CS5001 Project 9 Report

Shang Xiao

April 20, 2022

1 Problem Description

1.1 An abstract data type (ADT) is an organized way to store data with a particular description specifying what kinds of things can be stored as well as what kinds of operations can be done by the ADT. In module 11, we are introduced to two common ADTs, which are stack and queue.

A stack ADT can often be visualized by a pile of plates we often observe in restaurents. When the restaurant staff finishes cleaning the previously used plates and puts them back again for new use, the staff almost always puts the most recent cleaned plate at the top, and the customer almost always takes the top plate. This interesting behaviour we observe at the restaurant is known as the 'LIFO' - 'last in, first out' method. The stack ADT is a fantastic solution for situations where you need to deal with multiple tasks and might get distracted temporarily by some other tasks. You need to temporarily store the most recent task onto a stack and then go deal with the newest task and come back to this most recent task later.

Next, we have our second ADT: queue. If we look at the spelling of the name 'queue', we can easily understand that this type of ADT is very similar to our real-life line-ups, which can often be seen at bus stations or checkouts in superstores. Unlike the stack ADT, the queue ADT takes a 'FIFO' - 'first in, first out' approach. We can see how a queue works precisely by looking at the 'queue.py' file introduced in the module videos, where we dequeue the very first element in our queue and enqueue a new element to replace it. There is an issue when we dequeue something in the queue. Although the element which has been dequeued has been removed, the empty spot still remains, and eventually, we would see an error saying that the queue is full and can no longer take new elements for enque. The best solution to solve this issue is by using a shifting method explained in 'queue-shift.py' and 'queue-circular.py'.

In this project, we will implement our program using the files mentioned in the above paragraphs to build two new applications called brackets and tickets.

The bracket application uses similar logic introduced in the Stack class, and we want to move through the string from left to right, push the open bracket (the left bracket) and pop the close bracket (the right bracket) and verify if the inputting string is valid: (), {}, [], ⟨⟩.

The ticket application uses similar logic introduced in the Queue class, preferably the circular version. We want to simulate a real-life line-up scenario, where we loop our main function 100 times and randomly decide whether 0, 1, or 2 people join the queue each time. Each time we remove someone from the queue, we need to print out which customer has been served (the removing one), and at the end of the loop, we need to print out how many people are still in the queue.

2 Required Task Elements

2.1 class Stack uses the exact same logic as the Canvas module file, which contains the constructor, push(), pop(), and dump() functions. Additional function verify() validate the user input to push any open brackets, pop any close bracket, and verify the bracket that was popped matches the close bracket.

2.2 'Brackets.py' is the verification function for validating the brackets.

2.3 'testBrackets.py' tests possible combinations of brackets from the user and validate them with the required verification that contains (), {}, [], ⟨⟩.

2.4 class Queue also uses the exact same logic as the Canvas module file, particularly the circular buffer version. The circular buffer version effectively solves the issue from normal Queue classes which usually have empty spaces at front of queue that are not claimed.

2.5 I implemented a *for* loop within the 'Tickets.py' file to check the updated queue. We first randomly generate 0-2 people each time to join the queue, and if the queue is not empty, we dequeue the people at the front of the line, otherwise, the queue is empty.

2.6 The 'testTickets' file automatically generates a 'People.txt' file filled with randomlly generated letter combinations to simulate names of the customers.

3 Reflection

3.1 When I first watched the module videos on abstracted data type (ADT), I saw two words: "LIFO" and "FIFO". Immediately I understood the concept of 'stack' and 'queue' because coming from an accounting undergraduate background, these two methods, 'LIFO' and 'FIFO', are not fresh faces to me. In the accounting field, we have perpetual and periodic systems that both uses 'LIFO', 'FIFO', weighted-average, and moving average methods. The big takeaway here is the connection between my prior professional knowledge and the current new material. It amazes me how well these prior 'exposures' to the same concepts were able to help me quickly understand stack and queue in such an

easy way. Again, this amazing learning experience makes me believe subjects are interlinked, and you never know when these 'linked' knowledge will surprise you.

The second point I would like to share here is the algorithmic thinking we have discussed through recent modules. After module dictionaries and sets, we are now introduced to stack and queue. The ordering logic for these two ADTs can be straightforward to understand. By looking forward, I saw we would cover sorting and efficiency in the next module, introducing insertion sort, merge sort, bubble sort, and selection sort. I can see a learning pattern here. By going from stack/queue to sorting, we are again using the 'simple first, then gradually advancing to more complex scenarios, which accurately reflects the learning curve in computer science.

4 Acknowledgements

4.1 Website consulted (lecture notes and documentations can be found in these links):

`https://docs.python.org/3/tutorial/classes.html` - Official Documentation for classes and objects

`https://www.geeksforgeeks.org/constructors-in-python/` - Constructors

Other than course module introductions, external resources on stack methods: `https://www.geeksforgeeks.org/stack-in-python/`

Other than course module introductions, external resources on queue methods: `https://www.geeksforgeeks.org/queue-in-python/`

Website used for debugging codes and loops for each line execution:
`https://pythontutor.com/visualize.html#mode=edit`