CS102/CE171: Data Structures & Algorithms
Spring 2024 - Homework 01

Department of Computer Science

Dhanani School of Science and Engineering

Habib University

Contents

Introduction

1.1 Instructions

- This homework will contribute 5% towards your final grade.
- The deadline for submission of this homework is Monday,12th February, 9:00 AM.
- The homework must be submitted online via CANVAS. You are required to submit a zip file that contains all the *.py* files.
- The zip file should be named as *HW_01_aa1234.zip* where *aa1234* will be replaced with your student id.
- **Files that don't follow the appropriate naming convention will not be graded.**

1.2 Grading Rubric

Your assignment will be graded according to the rubric given in the Excel file `dsa-spring24-hw1-rubric.xlsx`

1.3 Homework Objectives

Use Nested Lists, Sorting Algorithms, List ADT, Stacks and Queue to solve problems effectively and efficiently.

1.4 Late submission policy

You are allowed to submit late till Tuesday, 13th February, 9:00 AM with a 20% penatly. No late submissions will be accepted afterwards whatsover.

1.5 Use of AI

Taking help from any AI-based tools such as ChatGPT is strictly prohibited and will be considered plagiarism.

1.6 Viva

Course staff may call any student for Viva to provide an explanation for their submission.

1.7 Queries

You can post your Queries on the Homework 01 Discussion on Canvas or visit course staff during their office hours.

1.8 Processing Text Files

For each question, you have to implement a `main()` function that takes a filename as an argument. The `main()` function should read the input from the file and call the required functions.

You can use the following code to read the input from a file:

```python
with open('input.txt') as f:
    lines = f.readlines()

input = []
for line in lines:
    line = line.strip() # remove leading and trailing spaces
    tokens = line.split() # split the line into tokens
    input.append(tokens[0]) # add the first token to the
   input list

main(input) # call the main function
```

Questions

2.1 Matrix

2.1.1 Convolution

An image in the form of a 2D fixed-size array is provided. Your task is to implement a function that processes this image by using the convolution operation. A convolution is the application of a filter (or a kernel) to an input image that modifies the image in a certain way. A kernel can be visualized as a 2D matrix. Its dimensions would be $n \times n$ (typically $n = 3$).

The way to apply the kernel would be to place the center of the kernel on top of the very first pixel of the input image ($row = 0, col = 0$). After placing the filter, multiply each element of the filter with the value of the corresponding image pixel that it is overlapping. Sum all of these products and update the pixel of the image on which the center of the kernel was placed. If any element of the kernel falls outside the image (e.g. for the very first pixel of the image, the first row and the left column of the kernel will lie outside the image), assume the value of the image pixel to be 0. Figure 2.1 visualizes this.

Now slide the kernel along the entire width of the image, and apply convolution over each pixel of the image as described above. For this question, assume that you move the filter by one pixel at each step. On reaching the end of the first row

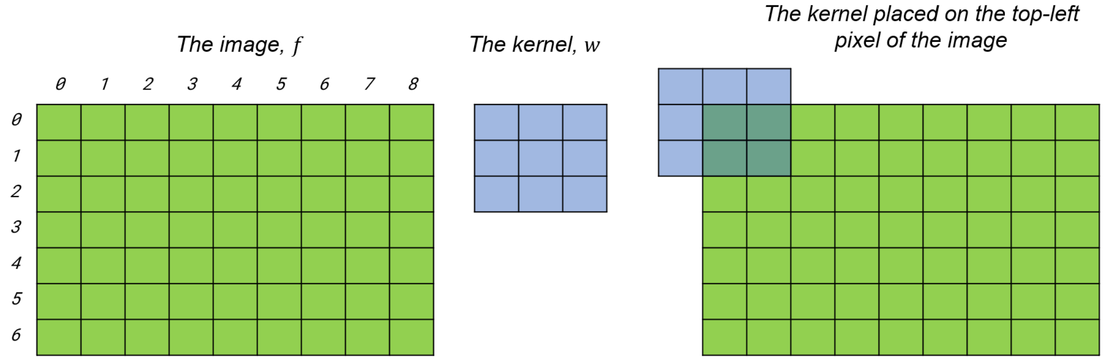

Figure 2.1: The image and the kernel

in the image, move the kernel to the start of the second row of pixels in the input image and repeat the entire process. Once the kernel has been applied on the last row of the input image, convolution is complete. Note that the dimensions of the convolved image are the same as the input image.

