Dane Madsen
n10983864@qut.edu.au

Contents

1   NoDVDs Method Design and Analysis

1.1   Algorithm Design

This method calculates the total number of DVDs in a MovieCollection through a in-order node tree traversal using a stack of BTreeNodes. Firstly, it initializes the DVD count to be 0, creates an empty stack of BTreeNodes and starts traversing from the root. Next, an outer while loop starts and continues until the current BTreeNode is null and the stack count is equal to 0. Inside the outer while loop, the inner while loop traverses the left subtree, pushing left child BTreeNodes onto the stack. After traversing the left subtree, the method pops the first movie from the top of the stack, adds its total copies to the running total, and moves to the right child of the popped movie. This process visits each node, cumulatively adding the total copies to the count. Once the traversal finishes, it returns the total DVDs.

ALGORITHM *NoDVDs()*
　// Calculates the total number of DVDs
　// Returns the total number of DVDs in the MovieCollection
　totalDVDs ← 0
　stack ← an empty BTreeNodes stack
　curr ← root node of the MovieCollection
　while *curr* ≠ null **or** *stack* **is not** empty
　　　while *curr* ≠ null
　　　　stack.push(curr)
　　　　curr ← *curr.LChild*
　　　curr ← *stack.pop()*
　　　totalDVDs ← *totalDVDs* + *curr.Movie.TotalCopies*
　　　curr ← *curr.RChild*
　return *totalDVDs*

1.2 Algorithm Analysis

To perform an emperical analysis of the NoDVDs method, I created 20 MovieCollections of exponentially increasing sizes (from 1 to 524388 movies which is roughly close to the amount of movies on IMDb). Then, for each of these MovieCollections, I ran the NoDVDs method 10 times and averaged the time taken to run the method. The results of this analysis are shown in Figure 1.
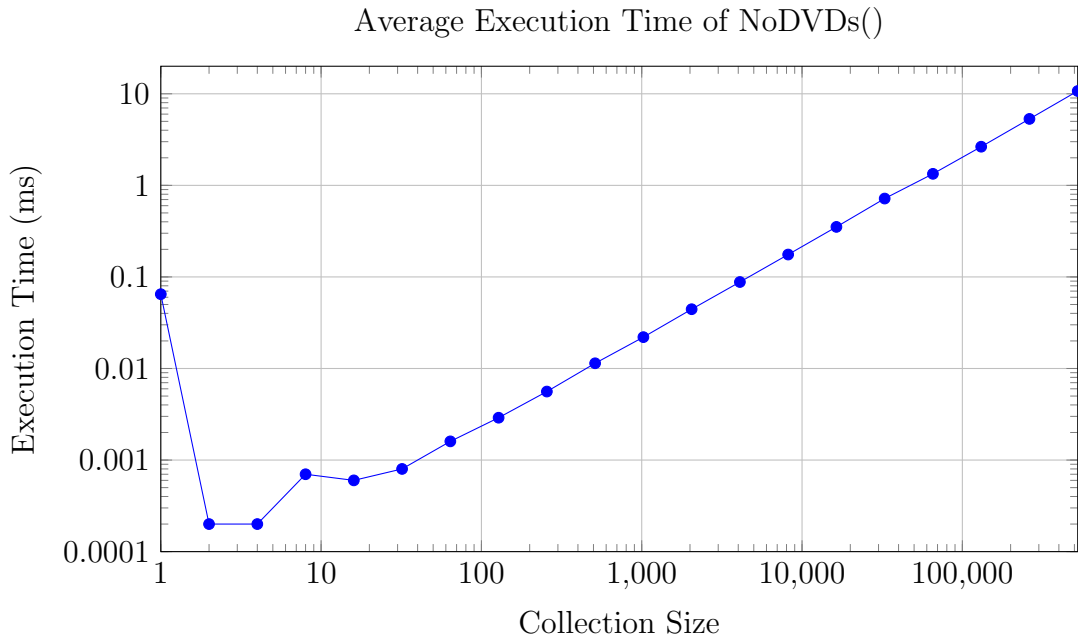

Figure 1: Average Execution Time of NoDVDs()

