Homework 4 Answers

November 13, 2023

Problem 1 (1 point each) is Problem 1 in Discussion Questions in 4.26. of *Problem Solving with Algorithms and Data Structures using Python*. Put the answer in a text file named `hw3_last_first/p1.txt`. Convert the following values to binary using "divide by 2." Show the stack of remainders.

a) 17

ANSWER

$$
\begin{aligned}
17/2 &= 8r1 \text{ push 1, stack: [1]} \\
8/2 &= 4r0 \text{ push 0, stack: [1, 0]} \\
4/2 &= 4r0 \text{ push 0, stack: [1, 0, 0]} \\
2/2 &= 2r0 \text{ push 0, stack: [1, 0, 0, 0]} \\
1/2 &= 0r1 \text{ push 1, stack: [1, 0, 0, 0, 1]}
\end{aligned}
$$

Now pop the stack until empty to obtain the binary number: 10001.

It is not required that the answer be presented exact as above.

Any notion of the stack [1, 0, 0, 0, 1] is adequate.

b) 45

ANSWER

$$
\begin{aligned}
45/2 &= 22r1 \text{ push 1, stack: [1]} \\
22/2 &= 11r0 \text{ push 0, stack: [1, 0]} \\
11/2 &= 5r1 \text{ push 1, stack: [1, 0, 1]} \\
5/2 &= 2r1 \text{ push 1, stack: [1, 0, 1, 1]} \\
2/2 &= 1r0 \text{ push 0, stack: [1, 0, 1, 1, 0]} \\
1/2 &= 1r1 \text{ push 1, stack: [1, 0, 1, 1, 0, 1]}
\end{aligned}
$$

Now pop the stack until empty to obtain the binary number: 101101

c) 96

ANSWER

$$
\begin{aligned}
96/2 &= 48r0 \text{ push 0, stack: [0]} \\
48/2 &= 24r0 \text{ push 0, stack: [0, 0]} \\
24/2 &= 12r0 \text{ push 0, stack: [0, 0, 0]} \\
12/2 &= 6r0 \text{ push 0, stack: [0, 0, 0, 0]} \\
6/2 &= 3r0 \text{ push 0, stack: [0, 0, 0, 0, 0]} \\
3/2 &= 1r1 \text{ push 1, stack: [0, 0, 0, 0, 0, 1]} \\
1/2 &= 0r1 \text{ push 1, stack: [0, 0, 0, 0, 0, 1, 1]}
\end{aligned}
$$

Now pop the stack util empty to obtain the binary number 1100000.

Problem 2 (2 points)

a) Create a `LinkedList` class. It must pass the `hw3/p2/test_linked_list.py`. The class MUST not use any Python built-in or standard library collection class, i.e., do not wrap a list or deque.

ANSWER

I included an implementation of `linked_list.py` in the repository. The answer provided by the student must pass `test_linked_list.py`.

b) Copy the Stack implementation found in the repository

`https://git.cs.olemiss.edu/harrison/csci-356`

in `lecture13and14/stack.py` into your homework directory `hw3_last_first/p2`, rename the class `LinkedListStack` and modify it so that it is implemented using your `LinkedList`. It must pass the unit tests in the repository in the directory `hw3/test_linked_stack.py`. It MUST use your LinkedList. the new Stack class MUST NOT use any Python built-in or standard library collection class, i.e., the `LinkedListStack` class MUST not wrap a `list` or `deque`.

ANSWER

I included an implementation of `linked_list_stack.py` in the repository in `hw3/`. The answer provided by the student must pass `test_linked_list.py`.

Problem 3 (1 point each) Problem 3 in Discussion Questions in 4.26 of *Problem Solving with Algorithms and Data Structures using Python*.

Convert the infix expressions to postfix (use full parentheses).

a) `(A+B)*(C+D)*(E+F)`

ANSWER

The book suggests the following procedure:

1. Create an empty stack called opstack for keeping operators. Create an empty list for output.
2. Convert the input infix string to a list by using the string method split.
3. Scan the token list from left to right.
4. If the token is an operand, append it to the end of the output list.
5. If the token is a left parenthesis, push it on the opstack.
6. If the token is a right parenthesis, pop the opstack until the corresponding left parenthesis is removed. Append each operator to the end of the output list.
7. If the token is an operator, *, /, +, or -, push it on the opstack. However, first remove any operators already on the opstack that have higher or equal precedence and append them to the output list.