The mathematical operation described above for a single pixel of the image can be expressed as follows.

$$g(x, y) = w * f(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i, j) f(x + i, y + j) \tag{2.1}$$

where,

- $g(x, y)$ is the filtered image,
- $f(x, y)$ is the original image, and,
- ω is the kernel.
- Assume that the rows of the kernel range between $-a \le i \le a$, and the columns range between $-b \le j \le b$. So if the kernel is 5×5, the range of i and j would be [-2, 2] (i.e. -2, -1, 0, 1, 2).
- x and y are the coordinates of the input image, where the coordinate of the top-left pixel would be $(0, 0)$.

2.1.2 Function Description

Make a function, `filter_image`, which takes two parameters, `image` and `kernel`. `image` is the input image and is represented by a 2D fixed-size array. `kernel` is the kernel (or filter) applied on the image and is represented by a 1D array. The function will be `filter_image(image, kernel)`.

2.1.3 Input Format

The input for the image and the kernel is available within a text (`.txt`) file, whose path is passed as an input to the `main(...)` function. Within the text file, the input will be given as follows.

- The first line will contain the `rows` and `cols` of the input image.
- The next `rows` lines are the rows of the input image. Each row will have `cols` number of elements.
- The last line will contain the kernel stored as a 1D fixed-size array. The first entry in the array of the kernel is the width of the kernel (recall that the kernel will be a square matrix). The remaining entries correspond to the value of each element in the kernel, arranged in a **column-major order**.

- The input in the text file is as follows.

```
5 5
10 10 10 10 10
20 20 20 20 20
80 80 80 80 80
60 60 60 60 60
70 70 70 70 70
3 3 4 5 2 3 4 1 2 3
```

Note that since the elements of the kernel are arranged in a column major order, the kernel can be visualized as follows.

```
[
    [3, 2, 1],
    [4, 3, 2],
    [5, 4, 3]
]
```

2.1.4 Output

The output of `filter_image` would be a 2D array (which is the result of the convolution).

```
[
    [190,  330,  330,  330,  250],
    [690,  1200, 1200, 1200, 910],
    [880,  1560, 1560, 1560, 1200],
    [1030, 1860, 1860, 1860, 1450],
    [530,  990,  990,  990,  790]
]
```

2.1.5 Note

You can create an additional matrix to store the result of the convolution. However, no additional data structures can be used.

A consequence of this (although, possibly, not the only consequence) is that you cannot create a matrix for the kernel; you have to come up with a 2D array to 1D array mapping (let's call it h) that would allow you to access the correct element in the kernel (given the row and column of interest) from within the given 1D array representation. Note that the first element of the kernel is located at $index = 1$ in the given 1D array.

$$h(row, col) = 1D\_index \tag{2.2}$$

2.2 Alien

The dedicated team of scientists at the Non-Human Communication Center (NHCC) achieved a groundbreaking milestone by establishing communication with extraterrestrial beings. These mysterious aliens have been sending messages, and our scientists have devised a method to receive these transmissions using an array of satellite dishes. Each dish has been meticulously assigned a sequence number, a positive integer, to organize and understand the incoming messages.

The signals arrive in abundance, propelled at a remarkable velocity that challenges the researchers to keep up with the influx. After countless hours of tireless work, the scientists observed a pattern within the received messages. It was as if an intricate code or language was embedded in the transmissions, revealing layers of information that sparked curiosity and excitement within the research facility.

The team of scientists has requested your assistance in developing a program that can help them organize and understand the messages. The pattern observed by scientists is as follows:

- Each message is composed of a sequence number S and a string of characters (M) forming the actual message.
- The first message is assigned a random number R.
- If the sequence number of the next message is less than the messages received so far, then the message is assigned a number $R - 1$.
- If the sequence number of the message is greater than the messages received so far, then the message is assigned a number $R + 1$.
- If a message is received with a sequence number that is between the minimum and maximum sequence numbers of the messages received so far, then the message should be discarded.
- The message with a negative sequence number is a special message used to discard the last received message.
- The message with sequence number zero is a special message. It is used to indicate the end of the communication.
- Returns the entire communication by combining all messages in a list in the increasing order of their assigned numbers R.

Computer scientists at NHCC have chosen to develop an Alien ADT for this purpose, relying on a Dequeue (an extended List ADT) to store the messages. The Alien ADT incorporates the following operations and utilizes a Dequeue implemented through a fixed-size circular array. Your implementation should provide the following operations:

1. `void add(seq: int, msg: str, alienList: dict)`: Adds a message as per the given rules.
2. `void delete(seq: int, msg: str, alienList: dict)`: Discard the last received message.
3. `void get_messages(alienList: dict)` → `list`: Returns the messages in the increasing order of their assigned numbers.

