HIEU TRUNG NGUYEN

Worst Case Run-Time Analysis of MyQueue

Version 1.0 - January 17, 2015

January 17, 2015

1 INTRODUCTION

This is a worst case runt-time analysis of MyQueue class. In this analysis document, we will review the run-time analysis of the MyQueue class by each method, the constructor and its inner node class. The list includes the inner node class, the constructor, isEmpty(), offer(), poll(), peek(), split() and toString().

2 ANALYSIS

2.1 INNER NODE CLASS

In this section, we will review the run-time for creating an inner node class in MyQueue. Reference to the code is below:

```
private class QueueNode {
        private Student student;
        private QueueNode next;

        public QueueNode(Student student) {
                this.student = student;
                this.next = null;
        }
}
```

1. There is a declaration in this line, the cost of this line is 1.
2. There is a declaration in this line, the cost of this line is 1.
3. There is an assignment in this line, the cost of this line is 1.
4. There is an assignment in this line, the cost of this line is 1.

Let c_0 be the cost of creating an inner node class, then $c_0 = 1 + 1 + 1 + 1 = 4$, we then can see that c_0 is O(1).
Total cost of creating an inner node class equals $c_0 = 4$; c_0 is O(1).

2.2 MyQueue Constructor

In this section, we will review the run-time for constructing an empty MyQueue class. Reference to the code is below:

```
    public class MyQueue {
1       private QueueNode front;
2       private QueueNode back;
3       private int size;

        public MyQueue() {
4           front = null;
5           back = null;
6           size = 0;
        }
    }
```

1. There is a declaration in this line, the cost of this line is 1.
2. There is a declaration in this line, the cost of this line is 1.
3. There is a declaration in this line, the cost of this line is 1.
4. There is an assignment in this line, the cost of this line is 1.
5. There is an assignment in this line, the cost of this line is 1.
6. There is an assignment in this line, the cost of this line is 1.

Let c_1 be the cost of constructing an empty MyQueue class, then $c_1 = 1 + 1 + 1 + 1 + 1 + 1 = 6$, we then can see that c_1 is O(1).
Total cost of MyQueue constructing operation equals $c_1 = 6$; c_1 is O(1).

2.3 isEmpty() Method

In this section, we will review the run-time for the isEmpty() operation. Reference to the code is below:

```
        public boolean isEmpty() {
1               return size <= 0;
        }
```

1. There is a comparison and a return operation in this line, the cost of this line is 2.

Let c_2 be the cost of the isEmpty() operation, then $c_2 = 2$, we can see that c_2 is O(1).
Total cost of the isEmpty() equals $c_2 = 2$; c_2 is O(1).

2.4 offer() Method

In this section, we will review the run-time for the offer() operation. Reference to the code is below:

```
        public void offer(Student student) {
1               QueueNode newNode = new QueueNode(student);
2               if (size == 0) {
3                       front = newNode;
4                       back = front;
                } else {
5                       back.next = newNode;
6                       back = back.next;
                }
7               size++;
        }
```

1. This line creates a new node, with a declaration and an assignment, the cost of this line is $c_0 + 2$.
2. There is a comparison in this line, the cost of this line is 1.
3. There is an assignment in this line, the cost of this line is 1.
4. There is an assignment in this line, the cost of this line is 1.
5. There is an assignment in this line, the cost of this line is 1.
6. There is an assignment in this line, the cost of this line is 1.
7. This line has a single increment operation, the cost of this line is 1.

Let c_3 be the cost of the offer() operation, then $c_3 = c_0 + 2 + 1 + 1 + 1 + 1 + 1 + 1 = c_0 + 8$. We know that c_0 is a constant operation. Therefore, we can see that c_3 is O(1).
Total cost of offer() operation equals $c_3 = c_0 + 8$; c_3 is O(1).

2.5 POLL() METHOD

In this section, we will review the run-time for poll() operation. Reference to the code is below:

```
        public Student poll() {
1               Student currentStudent = front.student;
2               if (front.next != null) {
3                       front = front.next;
                } else {
4                       front = null;
                }
5               size--;
6               return currentStudent;
        }
```

1. There is an a declaration and an assignment in this line, the cost of this line is 2.
2. There is a comparison in this line, the cost of this line is 1.
3. There is an assignment in this line, the cost of this line is 1.
4. There is an assignment in this line, the cost of this line is 1.
5. There is a single decrement operation in this line, the cost of this line is 1.
6. There is a return operation in this line, the cost of this line is 1.

Let c_4 be the cost of the poll() operation, then $c_4 = 2 + 1 + 1 + 1 + 1 + 1 = 7$. We can see that c_4 is O(1).

Total cost of poll() operation equals $c_4 = 7$; c_4 is O(1).