When the input expression has been completely processed, check the opstack. Any operators still on the stack can be removed and appended to the end of the output list.

The input is `(A+B)*(C+D)*(E+F)`, which is processed as follows

```
(A+B)*(C+D)*(E+F)  opstack = [], output=""
A+B)*(C+D)*(E+F)   opstack = ['('], output=""
+B)*(C+D)*(E+F)    opstack = ['('], output="A"
B)*(C+D)*(E+F)     opstack = ['(', '+'], output="A"
)*(C+D)*(E+F)      opstack = ['(', '+'], output="AB"
*(C+D)*(E+F)       opstack = [], output="AB+"
(C+D)*(E+F)        opstack = ['*'], output="AB+"
C+D)*(E+F)         opstack = ['*', '('], output="AB+"
+D)*(E+F)          opstack = ['*', '('], output="AB+C"

D)*(E+F)           opstack = ['*', '(', '+'], output="AB+C"
)*(E+F)            opstack = ['*', '(', '+'], output="AB+CD"
*(E+F)             opstack = ['*'], output="AB+CD+"
(E+F)              opstack = ['*'], output="AB+CD+*"
E+F)               opstack = ['*', '('], output="AB+CD+*"
+F)                opstack = ['*', '('], output="AB+CD+*E"
F)                 opstack = ['*', '(', '+'], output="AB+CD+*E"
)                  opstack = ['*', '(', '+'], output="AB+CD+*EF"
                   opstack = [], output="AB+CD+*EF+*"
```

So the postfix generated by the algorithm presented in the book is given by

```
AB+CD+*EF+*
```

b) `A+((B+C)*(D+E))`

ANSWER

Using the analogous procedure given for the conversion for Problem 3a, we generate the following postfix:

```
ABC+DE+*+
```

c) `A*B*C*D+E+F`

ANSWER

The procedure in the book would provide the following result:

```
AB*C*D*E+F+|
```

However, the following is an equivalent postfix notation for the given mathematical expression:

```
ABCD***EF++
```

All variants that produce the same result should be accepted.

Problem 4 (2 points) Use the 'Queue' found in the source code repository in 'hw3/p4/queue.py', which is based on the code in the book in Listing 1 of Section 4.12

a) Create file `perftest_list_queue_a.py` in `hw3_last_first/p4` whose `main` function enqueues n random integers into m Queue objects according to the following pseudocode:

```
create m empty 'Queue' objects and put them in a list named queues.
for n in some range:
    start timer
        for x in queues:
            enqueue the nth random integer into queue x
    stop timer
    divide the elapsed time by m to get an average
    append the average time for an execution of enqueue to a list of times.
```

Using `matplotlib` have your code plot the average execution time for a call to `enqueue()` as a function of n. Vary n at least to 10,000. You may skip n by increments of 10, but if you do then adjust the x-axis accordingly.

ANSWER

I committed my version of `perftest_list_queue_a.py` in the `csci-356` repository in the directory `hw3/p4/`.

The student should deliver a plot that looks linear as in Figure 1.

b) Analyze the performance of the `enqueue()` method using big-O notation. Put your analysis in a file named `hw3_last_first/p4/b.txt`.

