CS 313 - Project 1
Stacks and Queues

January 18, 2023

1 Project Overview

For this programming project, we will be learning how to build and use linear data structures like stacks and queues. Stacks and queues are very common data structures in the CS world. In this lab, we will focus on implementing these data structures using the linked list data structure as a base.

2 Programming Requirements

You will need to write three classes. To receive any credit, your classes must follow the specifications below exactly.

2.1 Write the Node class

- **Class Name:** Node
- **Methods:**
  - $\_\_init\_\_(< Node > self, < Generic > data) \rightarrow < Nonetype > None$
    - **Complexity:** $O(1)$.
    - **Valid Input:** Any. The data parameter defaults to *None* if no data is specified.
    - **Error Handling:** N/A
    - **Instance Variables:**
      - $< Generic > data$: This variable contains the data for a given node.
      - $< Node > next$: This variable contains a pointer to the next node.
      - Any other instance variables you want.
    - Any other methods you want.

2.2 Write the Queue class

- **Class Name:** Queue
- **Methods:**
  - $\_\_init\_\_(< Queue > self, < Int > capacity) \rightarrow < Nonetype > None$
    - **Complexity:** $O(1)$.
    - **Valid Input:** An integer from $(0, \infty)$.
    - **Error Handling:** Raises a **QueueCapacityTypeError** if the Queue capacity is of the wrong type. Raises a **QueueCapacityBoundError** if the capacity is negative or 0.
    - **Instance Variables:**
      - $< Node > head$: This variable contains the head of our linked list.

        - $< Node >$ *tail*: This variable contains the tail of our linked list.
        - $< Int >$ *capacity*: This variable contains the total number of items that can be fit into the queue.
        - $< Int >$ *currentSize*: This variable contains the current number of items in the queue.
        - Any other instance variables you want.
  - $enqueue(< Queue > self, < Generic > item) \rightarrow < bool > returnValue$
    - **Complexity:** $O(1)$.
    - **Valid Input:** Any Python object.
    - **Error Handling:** Raises a **QueueIsFull** exception if the *enqueue* method is called when the queue is full.
    - **Notes:** This method will add an item to the queue. It will then return True/False depending on if the enqueue was successful.
  - $dequeue(< Queue > self) \rightarrow < Generic > returnValue$
    - **Complexity:** $O(1)$.
    - **Error Handling:** Raises a **QueueIsEmpty** exception if the *dequeue* method is called on an empty queue.
    - **Notes:** This method will remove an item from the queue and return it.
  - $front(< Queue > self) \rightarrow < Generic/bool > returnValue$
    - **Complexity:** $O(1)$.
    - **Notes:** This method lets the user peak at the item at the front of the queue without deleting it. It will either return the item at the front of the queue or False if the queue is empty.
  - $isEmpty(< Queue > self) \rightarrow < bool > returnValue$
    - **Complexity:** $O(1)$.
    - **Notes:** This method will return True/False depending on if the queue is empty or not.
  - $isFull(< Queue > self) \rightarrow < bool > returnValue$
    - **Complexity:** $O(1)$.
    - **Notes:** This method will return True/False depending on if the queue is full or not.

2.3 Write the Stack class

- **Class Name:** Stack
- **Methods:**
  - $\_\_init\_\_(< Stack > self, < Int > capacity) \rightarrow < Nonetype > None$
    - **Complexity:** $O(1)$.
    - **Valid Input:** An integer from $(0, \infty)$.
    - **Error Handling:** Raises a **StackCapacityTypeError** if the stack capacity is of the wrong type. Raises a **StackCapacityBoundError** if the capacity is negative or 0.
    - **Instance Variables:**
      - $< Node >$ *head*: This variable contains the head of our linked list.
      - $< Int >$ *capacity*: This variable contains the total number of items that can be fit into the stack.
      - $< Int >$ *currentSize*: This variable contains the current number of items in the stack.
      - Any other instance variables you want.
  - $push(< Stack > self, < Generic > item) \rightarrow < bool > returnValue$
    - **Complexity:** $O(1)$.
    - **Valid Input:** Any Python object.

    * **Error Handling:** Raises a **StackIsFull** exception if the push method is called when the queue is full.
    * **Notes:** This method will add an item to the stack. It will then return True/False, depending on if the push was successful.
  - *pop*$(< Stack > self) \rightarrow < Generic > returnValue$
    * **Complexity:** $O(1)$.
    * **Error Handling:** Raises a **StackIsEmpty** exception if the pop method is called on an empty stack.
    * **Notes:** This method will remove an item from the stack and return it.
  - *peek*$(< Stack > self) \rightarrow < Generic/bool > returnValue$
    * **Complexity:** $O(1)$.
    * **Notes:** This method will return the item at the top of the stack without deleting it. It will return **False** if the stack is empty.
  - *isEmpty*$(< Queue > self) \rightarrow < bool > returnValue$
    * **Complexity:** $O(1)$.
    * **Notes:** This method will return True/False depending on if the stack is empty or not.
  - *isFull*$(< Queue > self) \rightarrow < bool > returnValue$
    * **Complexity:** $O(1)$.
    * **Notes:** This method will return True/False depending on if the stack is full or not.

2.4 Miscellaneous:

- If you encounter an undocumented exception, raise an **UndocumentedError** exception.
- You have to build your own linked list data structure. Using a prebuilt linked list, list, or any other data structure will result in a 0 for the assignment.
- Do not import anything in the source file.

3 Submission Requirements:

Submit a single file **p1.py** to canvas.

4 Grading:

The code will be graded by both a human grader and an auto grader. The auto grader uses Python 3.8.

Your work will be graded along three primary metrics: Correctness, Completeness, and Elegance.

- Correctness: (60 points)
  - You wrote the queue as a FIFO and the stack as a LIFO data structure.
  - You wrote the class methods as specified.
  - Your class methods meet the complexity requirements.
  - You utilize a linked list as the base data structure.
  - Your classes are robust and fault-tolerant.
- Completeness (25 points)
  - Program contains 3 classes named: Node, Queue, and Stack
  - Each class contains methods as defined above.

- The method signatures were implemented as specified.

- Elegance: (15 points)
  - Your code is well organized with good use of whitespace.
  - Your code is well documented.
  - You write your code in a readable manner.
  - You use descriptive variable/function/class names.
  - See the programming guide for the full list of expectations.