Programming Assignment 2 — Divide and Conquer

The Logistics company *LogTech* has hired you to implement an efficient divide and conquer algorithm that will be of great importance for ground-work operations.

Description

LogTech ground personnel has visited several locations and left a package at every location. The content of the packages is classified information as they have various clients including government institutions, but every package is associated with a code, which is represented as an integer. This code is based on the distance from the starting location to the location where the package is located, such that packages with larger codes are further away from the base (starting point). The locations where packages are, lie on a 2D rectangular grid. Since the base is located at the top left and the codes are computed using the distances to the base, the following properties hold about the grid:

- The package codes are increasing from left to right in every row
- The package codes are increasing from top to bottom in every column

Every location (y, x) is associated with a location label (e.g., in plain text "Rotterdam" or with code words "t153").

Goal: You will exploit the properties of the grid to devise a divide and conquer algorithm that is able to search a given package based on its code. The algorithm will recursively call itself on a subset/subrectangle of the 2D grid in order to find the value. As a result, it will be more efficient than a naive search that scans every cell until the package has been found (such a solution will not grant you a passing grade!). More specifically, your algorithm will return the label of the location where the given package can be found if the package code is valid, otherwise it will return None.

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

(a) The coordinate system

1	10	30
16	20	42
32	47	57

(b) The package codes

Figure 1: An example of a rectangular grid. The left subplot displays the coordinate system. Note that every location is described by an (y, x)-tuple. The right subplot displays the package codes for every location on the grid.

Note: the most efficient solution would be to create a dictionary mapping package codes to their location labels, but this is explicitly not allowed because that is prone to error (hash

collisions) when we have large maps and is prone to being easily hacked by other people. You must therefore implement a divide and conquer algorithm that searches the package location in an efficient way!

Programming

Since the information we will be working with is highly sensitive and cannot be communicated safely over the network, the inputs will be encoded using a simplified Caesar cipher scheme.

Caesar cipher encoding Suppose that we want to encode the following location label "hello". In our simplified Caesar cipher encoding scheme, we do the following. We transform every character into its ordinal/numerical representation (can be done using the built-in Python ord() function). This will yield the following representations:

- h: 104
- e: 101
- l: 108
- o: 111

We shift the numerical representations by adding the Caesar constant to them. Suppose that the Caesar constant is $c = 5$, we would obtain the encoded representations

- h: 109
- e: 106
- l: 113
- o: 116

We then transform every character to its binary (bitstring) representation, which grants us the following binary codes:

- h: 1101101
- e: 1101010
- l: 1110001
- o: 1110100

The encoded version of "hello" then becomes "1101101 1101010 1110001 1110001 1110100". You can decode messages by performing the steps in reverse. Importantly, we encode numbers in the same way by viewing them as strings. Thus, the number "41" would consist of characters "4" and "1", with numerical representations 52 and 49 respectively. The encoded representation of "41" would be "111001 101110". Try to work this out yourself to make sure that you understand the encoding/decoding algorithm.

To implement the divide and conquer algorithm described above, you will implement a class called `IntelDevice`. You will be given the dimensions of the 2D grid (width and height) as well as the locations (encoded) and the package codes (encoded). Moreover, you also obtain the Caesar shift constant c with which the messages have been encoded.

We provided a template program. It consists of two files:

- `divconq.py`: The main file in which you will implement the solution.
- `test_usecases.py`: A Unit Test file with various test cases that help you check whether your implementations of the different functions are correct.

The file `divconq.py` contains various functions that are not implemented yet. Note that you are not required to implement `start_search` as this function is already completed. Read the written documentation about these functions, and implement these functions. Most of these functions require less than 10 lines of code. **Do not change the headers of the functions provided.** You are allowed to add functions yourself, if you feel that that makes it easier. Note, however, that points are deducted if we think that they are unnecessary. Make sure to document these consistently.

The file `test_usecases.py` consists of several test functions. We provided these functions to help programming and testing the code. By running the following command, the unit tests can be executed:

```
python -m unittest test_usecases.py
```

Make sure to have the right packages installed. Do not use other packages than those present in the template. Feel free to add more unit tests. Do not alter the unit tests that are provided. If your program does not succeed on all unit tests that are provided, it is likely that there is still a problem in your code. Make sure that all other unit tests succeed, before submitting the code.

Additionally, also hand in a file named `test_private.py`. In here, you can create additional unit tests to verify the working of your program. Write at least 3 additional unit tests, and hand these in along with the assignment.

Also keep in mind that all unit tests should be able to run within a matter of seconds on any computer.

Report

Write a report in LaTeX(at most 3 pages) using the provided template (see Brightspace), addressing the following points / research questions:

- Introduction: describe the problem and your divide and conquer solution to find a given value. Do not give any code. Describe your approach conceptually. Explain why this approach is more efficient than a linear scan over all cells.
- Draw the search tree of a small example that your approach traverses. Is your approach optimal? (guaranteed to find the package if it exists)? You are allowed to draw this on paper and scan the result.
- Think of a greedy approach to search for a given package code. How well does this greedy approach work? Is your approach optimal? Explain why it is or why it is not. Include an adversarial test-case in your report where the greedy approach does not yield an optimal schedule. If your approach is optimal, you do not need to do this.
- Come up with an experimental design where you compare the *worst-case* running time of a naive linear scan and your divide and conquer approach. Measure the running time in terms of the number of cells that are searched by both methods. Think carefully about how to design your experiments. Perform this experiment for various rectangular grids and

create a plot where you vary the grid size on the x-axis and show the number of scanned cells on the y-axis for both methods. Additionally, you can also display the running time (in seconds) in another plot.

- Summary and Discussion. What was the goal of the assignment? What have you done and observed? (think about: divide and conquer vs linear scan, optimality, running time, greedy approach). Do not write about your personal experience and stories. Keep it scientific and simply summarize the report, making observations about the algorithms.

Work distribution

At the end of the report, include a distribution of the work: who did what? By default, we give both group members the same grade, but if there is a serious imbalance in the workload distribution, we may have to take action. The work distribution does not count towards the page limit.

Submission

Submit your assignment through Brightspace by submitting the following files:

- report.pdf (the report)
- divconq.py (your solution code)
- test_private.py (your unit tests)

The deadline for this assignment is **the 14th of April 2023, 23:59 CET.**