Graduate Project
CAS 6O03

The Rod Cutting Problem
Option 2 - Programming Problem 3
Version 0

November 15, 2015

Michael Liut
1132938
McMaster University

Prepared for:
Dr. Antoine Deza
Operations Research
Fall 2015

Contents

1 The Problem

1.1 The Question Restated

Your boss asks you to cut a wood stick into pieces. The stick-cutting company, Machinery Asynchronous Cutting Inc. (MAC), charges money according to the length of the stick being cut. Different selections in the order of cutting can led to different prices. For example, consider a stick of length 10 meters that has to be cut at 2, 4 and 7 meters from one end. There are several choices. One can be cutting first at 2, then at 4, then at 7. This leads to a price of $10 + 8 + 6 = 24$ because the first stick was of 10 meters, the resulting of 8 and the last one of 6. Another choice could be cutting at 4, then at 2, then at 7. This would lead to a price of $10 + 4 + 6 = 20$, which is a better price.

Let n be the number of cutting points of a stick, you are asked to write an $O(n^3)$ algorithm to find out the minimum cost for cutting a given stick.

1.1.1 Input

The first line of each test case will contain a positive number l that represents the length of the stick to be cut. You can assume $l < 1000$. The next line will contain the number n ($n < 50$) of cuts to be made.

The next line consists of n positive numbers c_i ($0 < c_i < l$) representing the places where the cuts have to be done, given in strictly increasing order. e.g.:

```
100
3
25 50 75
```

1.1.2 Output

You have to print the cost of the optimal solution of the cutting problem, that is the minimum cost of cutting the given stick. e.g., the output for the above input should be:

```
200
```

1.2 Analysis

The stick cutting problem, a.k.a. the rod cutting problem, is a dilemma which utilises dynamic programming to make the decision of where to cut a rod by means of minimizing the total cost. Herein I have implemented a dynamic programming solution in Java with the use of the Binary Tree and Huffman Tree data structures. In the process of creating a viable solution I have also optimized the problem's complexity.

For mathematics sake I will demonstrate a few things, lets assume that the rod has a length $\mathcal{L}$ which requires cutting $\mathcal{N}$ number of rods into $\mathcal{P}_{\mathcal{N}}$ pieces. Since the cost $\mathcal{C}$ is produced at the time of each cut, the total cost can be easily illustrated as $\sum_{i=1}^{n-1} \mathcal{C}_i$.

2 The Solution

2.1 Data Structure Implementation

2.1.1 Binary Tree

Due to the nature of the problem, a binary tree is the utmost natural data structure implementation for this dilema. Each time a rod is cut it is seperated into two smaller rods. This is the same with a binary tree data structure, as it can only have at most two child nodes (a left and right node). For all intents and purposes, the entire rod will be refered to as the *parent node* or *root node*. Each time an cut occurs, the parent node has two *children*: $child_1$ (a.k.a. $\mathcal{P}_1$) which contains the left derivation of the rod and $child_2$ (a.k.a. $\mathcal{P}_2$) which contains the right most. Note that: $\mathcal{P}_\mathcal{N} = \mathcal{P}_1 + \mathcal{P}_2$.

2.1.2 Huffman Tree

To minimize the cutting cost, use of the Huffman Tree data structure is imperative as it provides the most optimal schema when programming prefix-free code for lossless data compression. Huffman's algorithm generates a variable-length code table using probability or the frequency of occurences (weight) for each possible value of the source.
The frequency of these characters are given as: $f(i)$, where $i = 1, 2, ..., n$ and the depth of the length of the binary tree: $d(i)$, where $i = 1, 2, ..., n$. This allows us to generate the minimized function from Huffman coding: $\sum_{i=1}^{n} f(i)d(i)$.

2.2 Algorithm Implementation and Complexity

2.2.1 Explanation

It is clear that the stick cutting problem in 1.1 is the inverse problem from the original rod cutting problem, depicted in section 15.1 from *Introduction to Algorithms, 3rd Edition, Cormen et. al., MIT Press.* There is a slight tweak involved with the calculation of the cost, however the total cost remains equivalent; the math here is trivial as the two summations are equal.

2.2.2 Complexity

The implementation of this algorithm gives us a space complexity of $\mathcal{O}(n)$ and a time complexity of $\mathcal{O}(n)^2$.

1. Input size $\mathcal{N}$ is passed via `args`.
2. Initialization contains a loop with $\mathcal{N}$-steps. Nodes decrement by 1.
   (a) Find minimum value (takes at most $\mathcal{N}$-steps).
   (b) All operations here have a time complexity of $\mathcal{O}(1)$

2.2.3 Correctness

From 2.2.1 we know that the total cost is equivalent. For correctness sake, we define the following lemmas:

Lemma 1 *The Huffman Tree contains the minimum sum of weighted path lengths for the given set of children nodes.*

Lemma 2 *Lets assume a and b are siblings (children nodes from the same parent/root) and are the two least frequent characters.*

The definition of a Huffman Tree states that a character with a certain frequency is related to a rod/stick with a length of $\mathcal{L}$. Furthermore, the method merges the two least frequent characters in the same manner depicted in 2.3. Therefore, the optimal solution which is provided by our algorithm is: $\sum_{i=1}^{n} \mathcal{L}(i)d(i)$.

