Katrina Slivkoff

Professor Potika

CS 146

13 October 2019

Project 2 Report

This project implements the maximum subarray using 3 different algorithms:

1) Brute Force
2) Divide and Conquer
3) Kadane's Algorithm

A. Brute Force

The brute force algorithm has a running time of O(n^2). This is because it determines all the possible combinations of arriving and departing dates.

i. My Implementation of Brute Force

I made a class called "BruteForce." I then initialized the max, start, and end values as zero. I made an outer loop to choose the starting element of the array. Then, I initialized the sum to zero and made an inner loop. This inner loop set the sum equal to the sum plus the array at a specified element. If the sum was greater than the amx, then the new max was the sum and the start day would be set to the outer loop element and the end date would be set to the inner loop element. Lastly, I returned the max, start, and end values.

B. Divide and Conquer

The divide and conquer algorithm has a running time of O(nlgn). This is because it makes recursive calls to find the maximum subarray.

i. *My Implementation of Divide and Conquer*

I made a class called "DivideAndConquer." I made 3 instance variables named, start, end, and maxSum. I then made a method named "findMaxCrossSubarray." In this method, I first initialized leftSum, sum, and leftMax to zero. I then made a loop to find the leftSum max value. Once I found the leftSum, I set the leftMax to the element of the array where we found the leftSum. The next loop was the same thing but looking for the rightMax value. Once I found the leftSum and the rightSum, I returned them individually and them added together.

The next method I made was the "findMaxSubarray" method. If the high value equaled the low value, then the low, high, and the array at the index of the low is returned. Else, the middle of the array is set to the low and the high divided by two. Then, I initialized the leftSum, rightSum, and crossSum by calling methods recursively. If the leftSum is greater than or equal to the rightSum and the leftSum is greater than or equal to the crossSum, then the leftSum gets

returned. If the rightSum is greater than or equal to the leftSum and the rightSum is greater than or equal to the crossSum, then the rightSum is returned. Else, the crossSum is returned.

C. Kadane's Algorithm

This algorithm has a running time of O(n). It keeps the best positive summation that is kept in maxTemp and saves it in the maxSum.

i. My Implementation of Kadane's Algorithm

I made instance variables for maxSum as zero, arrive as -1, and depart as -1. I then made a method called "findMaxSubArray." In this method, I initialized maxTemp and tempArrive to zero. I then made a loop. This loop looked through the array for the maxSum. First, maxTemp was set to the maxTemp plus the array at a specific element. If the maxTemp was less than zero, maxTemp would be set to zero and arrive would be set to that element plus one. If the maxSum was less than the maxTemp, the maxSum would be set to the maxTemp, depart would be set to that element, and the tempArrivve would be set to arrive. Outside of the loop, arrive would be set to tempArrive.

D. JUnit Test

The JUnit tests were complicated for me to do. I was able to load the maxSumtest, but I was not sure how to test the numbers in there. My solution was making an array and calling the algorithms on those arrays and seeing if they were equal to what they were supposed to be equal to. That worked for me.

The next part of the test was seeing how long each algorithm took to compute at certain sample sizes. I did this by making a random array of a certain sample size. I then called the algorithm on the array and computed how long it took. Of course, brute force took the longest while Kadane's Algorithm took the shortest.

Conclusion

This program taught me why certain algorithms are more efficient and quicker than others. The divide and conquer algorithm was a lot more complex to make than the brute force algorithm, but it was quicker to find the max sum of the array. This means that just because the code is short does not mean that it is the most efficient code.

The JUnit tests were still challenging for me to do. I figured out how to do most of it, but I could not figure out how to implement the test file into my algorithms.