As Figure 1 shows, though some of the first few MovieCollections have timings that are outliers, after the third MovieCollection (which has 8 values), the average execution times of the NoDVDs method increase linearly. As such the closest efficiency classification for this method would be $O(n)$, where n is the number of movies in the given MovieCollection. This hypothesis is can be supported by ploting the T/n vs. n as shown in Figure 2.
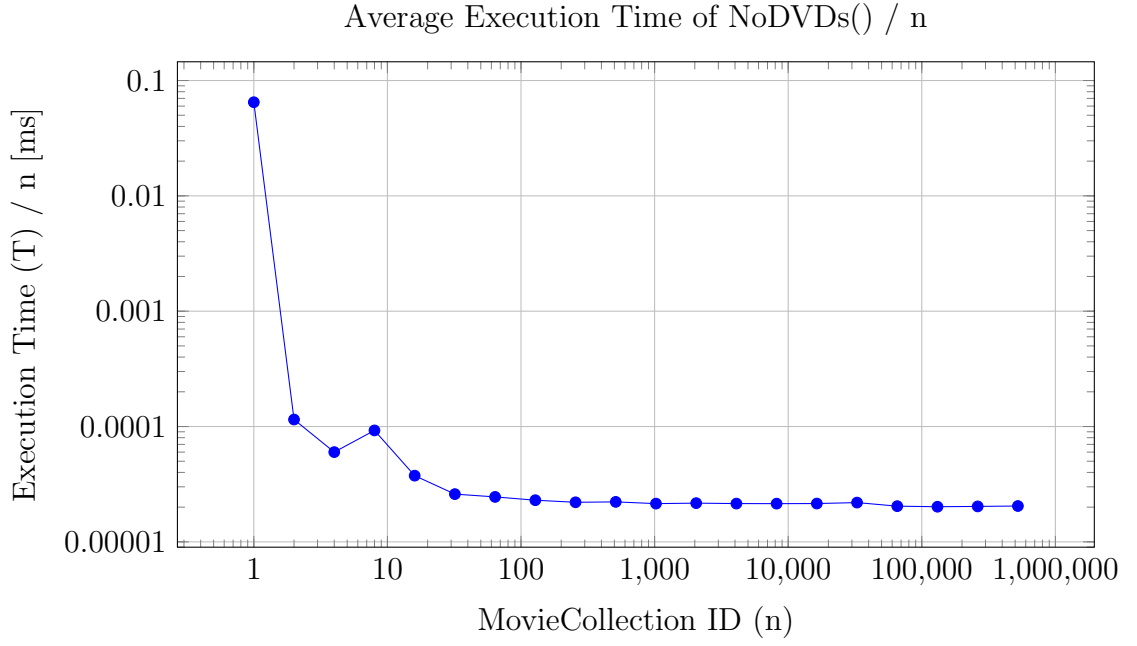
Figure 2: Plot of MovieCollection ID (n) vs. Execution Time (T) / n

Similar to Figure 1, Figure 2 shows that the first few MovieCollections have T/n values that are outliers, but again after the third MovieCollection, the T/n values become quite stable which supports the hypothesis that the algorithm used for the NoDVDs method has an efficiency classification of $O(n)$.

2   Testing

2.1   CompareTo Method Testing

To test the CompareTo method I created three XUnit tests. The first test
ensures that the method returns -1 when the movie CompareTo is being
called on is alphabetically less than the movie being provided to CompareTo
as a parameter. It achieves this by creating two movies, one with the letter
A as a title and the other with the letter B as a title. Then the unit test
calls CompareTo on the first movie (with the letter A) and passes the second
movie (with the letter B) as a parameter and asserts that the returned value
is equal to -1.

The second test ensures that the CompareTo method returns 0 when the
movie CompareTo is being called on is equal to the movie being provided to
CompareTo as a parameter. It achieves this by creating two movies, both
with the letter A as a title. Then the unit test calls CompareTo on the first
movie and passes the second movie as a parameter, then it asserts that the
returned value is equal to 0.

The third and final test ensures that the CompareTo method returns 1 when
the movie CompareTo is being called on is alphabetically greater than the
movie being provided to CompareTo as a parameter. It achieves this by
creating two movies, one with the letter B as a title and the other with the
letter A as a title (the same as the first test only in reverse). Then the unit
test calls CompareTo on the first movie (with the letter B) and passes the
second movie (with the letter A) as a parameter and asserts that the returned
value is equal to 1.

2.2 ToString Method Testing

To test the ToString method I created three XUnit tests. The first test ensures that the method returns the correct string when the movie has only one copy. It achieves this by creating a movie with the parameters: "A", Action, G, 1, 1. Then the unit test calls ToString on the movie and asserts that the returned value is equal to the expected string. This test can also be considered the minimum boundary test.