Both functions `add` and `delete` should run in constant time. The function `print` should run in linear time. You are required to implement the Alien ADT using a Dequeue implemented through a fixed-size circular array. You are not allowed to use any other data structure. You are also not allowed to use any built-in data structures provided by the programming language you are using.

2.2.1 Input Format

You are provided the list of messages in the following format:

```
<seq> <msg>
```

The sequence number and the message are separated by a space. The sequence number is an integer. The message is a string of characters. The sequence number zero indicates the end of the communication. The conversation should be printed at the end where each message is separated by space. The messages are provided in no particular order.

Sample Input

```
10 Hello
20 world
30 !
-1 3
15 Goodbye
0
```

Sample Output

```
Hello World
```

Explanation: Let's assume that the first message is assigned a random number, say 5. Therefore, each message can be assigned the following values:

- 10 Hello $->$ 5

- 20 world $- >$ 5 + 1 = 6
- 30 ! $- >$ 6 + 1 = 7
- -1 3 $- >$ discarded the last received message
- 15 Goodbye $- >$ ignored

The message with sequence number zero indicates the end of the communication. Therefore, the entire conversation should be printed in the increasing order of their assigned numbers.

2.2.2 Constraints:

- Each conversation will have at most 100 messages.
- A message will have at most 100 characters.

2.2.3 Implementation

You are provided with a template file `alien.py` and `list_adt.py`. You are required to implement the required functions in both files. You are not allowed to change the function signatures. You are also not allowed to use any built-in data structures provided by the programming language you are using. The description of each function is given in the template files as comments.

2.3 Call Center

A call center wants to keep track of records of their calls. The call center has an infinite number of lines but a fixed number of agents i.e. we can receive as many calls as we want but an agent can handle only one call at a time. If all agents are busy, the call is put on hold. The call center wants to keep track of the waiting time for each call. The waiting time is the time between the call being put on hold and the time the call is assigned to an agent.

We decided to accomplish this using a combination of Queue and Priority Queue data structures. The Queue will be used to keep track of the incoming calls and call logs, and the Priority Queue will be used to keep track of the agents.

2.3.1 Priority Queue

The priority queue is similar to the queue data structure. However, the elements in the priority queue also store a priority value. The priority value is used to determine the order in which the elements are removed from the priority queue. The element with the highest priority is removed first. If two elements have the same priority then the element that was inserted first is removed first. The priority queue could be a minimum priority queue or a maximum priority queue.

In a minimum priority queue, the element with the lowest priority is removed first. In a maximum priority queue, the element with the highest priority is removed first.

In our case, the priority value will be the time when the agent will be free. Therefore, the priority queue will be a minimum priority queue. Initially, all agents have the priority of zero i.e. they all are available. Once an agent is assigned a call, then priority is set to the time the call will be ended.

2.3.2 Implementation

You are required to implement the Queue and Priority Queue data structures using a fixed-size circular array. You are not allowed to use any other data structure. You are also not allowed to use any built-in data structures provided by the programming language you are using.

You have to provide the implementation of the following operations for the Queue ADT using a fixed-size circular array. The Queue ADT should be generic i.e. it should be able to store any type of data.

- `void enqueue(queue, item)`: Adds an item to the queue.
- `void dequeue(queue)`: Removes an item from the queue.
- `void peek(queue)`: Returns the item at the front of the queue.
- `bool is_empty(queue)`$- >$ `bool`: Returns true if the queue is empty.
- `bool is_full(queue)`$- >$ `bool`: Returns true if the queue is full.

You also have to provide the implementation of the following operations for the Priority Queue ADT using a fixed-size circular array.

- `void enqueue_priority(priority_queue, item, priority)`: Adds an item to the priority queue with a priority value.
- `void dequeue_min_priority(priority_queue)`: Removes and returns an item from the priority queue with the minimum priority value.
- `void peek_min_priority(priority_queue)`: Returns the item from the priority queue with the minimum priority value.

Note that `is_full()` and `is_empty()` functions of the queue can be used for the priority queue as well.

Call Center Simulation

You have to simulate the call center by implementing the `CallSimulator(callQueue, agentQueue)` function. Here,

- *callQueue*: a queue of calls to process
- *agentQueue*: a priority queue of agents. Initially, all agents are free i.e. having priority 0.