2.6 PEEK() METHOD

In this section, we will review the run-time for peek() operation. Reference to the code is below:

```
        public Student peek() {
1               if (front != null) {
2                       return front.student;
                } else {
3                       return null;
                }
        }
```

1. There is a comparison in this line, the cost of this line is 1.
2. There is a return operation in this line, the cost of this line is 1.

3. There is a return operation in this line, the cost of this line is 1.

Let c_5 be the cost of the peek() operation, then $c_5 = 1 + 1 + 1 = 3$. We can see that c_5 is O(1). Total cost of peek() operation equals $c_5 = 3$; c_5 is O(1).

2.7 SPLIT() METHOD

In this section, we will review the run-time for the split() operation. Reference to the code is below:

```
        public MyQueue split() {
1               MyQueue newQueue = new MyQueue();
2               int n = size;
3               for (int i = 1; i <= n; i++) {
4                       Student currentStudent = this.poll();
5                       if (i % 2 == 0) {
6                               newQueue.offer(currentStudent);
                        } else {
7                               this.offer(currentStudent);
                        }
                }
8               return newQueue;
        }
```

1. This line create a new MyQueue, with a declaration and an assignment, the cost of this line is $c_1 + 2$.
2. There is a declaration and an assignment in this line, the cost of this line is 2.
3. We will consider the loop analysis soon. For this line, there is a declaration, an assignment, a comparison and a single increment operation, the cost of this line is 4.
4. There is a declaration, an assignment and a poll() operation in this line, the cost of this line is $c_4 + 2$.
5. There is a comparison and a modulo operation in this line, the cost for this line is 2.
6. There is an offer() operation in this line, the cost for this line is c_3.
7. There is an offer() operation in this line, the cost for this line is c_3.
8. There is a return operation in this line, the cost for this line is 1.

Let f(n) be the cost of the for loop.

$$\begin{aligned}
f(n) &= \sum_{i=1}^{n}(4+c_4+2+2+c_3+c_3) \\
&= (4+c_4+2+2+c_3+c_3)\sum_{i=1}^{n}1 \\
&= (4+c_4+2+2+c_3+c_3)(n-1+1) \\
&= (4+c_4+2+2+c_3+c_3)\cdot(n)
\end{aligned}$$

Let $c_6 = 4 + c_4 + 2 + 2 + c_3 + c_3$. c_6 is a constant as we know that both c_3 and c_4 are constants. Then the total cost of the for loop is $c_6 \cdot n$.

Let c_7 be the cost of the split() operation, then $c_7 = c_1 + 2 + 2 + (c_6 \cdot n) + 1 = 5 + c_1 + (c_6 \cdot n)$.

Total cost of split() operation equals $c_7 = 5 + c_1 + (c_6 \cdot n)$. We know that c_1 and c_6 are both constants. Therefore, we can see that the total cost of the split() operation or c_7 is O(n).

2.8 size() Method

In this section, we will review the run-time analysis for the size() operation. Reference to the code is below:

```
    public int size() {
1           return size;
    }
```

1. There is a return operation in this line, the cost of this line is 1.

Let c_8 be the cost of the size() operation, then $c_8 = 1$. We can see that c_8 is O(1). Total cost of size() operation equals $c_8 = 1$; c_8 is O(1).

2.9 toString() Method

In this section, we will review the run-time analysis for the toString() operation. Reference to the code is below:

```
public String toString() {
1         if (front != null) {
2                   String queueDisplay = "[" + front.student;
3                   for (QueueNode current = front.next; current != null;
                                        current = current.next) {
4                             queueDisplay = queueDisplay + ",␣" + current.student;
                    }
5                   return queueDisplay + "]";
          } else {
6                   return "[]";
          }
}
```

1. There is a comparison in this line, the cost of this line is 1.
2. There is a declaration, an assignment and a string concatenation operation in this line, the cost of this line is 3.
3. There is a declaration, 2 assignment operations, and a comparison in this line, the cost of this line is 4.
4. There is an assignment and 2 string concatenations in this line, the cost of this line is 3.
5. There is a string concatenation and a return operation in this line, the cost of this line is 2.
6. There is a return operation in this line, the cost of this line is 1.

Let f(n) be the cost of the for loop.

$$\begin{aligned} f(n) &= \sum_{i=1}^{n}(4+3) \\ &= (4+3)\sum_{i=1}^{n}1 \\ &= (4+3)(n-1+1) \\ &= (4+3)\cdot(n) \end{aligned}$$

Let c_9 be the cost of the toString() operation, then $c_9 = (4+3)\cdot n = 7\cdot n$. We can see that c_9 is O(n).

The total cost of the toString() operation equals $c_9 = 7\cdot n$; c_9 is O(n).

3 Conclusion

After the detailed analysis of MyQueue, we concluded that the run-time for constructing an empty MyQueue or creating a new inner node is O(1). For the methods, isEmpty(), size(), offer(), poll() and peek() operations are O(1). While split() and toString() operations are O(n).