The second test ensures that the ToString method returns the correct string when the movie has more than one copy. It achieves this by creating a movie with the parameters: "Sophies Choice", Drama, M, 151, 12137123. Then the unit test calls ToString on the movie and asserts that the returned value is equal to the expected string.

The third and final test ensures that the ToString method returns the correct string when the movie has a very large number of copies. It achieves this by creating a movie with the parameters: "ZZZZZZZZ", Western, M15Plus, and the duration and totalcopies both 2147483647. Then the unit test calls ToString on the movie and asserts that the returned value is equal to the expected string. This test can also be considered the maximum boundary test.

2.3 IsEmpty Method Testing

To test the IsEmpty method I created two XUnit tests. The first test ensures that the method returns true when the MovieCollection is empty. It achieves this by creating an empty MovieCollection and asserting that when the IsEmpty method is called on the MovieCollection, it returns true.

The second test ensures that the IsEmpty method returns false when the MovieCollection is not empty. It achieves this by again creating a MovieCollection, but this time populating it with a movie. Then the unit test asserts that when the IsEmpty method is called on the MovieCollection, it returns false.

2.4   Insert Method Testing

To test the Insert method I created four XUnit tests. The first test ensures that the method returns true when the movie being inserted has minimum boundary values. It achieves this by creating a MovieCollection and a movie with the parameters: "A", Action, G, 1, 1. Then the unit test calls Insert on the MovieCollection and passes the movie as a parameter and asserts that the returned value is equal to true.

The second test ensures that the method returns true when the movie being inserted does not already exist in the MovieCollection. It achieves this by creating a MovieCollection and a movie with the parameters: "Sophies Choice", Drama, M, 151, 12137123. Then the unit test calls Insert on the MovieCollection and passes the movie as a parameter and asserts that the returned value is equal to true.

The third test ensures that the Insert method fuctions correctly when the movie being inserted has maximum boundary values. It achieves this by creating a MovieCollection and a movie with the parameters: "ZZZZZZZZZ", Western, M15Plus, and the duration and totalcopies both 2147483647. Then the unit test calls Insert on the MovieCollection and passes the movie as a parameter and asserts that the returned value is equal to true.

The fourth test ensures that the Insert method returns false when the movie being inserted already exists in the MovieCollection. It achieves this by creating a MovieCollection and a movie with the parameters: "A", Action, G, 1, 1. Then the unit test inserts the movie into the MovieCollection once, ensures that the returned value is true, and then inserts the movie into the MovieCollection again and asserts that the returned value is false.

2.5   Delete Method Testing

To test the Delete method I created three XUnit tests. The first test ensures that the method returns true when the movie has successfully been deleted. It achieves this by creating a MovieCollection and a movie with the parameters: "A", Action, G, 1, 1. Then the unit test inserts the movie into the MovieCollection, ensuring that the returned value is true, and then, deletes the same movie from the MovieCollection and asserts that the returned value is again true.

The second test ensures that the Delete method returns false when the

MovieCollection is empty. It achieves this by creating an empty MovieCollection and a movie with the parameters: "A", Action, G, 1, 1. Then the unit test calls Delete on the MovieCollection and passes the movie as a parameter and asserts that the returned value is equal to false.

The third and final test ensures that the Delete method returns false when the movie being deleted does not exist in the MovieCollection. It achieves this by creating a MovieCollection and a movie with the parameters: "A", Action, G, 1, 1. Then the unit test inserts the movie into the MovieCollection, ensuring that the returned value is true, and then, attempts to delete a different movie from the MovieCollection (with the parameters: "B", Action, G, 1, 1) and asserts that the returned value is false.

2.6   NoDVDs Method Testing

To test the NoDVDs method I created three XUnit tests. The first test ensures that the method returns 0 when the MovieCollection is empty. It achieves this by creating an empty MovieCollection and asserting that when the NoDVDs method is called on the MovieCollection, it returns 0. This can also be considered the minimum boundary test.

The second test ensures that NoDVDs returns the correct number of DVDs under normal circumstances. To achieve this the unit test creates a MovieCollection and then populates it with two movies each with 1000 totalDVDs. Then, the unit test asserts that the the returned value of NoDVDs is equal to 2000.

The third and final test ensures that the NoDVDs method returns the correct number of DVDs when the amount of DVDs is very large. To achieve this, the unit test creates a MovieCollection and then populates it with two movies each with half the maximum integer value (2147483647) minus one (1073741823, because the maxint is odd) as their totalDVDs. Then, the unit test asserts that the returned value of NoDVDs is equal to the maximum integer value minus one.