The function will return the list of calls with the caller name, start time, end time, and waiting time. The calls should be in the order they are processed.

To simulate the call center, the `CallSimulator()` function will implement a loop. Each iteration of the loop will simulate one second, and the loop will terminate when all calls are processed. During each iteration, the following tasks are performed:

1. If there are calls with a start time less than or equal to the simulation time, and an agent is available, assign the call to the agent and set the agent's priority to the time the call will end.
2. The agent with the lowest priority will be the first to become available. If the agent's priority is less than or equal to the simulation time, the agent is now free. Remove the agent from the priority queue, and then add it back with the updated priority.
3. After processing the call, record the details in a queue. The record should include the caller's name, start time, end time, and waiting time.
4. If there are calls to process and no agent is available, the call cannot be processed at this time. Therefore, the call is placed on hold, and we move to the next iteration.
5. Increment the simulation time by one second.

main()

You need to implement the `main()` function to read input from the specified file and call the `CallSimulator()` function. The `main()` function should return a Python list containing tuples for the caller name, start time, end time, and waiting time. The calls in the list should be in the order they are processed.

2.3.3 Input

The first line of input contains the names of the agents separated by spaces. The second line contains the number of calls to process. Each subsequent line includes the time the call arrived, the customer name, and the duration of the call, separated by spaces. The arrival time and duration of the call are integer values, and the customer name is a string of characters. The arrival times are in increasing order. However, if two calls arrive at the same time, they will be processed on a first-come, first-served basis.

Sample Input

```
Faisal Maaz Yusra Rabia Hiba
8
0 A 2
0 B 3
0 C 2
0 D 3
0 E 5
0 F 3
0 G 2
10 H 4
```

The first line lists five agents separated by spaces. The second line indicates the number of calls to be processed. The subsequent eight lines contain information about the arrival time, customer name, and duration of each call. The first seven calls arrive at the beginning, while the last call arrives at the 10_{th} second.

2.3.4 Output

The output should be the list of calls with the caller name, start time, end time, and waiting time. The calls should be in the order they are processed.

Sample Output

```
[
    ('A', 0, 2, 0),
    ('B', 0, 3, 0),
    ('C', 0, 2, 0),
    ('D', 0, 3, 0),
    ('E', 0, 5, 0),
    ('F', 2, 5, 2),
    ('G', 2, 4, 2),
    ('H', 10, 14, 0)
]
```

The first five calls are processed at the beginning since we have five agents. The next two calls are handled after a two-second delay because the first and third agents will become available at that time. Both calls have a waiting time of 2, as they arrived at the start but were processed after two seconds. The last call arrives at the 10_{th} second and is assigned to the agent who will be available at that time. Therefore, the waiting time is zero. In this example, only calls from F and G experience waiting time.

2.4 Sorting

Federal elections are going to happen soon, for which the census data needs to be updated, so that it shows the exact reflection of how many people are supposed to be in the voters list. However, due to the shortage of time, it is not possible to get the census done from scratch. The election body decides to use the existing data, and only get the details of the missing people. In order to do this, they have sent a request to every household asking them if there is a change in the number of people living at their address. Each house is assigned a positive number (starting at 1) based on the area, then on the city, and then on the province - all across the country. Once these unique numbers have been assigned, the responses will be sorted based on these unique IDs, and the task of updating the census data for the required addresses will then be assigned to different volunteers.

As there are millions of people, they have decided to use a sorting technique called "Partition and Prevail". This is a sorting technique that filters the data in groups first then applies insertion sort on smaller sets of data, and combines the groups back together again. This results in a faster result, as applying sorting on all the millions of addresses directly will be very time-consuming, as well as difficult to confirm the results. In order to develop the sorting algorithm, they have hired you to help them out.

2.4.1 Algorithm for "Partition and Prevail"

1. First calculate the maximum value of the valid data elements in the array. **For example, if the input array is: [55, 99, 10, 22, 46, 13, 29, 48], the maximum value will be 99**

2. Using the maximum value obtained in Step 1, add 1 to it. Calculate the group size by dividing this updated value by the number of valid data elements in the array, and then take the ceiling value of the result.

   For example, for the array given as example in Step 1, where the maximum was 99, the number of elements in the array was 8. Thus the group size will be calculated as:

   - Group size = $\text{ceil}(\frac{99+1}{8}) = \text{ceil}(\frac{100}{8}) = \text{ceil}(12.5) = 13$

3. Now define a 2D array/matrix of n x n size, where n represents the number of valid data elements in the original array, containing some default values, like None.