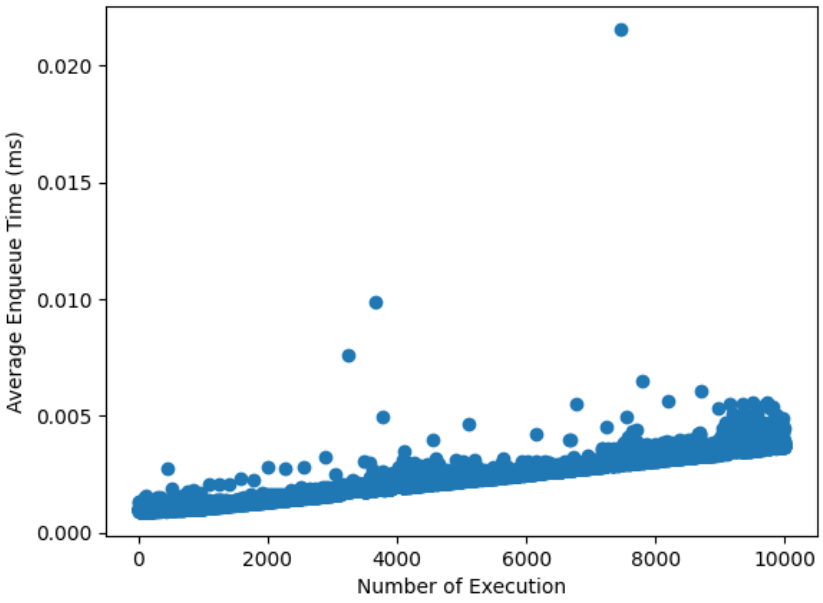

Figure 1: Problem 4: The average run time per call to enqueue for the Queue class.

ANSWER

The Queue uses a Python list as the underlying data structure. The analysis is simple.

```
def enqueue(self, item) -> None:
    self.items.insert(0,item)     // O(n)
```

Since the `insert` is $O(n)$, `enqueue` is $O(n)$.

c) In a file `perftest_list_queue_c.py` Create a variant of the code created for (a) that starts by creating m Queues of the largest length (e.g., $n = 10000$) and then dequeues one item from each list while recording the average time for a dequeue. Using `matplotlib` plot the average execution time for a call to `dequeue()` as a function of n. If it runs too slowly you can start with $n = 1000$. You may skip n by increments of 10, but if you do then adjust the x-axis accordingly.

ANSWER

I committed my version of `perftest_list_queue_c.py` in the `csci-356` repository in the directory `hw3/p4/`.

The student should deliver a plot that looks like performance is $O(1)$ as in Figure 1. In Figure 1, there are a few spikes. I have run this with different seeds, and the spikes appear in the same place and they persist even if garbage collection is turned off. This could be due to shrinking the list's underlying array as the list shrinks. This would be consistent with list having amortized $O(1)$ time complexity.

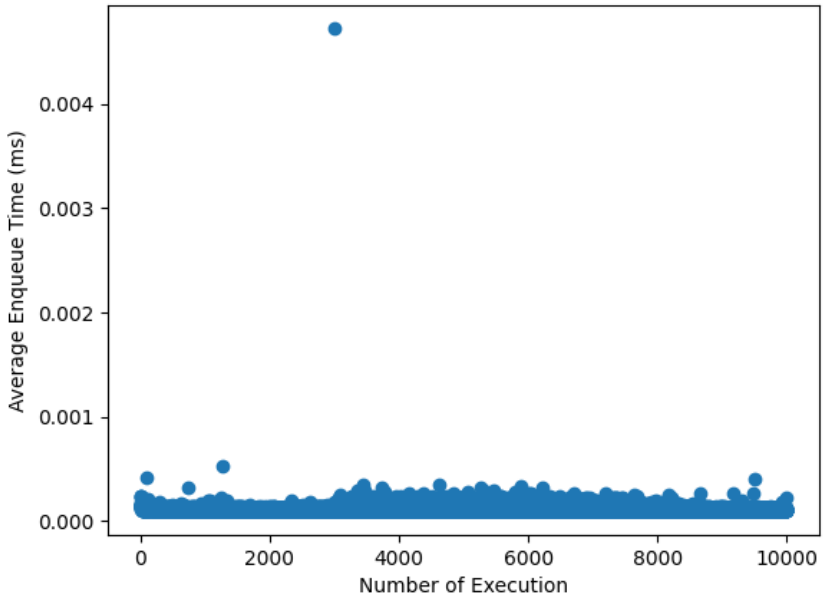

Figure 2: Problem 4: The average run time per call to dequeue for the Queue class.

d) Analyze the performance of the `dequeue()` method using big-O notation and put your analysis in a file named `hw3_last_first/p4/d.txt`.

