Assignment 4 - Stacks & Queues

Due: Friday 28, 2025 @ 9:59 pm
Jayden Cruz

1. (text) Stacking [10 pts] a stack is LIFO, the leftmost element is the bottom, the right most is the top ex: [8, 2] bottom, top

Operations: start: [ ]: empty

1. push(8) 8 inserted [8]
2. push(2) 2 inserted [8, 2]
3. pop(): 2 is removed [8]
4. push(pop()*2): will remove 8 multiply it by 2 [8·2 = 16] -> then it pushed onto the stack [ ] -> [16]
5. push(10) 10 inserted [16, 10]
6. push(pop()/2): will remove 10 from stack then divide it [10/2 = 5] then its pushed on to the stack [16] -> [16, 5]

Final Stack • [16, 5] top, bottom

2. (text) Queueing [10 pts] a Queue is FIFO ex: [8, 2, 4] front, back

Operation: start [ ]: empty

1. push(4) 4 inserted [4]
2. push(pop()+4): 4 is popped then we add 4 [4+4=8] [8]
3. push(8) 8 inserted [8, 8] original 8
4. push(pop()/2): 8 is popped then we div by 2 [8/2=4] now we pop 4. [8, 4]
5. pop() 8 is removed [4]
6. pop() 4 is removed [ ]

Final Queue: [ ] now empty

3. (text) find in deque [10 points]

algorithm:

left_index = 0 // front of the deque
right_index = n - 1 // start from the back

traverse to find element x from both ends of the deque

deque[left_index] = x, return left_index } increment left_index
deque[right_index] = x return right_index } decrement right_index } each step

IF element x not found return -1 or null

We iterate at most $O(\frac{n}{2}) = O(n)$ - deque allows to travers from both sides @ the same time. $O(\frac{n}{2})$

iterate at most $\frac{n}{2}$ times

n = len(deque)
for i in range((n+1)/2):
loop runs half of the input size
it will find it be searching from both ends.

7. (text) Algorithm Analysis

• BalancedBrackets.java

Time Complexity: O(n) isBalanced method has a time complexity of O(n), where n is the length of input string S. Function iterates through the string once, for loops has O(1) operations (adding to or removing from the stack for each char. Each char is processed only once ∴ time complexity is O(n)

Space Complexity: O(n) in the worst case, where all characters in the input are opening brackets, all n characters are stored in the stack. In the best case the sequence is balanced, the stack never grows more than the original input. But still consider the worst case possibility ∴ O(n) is the space complexity where n is the original input.

REST ON NEXT PG.!!

- DecodeString.java

Time Complexity: O(n) decodeString method has time complexity of O(n), where n is the length of the input string s. Function processes each char once, single pass through the string for loop. Inside the loop we have O(1) atomic operations like push() & pop() from the stack. When a] bracket is met, the worst case scenario involves concat a substring multiple times, but since each char is only push() & pop() from the stack a limited # of times we can say O(n)

Space Complexity: O(n) in the worst case bc of stackStr & stackNum stacks will hold substring & #'s for the nested bracket sequences. Also, the StringBuilder (↳ mutable string) used for concating/construct the final output can take up to O(n) space. All data structures hold at most n chars in the worst case, ∴ space complexity is O(n)

- InfixtoPostfix.java

Time Complexity: O(n) infixtoPostfix method has time complexity of O(n), where n is the length of the input infix expression. Each char in the string is processed once in the for loop, O(1) atomic operations like push() & pop(). Worst case, operation may be popped more than once when a closing () or operator sign with lower precedence, but since each element is pushed & popped at most once time complexity O(n)

Space Complexity: O(n) in the worst case the stack may store all operators before they are appended to the result string. The SB (stringbuilder) used to store the final postfix expression also take O(n) space. Both data structures hold at most n characters where n is the input. Overall space complexity is O(n)