   For example, for the given array example in Step 1, the new matrix will be of size 8 x 8 with each cell containing a None value, i.e. something like the following:

None	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None

4. Iterate over the given array to access each element

5. For each element, divide the value by the group size calculated in Step 2, and get its floor value. This will represent the row number of the 2D array in which the element needs to be placed.

   For example, in the array example given in Step 1, the group size was 13, so for the first element, i.e. 55, the resulting row number will be calculated as:

   - **Row number** $= \text{floor}(\frac{55}{13}) = \text{floor}(4.231) = 4$

   This means 55 will be placed in row 4 of the 2D array/matrix defined in Step 4. The elements within the row will be added sequentially, just like in any single-dimensional array.

6. Do the same for all the elements in the given array.

   Thus, for the array example given in Step 1, the resultant matrix will look like as follows:

10	None	None	None	None	None	None	None
22	13	None	None	None	None	None	None
None	None	None	None	None	None	None	None
46	39	48	None	None	None	None	None
55	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
99	None	None	None	None	None	None	None

7. For every row in the updated matrix, apply insertion sort to sort the valid data elements, i.e. the non-None values within each sub-array.

   Thus, the sorted matrix for the array example given in Step 1 will then look like as follows:

10	None	None	None	None	None	None	None
13	22	None	None	None	None	None	None
None	None	None	None	None	None	None	None
39	46	48	None	None	None	None	None
55	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
None	None	None	None	None	None	None	None
99	None	None	None	None	None	None	None

8. Once each row has sorted values, iterate over each row in the matrix calculated in Step 7 to read the valid data items, i.e. non-None values, and add them to the original given array starting from index 0. This will result in a sorted array.

   For instance, in the given array example in Step 1, the final sorted array would like: [10, 13, 22, 39, 46, 48, 55, 99]

2.4.2 Function Description:

Define the following functions to develop the "Partition and Prevail" algorithm:

1. `initialize_matrix` - A fruitful function that takes an integer 'n', as an argument, and returns a 2D array of size n x n with each cell containing None values.

2. `length` - A fruitful function that takes a single dimensional array, 'arr', as an argument, and returns the count of valid data items in it, i.e. the non-None values.

   Remember arrays are sequential/linear structures so all the valid data items will be in sequence together starting from index 0.

3. `get_maximum` - A function that takes an array as argument, and WITHOUT using the built-in functions, calculates and returns the maximum value of the array.

4. `insertion_sort` - A void function that takes a single-dimensional array, 'arr', as an argument, and applies insertion sort on the valid data items in the array, i.e. the non-None values. This is an **in-place function**, meaning the original array that was passed as a reference will be updated with the sorted values. The function **DOES NOT** returns anything useful.

5. `partition_and_prevail` - A void function that takes the array to be sorted, as an argument, and applies the "Partition and Prevail" algorithm to sort the valid data items in the array, as explained in the algorithm steps above. The function **DOES NOT** returns anything, which means the array passed as an argument will be updated within the function.

YOU ARE NOT ALLOWED TO USE ANY ADDITIONAL ARRAYS OR DATA STRUCTURES, EVEN IF IT'S FOR TEMPORARY PURPOSE!!

2.4.3 Constraints:

- Default value in each cell of the fixed size arrays is ALWAYS None.
- The array contains unique, positive integers ONLY.
- DO NOT use any additional arrays and/or data structures, except for the input, single-dimensional array, and the resultant 2D array, i.e. matrix created for calculation purposes.
- Number of elements in the input array 1, i.e. there will always be at least 2 valid data elements in any given input array.
- DO NOT USE ANY BUILT-IN METHODS FOR ARRAYS.
- However, the math library may be used to perform any mathematical computations.

2.4.4 Sample Cases:

1. **Sample Case 1**
   - **INPUT:** [55, 99, 10, 22, 46, 13, 39, 48]
   - **OUTPUT:** [10, 13, 22, 39, 46, 48, 55, 99]
2. **Sample Case 2**
   - **INPUT:** [875, 777, 13, 534, 96, None, None, None, None, None]
   - **OUTPUT:** [13, 96, 534, 777, 875, None, None, None, None, None]
   - **Explanation:** Sorting the data while leaving the invalid items, i.e. None as they are.
3. **Sample Case 3**
   - **INPUT:** [None, None, None, None, None]
   - **OUTPUT:** [None, None, None, None, None]
   - **Explanation:** As the array contains no valid data items, the array remains the same.