ANSWER

The Queue uses a Python list as the underlying data structure. The analysis is simple.

```
def dequeue(self) -> object:
  return self.items.pop()      # O(1)
```

Since the `pop` is $O(1)$, `dequeue` is $O(1)$. However, as indicated in the plot and with our prior knowledge of the Python list being a dynamic vector (a.k.a., dynamic array), we should see some spikes as the Queue shrinks which would be consistent with the Python list reallocating to a smaller array and copying the elements from the old array to the new one.

If the student specifies $O(1)$ this is good enough, but kudos to those that state amortized $O(1)$.

Problem 5 (1 point) is Problem 5 in Discussion Questions in 4.26. of *Problem Solving with Algorithms and Data Structures using Python*. I copy the problem below:

> Evaluate the following postfix expressions. Show the stack as each operand and operator is processed.

a) `2 3 * 4 +`

ANSWER

We push every encountered number in the input stream into the stack until we encounter a mathematical operator at such point, we pop the last two numbers, perform the mathematical operation and push the result onto the stack. When there are no more characters in the input stream, the stack should have a depth of 1 and this element is the result.

```
2 3 * 4 +    stack=[]
  3 * 4 +    stack=[2]
    * 4 +    stack=[2, 3]
      4 +    stack=[6]
        +    stack=[6, 4]
             stack=[10]
```

The result of evaluating the expression is 10.

b) `1 2 + 3 + 4 + 5 +`

ANSWER

```
1 2 + 3 + 4 + 5 +    stack=[]
  2 + 3 + 4 + 5 +    stack=[1]
    + 3 + 4 + 5 +    stack=[1, 2]
      3 + 4 + 5 +    stack=[3]
        + 4 + 5 +    stack=[3, 3]
          4 + 5 +    stack=[6]
            + 5 +    stack=[6, 4]
              5 +    stack=[10]
                +    stack=[10, 5]
                     stack=[15]
```

c) `1 2 3 4 5 * + * +`

ANSWER

```
1 2 3 4 5 * + * +    stack=[]
  2 3 4 5 * + * +    stack=[1]
    3 4 5 * + * +    stack=[1, 2]
      4 5 * + * +    stack=[1, 2, 3]
        5 * + * +    stack=[1, 2, 3, 4]
          * + * +    stack=[1, 2, 3, 4, 5]
            + * +    stack=[1, 2, 3, 20]
              * +    stack=[1, 2, 23]
                +    stack=[1, 46]
                     stack=[47]
```

The result of evaluating the expression is 47.

Problem 6 Repeat problem 4, but implement a Queue using a Python deque (a.k.a., doubly-ended queue). Generate the plots and analyze the enqueue and dequeue methods using big-O notation in the same manner. In the plots for (a) and (c) include the plot for the same scenario but using the list implementation of the Queue. This way we can visually compare the performance of the list and deque implementations.

ANSWER

The Queue uses a Python deque as the underlying data structure. The analysis enqueue is simple.

```
def enqueue(self) -> object:
    return self.items.append()      # O(1)
```

Thus the `enqueue` call is $O(1)$.

Your enqueue can append to the front or the rear as long as the deqeue pops from the opposite end.

Similar argument applies for `dequeue`.

```
def dequeue(self) -> object:
    return self.items.popleft()      # O(1)
```

Thus the `deque` call is also $O(1)$.

This time BOTH plots should show $O(1)$ behavior similar to what appear in Figure 3 and Figure 4.

However, as indicated in the plot and with our prior knowledge of the Python list being a dynamic vector (a.k.a., dynamic array), we should see some spikes as the Queue shrinks which would be consistent with the Python list reallocating to a smaller array and copying the elements from the old array to the new one.

If the student specifies $O(1)$ this is good enough, but kudos to those that state amortized $O(1)$.

Problem 7 (1 point) is Problem 11 in Programming Exercises in 4.27. of *Problem Solving with Algorithms and Data Structures using Python*. It must pass the unit tests for `p7/test_html_balance.py`.