2.3 Dynamic Programming Implementation in Java

```java
/*
   * =========================================================================
   *  Coder Information
   *        Name: Michael Liut
   *        SID: 1132938
   *        MacID: liutm
   *  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   *  Submission:
   *        To: Dr. Antoine Deza
   *        Class: CAS 6O03
   *        Dept: Computing and Software,
   *              Faculty of Engineering,
   *              McMaster University
   *  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   *  Description:
   *        Rod Cutting (Stick Cutting) problem. Solved with the use of the
   *        Huffman Tree in O(n^2) time.
   *  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   *  References:
   *        Introduction to Algorithms, 3rd Edition, Cormen et. al., MIT Press
   *        https://en.wikipedia.org/wiki/Huffman_coding
   * =========================================================================
*/

/* Imports */
import java.util.*;
import java.lang.*;

/* Begin RodCutting Class */
public class RodCutting {

    /*
     * ======================================================================
     *  Begin Tree Structure aka Tree Node Class
     * ======================================================================
    */
    class Tree implements Comparable<Tree> {
        /* Attribute Declarations */
        private Tree right, left;
        private long weight;

        /* Default Constructor */
        public Tree() {
            this.weight = 0;
            this.right = null;
            this.left = null;
        }

        /* Constructor */
        public Tree(long weight) {
            this.weight = weight;
            this.right = null;
            this.left = null;
        }

        /* Method - Returns the Right Child Node */
        public Tree getRightChild() {
            return this.right;
        }

        /* Method - Returns the Left Child Node */
        public Tree getLeftChild() {
            return this.left;
        }

        /* Method - Adds a Right Child Node */
        public void addRightChild(Tree rightChild) {
            this.right = rightChild;
        }

        /* Method - Adds a Left Child Node */
        public void addLeftChild(Tree leftChild) {
            this.left = leftChild;
        }

        /* Method - Returns the Weight of the Node */
        public long getWeight() {
            return this.weight;
        }

        /* METHOD OVERRIDE - Comparison of Nodes */
        @Override
```

```java
        public int compareTo(Tree node) {
            if (this.weight < node.getWeight()) {return -1;}
            else {
                if (this.weight > node.getWeight()) {return 1;}
                else{return 0;}
            }
        }

    } // End of Tree Class

    /*
     * ========================================================================
     *   Begin HuffmanTree Class
     * ========================================================================
    */
    class HuffmanTree {
        /* Attribute Declarations */
        private Tree node;
        private long weight;

        /* Default Constructor */
        public HuffmanTree() {
            this.node = null;
            this.weight = 0;
        }

        /* Constructor */
        public HuffmanTree(Tree node) {
            this.node = node;
            this.weight = node.getWeight();
        }

        /* Method - Returns the Weight of the Node */
        public long getWeight() {
            return this.weight;
        }

        /* Method - HuffmanTree created on array parameter (length of rods) */
        public boolean createHuffmanTree(ArrayList<Long> longList) {
            /* Tree Structure Variable Declarations */
            Tree tempOne, tempTwo, parentTree;

            /* Tree Structure Variable Initialization */
            tempOne = null;
            tempTwo = null;
            parentTree = null;

            /* Initialize Variable to hold total node weight */
            long totalWeight = 0;

            /* ArrayList Tree Structure Instantiation */
            ArrayList<Tree> treeList = new ArrayList<Tree>();
            for (Long i : longList) {
                Tree newNode = new Tree(i);
                treeList.add(newNode);
            }

            /* Iterates Through Data Structure */
            while (treeList.size() > 1) {
                tempOne = Collections.min(treeList);
                treeList.remove(tempOne);

                tempTwo = Collections.min(treeList);
                treeList.remove(tempTwo);

                totalWeight = tempOne.getWeight() + tempTwo.getWeight();

                parentTree = new Tree(totalWeight);

                parentTree.addLeftChild(tempOne);
                parentTree.addRightChild(tempTwo);
                treeList.add(parentTree);

                this.weight += parentTree.getWeight();
            }

            if (treeList.size() < 1){
                return false;
            } else {
                this.node = treeList.get(0);
                return true;
            }
        } // End of HuffmanTree Method

    } // End of HuffmanTree Class

```

```java
    /*
     * ========================================================================
     *   Main Method
     * ========================================================================
    */
    public static void main(String[] args) throws IllegalArgumentException{

        try{
            if(args.length < 2){
                throw new IllegalArgumentException("Invalid Number of Arguments!");
            }

            long total = Long.parseLong(args[0]);
            long occurences = Long.parseLong(args[1]);

            if (occurences != args.length-2){
                throw new IllegalArgumentException("Invalid Number of Arguments!");
            } else {
                ArrayList<Long> length = new ArrayList<Long>();
                length.add(Long.parseLong(args[2]));

                for(int i = 2; i < args.length-1; i++){
                    length.add(Long.parseLong(args[i+1])-Long.parseLong(args[i]));
                }

                length.add(total-Long.parseLong(args[args.length-1]));

                /* Instantiate a HuffmanTree Object start */
                RodCutting start = new RodCutting();
                RodCutting.HuffmanTree cutting = start.new HuffmanTree();
                cutting.createHuffmanTree(length);
                System.out.printf(String.valueOf(cutting.getWeight())+"\n");

            }
        } catch(IllegalArgumentException e) {
            System.out.println("Error! Invalid Argument(s)!");
            System.out.println("Trying to break my program is not nice...");
        }
    } // End of Main Method
} // End of RodCutting Class
/*
   * ==========================================================================
   * ==========================================================================
   *                            END OF PROGRAM
   * ==========================================================================
   * ==========================================================================
*/
```

3 References

3.1 Printed Resource(s)

1. Introduction to Algorithms, 3rd Edition, Cormen et. al., MIT Press

3.2 Web Resource(s)

1. https://en.wikipedia.org/wiki/Huffman_coding