problem domain

-implement both a Stack and a Queue for data storage mechanism with node

abdelrahman elatrash

BIG-O complexity
time → o(1)
all method has a constant time complexity.
space → o(n)where n is the number of elements in the data structure

abdelrahman elatrash

Stack

- The class should contain the following methods:

1- push : to add a value to the top of stack
2- pop : Removes the node from the top of the stack
3- peek Returns: Value of the node located at the top of the stack
4-is empty : Returns: Boolean indicating whether or not the stack is empty.

test cases Stack

push(2) ⟹ peek() == 2
pop() == 2
is_empty() ==False

Queue

- contain the following methods:

1- enqueue : adds a new node with that value to the back
2- Removes the node from the front of the queue
3- peek : Returns: Value of the node located at the front of the queue
4- is empty : Returns: Boolean indicating whether or no t the queue is empty

test cases Queue

is_empty() ==True
enqueue(1)) ⟹ peek() == 1
dequeue() == 1

Section 4

edge cases

1- stack or queue is empty

algorithm

create a class named as anode with init method

stack

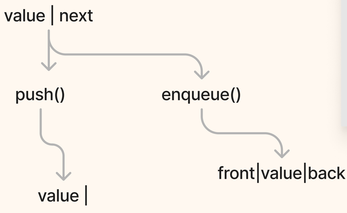
create a class Stack // with method

push add a node value with 1 arg value for the node any
pop , peek , is empty
using assignment operator with if else to check has a value or not

queue

create a class queue // with method
enqueue add node value with 1 arg value for the node any
dequeue , , peek , is empty
using assignment operator with if else to check has a value or not

visualization



verification

node has a value → true
push(1) →

enqueue ('hello') →

Section 5

code

```python
class Node:
    def __init__(self, value=None, next=None):
        self.value = value
        self.next = next
```

```python
from node import Node

class Queue:
    def __init__(self):
        self.front = None
        self.back = None

    def enqueue(self, value):
        new_node = Node(value)
        if self.is_empty():
            self.front = new_node
            self.back = new_node
        else:
            self.back.next = new_node
            self.back = new_node

    def dequeue(self):
        if self.is_empty():
            raise Exception("Queue is empty")
        value = self.front.value
        self.front = self.front.next
        if self.front is None:
            self.back = None
        return value

    def peek(self):
        if self.is_empty():
            raise Exception("Queue is empty")
        return self.front.value

    def is_empty(self):
        return self.front is None
```

```python
from node import Node
class Stack:
    def __init__(self):
        self.top = None

    def push(self, value):
        new_node = Node(value)
        new_node.next = self.top
        self.top = new_node

    def pop(self):
        if self.is_empty():
            raise Exception("Stack is empty")
        value = self.top.value
        self.top = self.top.next
        return value

    def peek(self):
        if self.is_empty():
            raise Exception("Stack is empty")
        return self.top.value

    def is_empty(self):
        return self.top is None
```