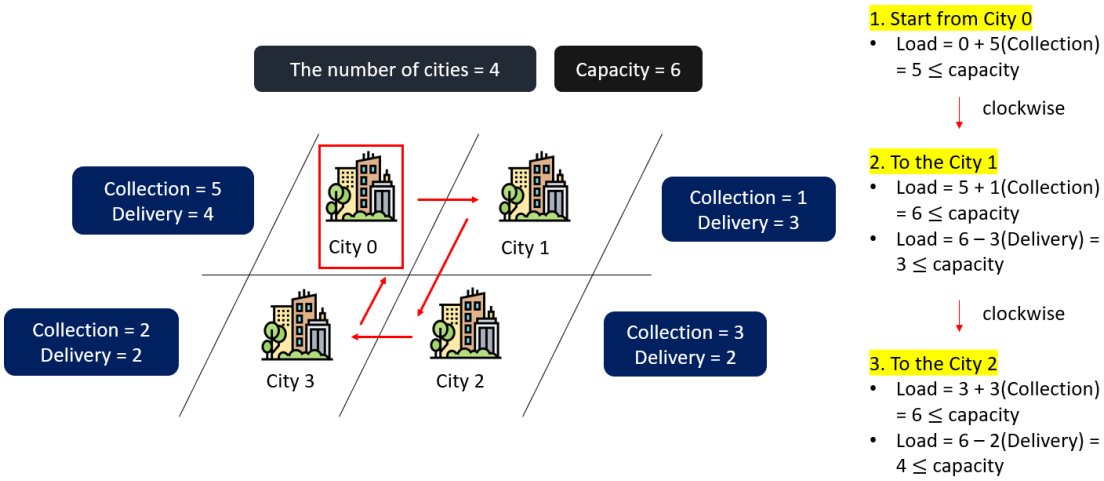

Figure 1. Delivery route example starting from City 0

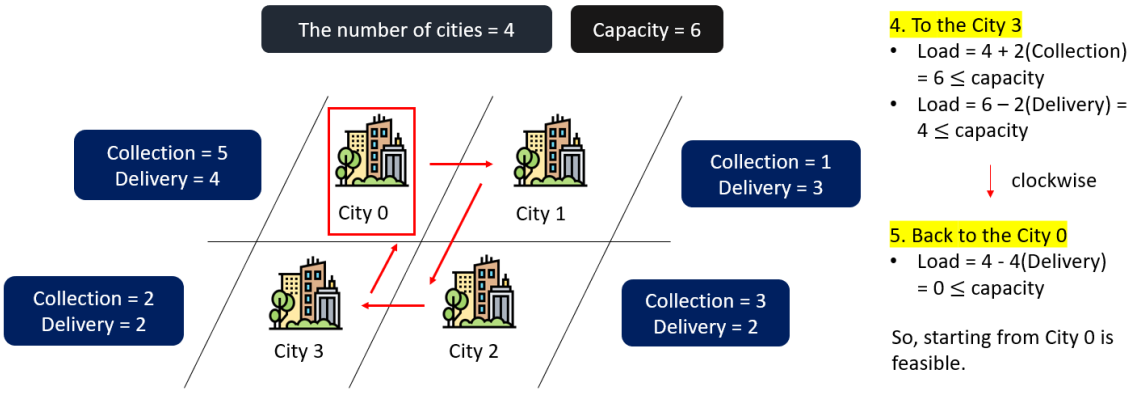

Figure 2. Continuation of the delivery route starting from City 0

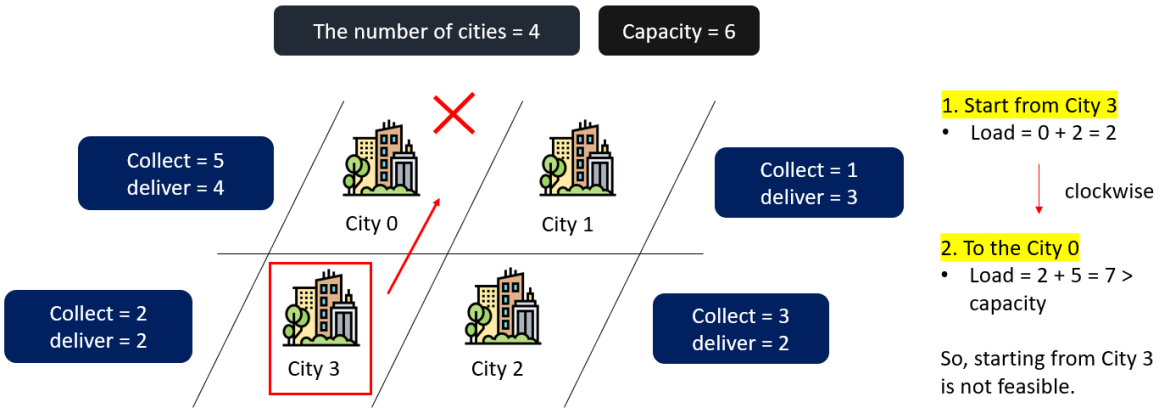

Figure 3. Delivery route example starting from City 3

Readme, comments, and coding style

An indicator for good source code is readability. To keep source code maintainable and readable, you should add comments to your source code where reasonable. A consistent coding style also helps a lot when tracing the source code. For this assignment, please also compose a readme file in *.txt format and name it as “README.txt”. This file should contain a brief explanation of how to use your program. Please remember to have your source code comments and readme file in English.

Submission

To submit your files electronically, login DomJudge website through the following url : http://domjudge.csie.io:54321

Press the submit button and choose the homework questions you want to submit. After submitting your code, DomJudge will give you a result to tell you whether your code is correct or not. **Please note that our code will be evaluated by different sets of test cases.** Please make sure your code can work correctly based on the description above. Additionally, you must compress your code and the README file into a **zip** file and upload it to Ecourse2. Otherwise, you will get zero point.

ATTENTION: Do NOT copy others’ work or you will get a zero.

Grading policies

The TA(s) will mark and give points according to the following rules:

55% - Problem 1
40% - Problem 2
5% - Readme, comments and coding style.