> Another example of the parentheses matching problem comes from hypertext markup language (HTML). In HTML, tags exist in both opening and closing forms and must be balanced to properly describe a web document. This very simple HTML document:

```
<html>
   <head>
      <title>
         Example
      </title>
```

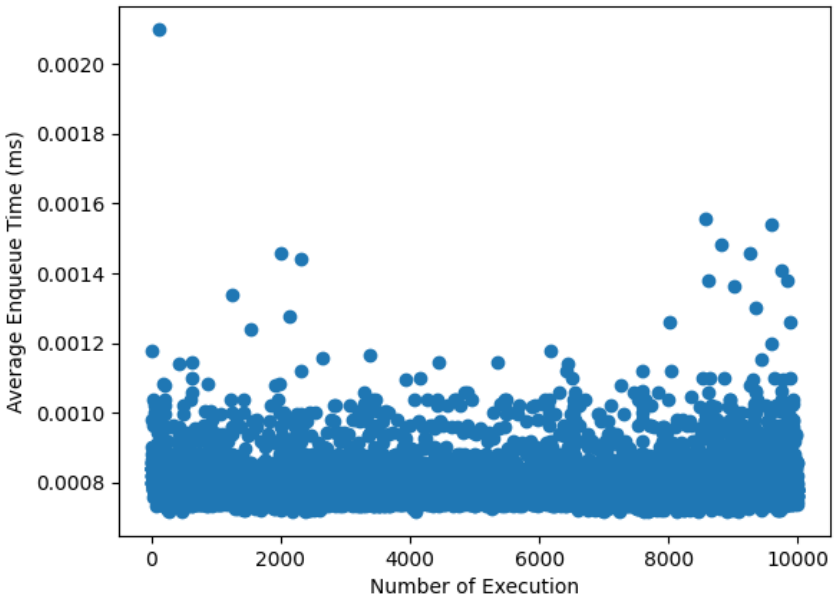

Figure 3: Problem 6: The average run time per call to enqueue for the deque-based Queue class.

```
  </head>

  <body>
     <h1>Hello, world</h1>
  </body>
</html>
```

is intended only to show the matching and nesting structure for tags in the language. Write a program that can check an HTML document for proper opening and closing tags.

ANSWER

I committed my answer to this question in `hw3/p7/test_html_balance.py`.

The student's answer should pass the tests in `test_html_balance.py`.

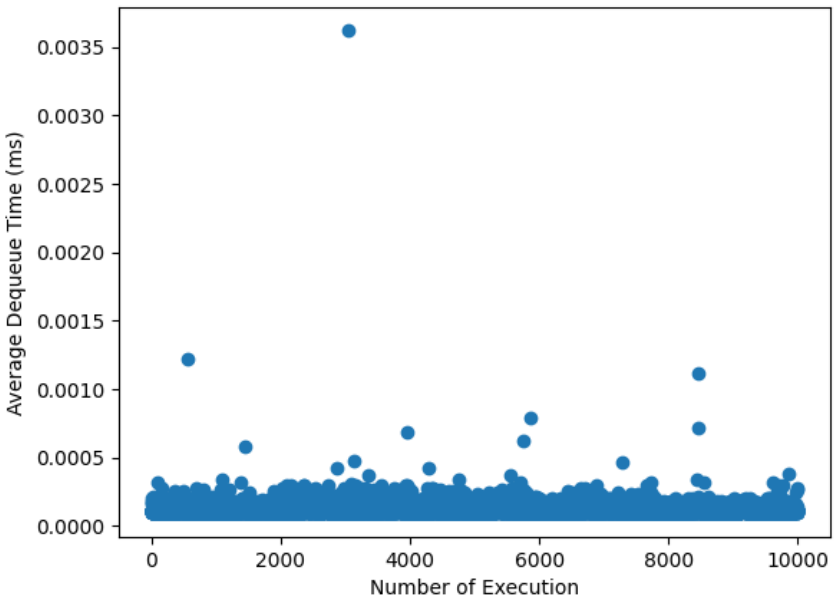

Figure 4: Problem 6: The average run time per call to dequeue for the deque-based Queue class.