Cabrera, Jen Jade B.
2nd Year BSCS
Algorithms and Complexity

Instructions

Given an array of integers, find the maximum subarray sum.
Input: An array of integers
Output: An integer representing the maximum subarray sum
Time complexity: O(n) (Linear time complexity)
Space complexity: O(1) (Constant time complexity)

Code

```
import java.util.*;

public class SubarraySum {
    public static void main(String[] args) throws Exception {
        int[] array = new int[] {1,2,3,4};
        String[] arrayStringified = (
            Arrays.toString(array)
            .replace("[", "").replace("]", "")
            .replace(" ", "").split(","));

        List<String> elementList = Arrays.asList(arrayStringified);
        List<String> subArrays = getSubarray(elementList);
        subArrays = formatSubarray(subArrays.subList(1, subArrays.size()));
        System.out.println(String.format("Subarrays: %s", subArrays));
        System.out.println(String.format("Maximum subarray sum: %d", getSubarraySum(array)));
    }

    public static int getSubarraySum(int[] array) {
        int highestValue = array[0];
        int secondHighestValue = array[0];

        for (int value : array) {
            if (value > highestValue){
                secondHighestValue = highestValue;
                highestValue = value;
            }
        }
```

```
        return highestValue + secondHighestValue;
    }

    public static List<String> getSubarray(List<String> elements) {
        if (elements.size() == 0){
            return new ArrayList<>(Arrays.asList(""));
        } else {
            String firstElement = elements.get(0);
            List<String> restOfElements = elements.subList(1, elements.size());
            List<String> combinationsWithoutFirst = getSubarray(restOfElements);
            List<String> combinationsWithFirst = new ArrayList<>();

            for (String combination : combinationsWithoutFirst) {
                combination += "." + firstElement;
                combinationsWithFirst.add(combination);
            }

            List<String> combinationToReturn = new ArrayList<>();
            combinationToReturn.addAll(combinationsWithoutFirst);
            combinationToReturn.addAll(combinationsWithFirst);

            return combinationToReturn;
        }
    }

    public static List<String> formatSubarray(List<String> subArrays) {
        for (int i = 0; i < subArrays.size(); i++) {
            String subArray = subArrays.get(i);
            String formattedSubArray = String.format(
                "[%s]", (
                    subArray.replace(".", ",")
                    .substring(1, subArray.length()))
            );
            subArrays.set(i, formattedSubArray);
        }

        return subArrays;
    }
}
```

Step by step procedure and how the recursive main algorithm works

1. First, it checks if the input list is empty. If it is, it returns a new list containing an empty string.
2. If the input list is not empty, it takes the first element from the list and stores it in a variable called "firstElement".
3. It then creates a new list called "restOfElements" which contains all the elements in the input list except for the first one.
4. Next, the method calls itself recursively with the "restOfElements" list as input, which gives a list of all possible subarrays that can be formed from the remaining elements.
5. For each subarray in the list returned from the recursive call, the method concatenates the "firstElement" with a period to create a new subarray that includes the first element, and adds it to a new list called "combinationsWithFirst".
6. Finally, the method creates a new list called "combinationToReturn" by combining the "combinationsWithoutFirst" list (which was returned from the recursive call) with the "combinationsWithFirst" list, and returns it.

By using recursion, this method generates all possible subarrays that can be formed from the input list, starting with the smallest subarray containing a single element and gradually building up to the full list of elements.

Step by step procedure on how getting the maximum subarray sum algorithm works

1. It starts by initializing two variables, "highestValue" and "secondHighestValue", to the first value in the array.
2. Then, it loops through each value in the array using a for-each loop.
3. For each value, it checks if it is greater than the current "highestValue". If it is, then it sets the "secondHighestValue" to the current "highestValue", and the "highestValue" to the current value.
4. By doing this, the algorithm ensures that "highestValue" always holds the highest value seen so far in the array, while "secondHighestValue" holds the second highest value seen so far.
5. Finally, the algorithm returns the sum of "highestValue" and "secondHighestValue".

In simpler terms, this algorithm finds the two largest numbers in an array and returns their sum.

Illustrated step by step procedure

elements = [“a”, “b”, “c”]
getSubarray(elements)
--
Call stack:
    getSubarray([“”])
    getSubarray([“c”])
    getSubarray([“b”,”c”])
    getSubarray([“a”,”b”,”c”])
    main()
--
callStack.pop(getSubarray([“”]) -> return [“”]
callStack.pop(getSubarray([“c”]))
    elements = [“c”]
    firstElement = “c”
    restOfElements = []
    combinationWithoutFirst = [“”]
    combinationWithFirst = [“c”]
    return [“”] + [“c”]
callStack.pop(getSubarray([“b”,”c”])
    elements = [“b”, “c”]
    firstElement = “b”
    restOfElements = [“c”]
    combinationWithoutFirst = [“”, “c”]
    combinationWithFirst = [“b”, “cb”]
    return [“”, “c”] + [“b”, “cb”]
callStack.pop(getSubarray([“a”, “b”, “c”])
    elements = [“a”, “b”, “c”]
    firstElement = “a”
    restOfElements = [“b”, “c”]
    combinationWithoutFirst = [“”, “c”, “b”, “cb”]
    combinationWithFirst = [“a”, “ca”, “ba”, “cba”]
    return [“”, “c”, “b”, “cb”] + [“a”, “ca”, “ba”, “cba”]
callStack.pop(main())
    output = [“”, “c”, “b”, “cb”, “a”, “ca”, “ba”, “cba”]
    endProgram()