Parker Bedlan

Mr. Speice

Independent Study and Mentorship, 2A

16 October 2016

Research Assessment #4

Subject: Ecological Bin Packing - Prioritization and 2D Arrays

MLA Citation:

"Ecological Bin Packing." *UVa Online Judge*. University of Valladolid, 30 Aug. 2002. Web.

Assessment:

My approach to this Research Assessment was more based around the technical aspects of programming than in the research of broader concepts. I went to UVa Online Judge, a site that holds various programming projects to learn from. In this case, I worked on a solution for the Ecological Bin Packing dilemma, and I grew as a programmer from working on it.

The Ecological Bin Packing dilemma is a problem describing that there are three bins, each with a mixture of brown, green, and clear glass to be recycled. The input consists of the amounts of these three in each bin, and the output is the minimum amount of moved bottles it takes to separate each type into their own bin, and which bin holds which type of glass.

When I first encountered this problem, it seemed very self explanatory; I could simply move the brown glass to bin 1, the green glass to bin 2, and the clear class to bin 3 every time, and count the amount I had to subtract from the other bins. However, having a similar lesson that was learned in my last Research Assessment relating to Dynamic Programming, more simple to

program does not always mean more simple to execute. It actually makes much more sense to decide on which bin has the most of a certain bottle and decide that as the bin that uses that type, then move onto the second most obvious choice for which color of bottle to use, then use the third for the one that is left. In the context of programming, this was more difficult than it seems, because often times the two largest amounts of bottles may be in the same bin, and to continue with the method of highest to lowest amounts would leave one bin empty. To make the method effective, I had to let it take the bin with the highest amount of a color, then omit that bin when searching for the next highest amount to work off of. I accomplished this by having an arraylist (a list that can change in size) of which bins to check each time I looked for the largest number of a type of bottle.

A second, more technical dilemma I faced was whether to use a 1D array (a basic list) to display the bins and bottles or a 2D array (a grid).

1D Array:
10 15 20 30 12 8 15 8 31

2D Array:
```
          B   G   C
Bin #1: 10 15 20
Bin #2: 30 12 8
Bin #3: 15  8  31
```

The difference here was once again between ease to program or ease to execute. It would have been very easy for me to have a list of nine numbers: 3 bottle colors times 3 bins equals nine numbers. However, I would have had to hardcode how the numbers would line up to act as a grid, so it was simply more professional to use a 2D array. At first it was difficult to get the hang of how it worked, because I had to worry about an x and y axis instead of just an x axis.

However, it became much more elegant to type. If I want to check all of the green bottles, for example, I could simply use (1, y) like a graph instead of listing out indexes 1, 4, and 7.

Overall, it was refreshing, difficult, and enjoyable to solve this problem in a way that I had not used before. Because of my work on this small project, I was able to open my mind to the idea that the path that is most difficult to start out can often be the most efficient one in the long run. I will make sure to remain open to all approaches to problem-solving, even the ones I am not familiar with.

Ecological Bin Packing

Background

Bin packing, or the placement of objects of certain weights into different bins subject to certain constraints, is a historically interesting problem. Some bin packing problems are NP-complete but are amenable to dynamic programming solutions or to approximately optimal heuristic solutions.
In this problem you will be solving a bin packing problem that deals with recycling glass.

The Problem

Recycling glass requires that the glass be separated by color into one of three categories: brown glass, green glass, and clear glass. In this problem you will be given three recycling bins, each containing a specified number of brown, green and clear bottles. In order to be recycled, the bottles will need to be moved so that each bin contains bottles of only one color.
The problem is to minimize the number of bottles that are moved. You may assume that the only problem is to minimize the number of movements between boxes.
For the purposes of this problem, each bin has infinite capacity and the only constraint is moving the bottles so that each bin contains bottles of a single color. The total number of bottles will never exceed 2^31.

The Input

The input consists of a series of lines with each line containing 9 integers. The first three integers on a line represent the number of brown, green, and clear bottles (respectively) in bin number 1, the second three represent the number of brown, green and clear bottles (respectively) in bin number 2, and the last three integers represent the number of brown, green, and clear bottles (respectively) in bin number 3. For example, the line 10 15 20 30 12 8 15 8 31
indicates that there are 20 clear bottles in bin 1, 12 green bottles in bin 2, and 15 brown bottles in bin 3.
Integers on a line will be separated by one or more spaces. Your program should process all lines in the input file.

The Output

For each line of input there will be one line of output indicating what color bottles go in what bin to minimize the number of bottle movements. You should also print the minimum number of bottle movements.

Comment [1]: Upon further research, I found that this is accurate. There are many variations to this problem.

Comment [2]: This sounds like a 2D array problem

Comment [3]: So the problem is not to move the bottles into their own bins. It is to do so in the absolute smallest amount of movements each time.

Comment [4]: 'int' datatype will suffice

Comment [5]: while loop, if(hasNext())

Comment [6]: Scanner, using nextInt() to store into an array

Comment [7]: not printing final appearance, printing movement count. Will need an int counter state variable.

The output should consist of a string of the three upper case characters 'G', 'B', 'C' (representing the colors green, brown, and clear) representing the color associated with each bin.
The first character of the string represents the color associated with the first bin, the second character of the string represents the color associated with the second bin, and the third character represents the color associated with the third bin.
The integer indicating the minimum number of bottle movements should follow the string.
If more than one order of brown, green, and clear bins yields the minimum number of movements then the alphabetically first string representing a minimal configuration should be printed.

Sample Input

1 2 3 4 5 6 7 8 9
5 10 5 20 10 5 10 20 10

Sample Output

BCG 30
CBG 50

Comment [8]: Something to test my code with

```
import java.io.*;
import java.util.*;

public class EcoBins
{
	public static void main(String args[])
	{
		int[] inputs = new int[9];

		Scanner kb  = new Scanner(System.in);

		while(kb.hasNext())
		{
			for(int i = 0; i < 9; i++)
			inputs[i] = kb.nextInt();

			Scanner sc;

			EcoBins eb = new EcoBins(inputs);

			for(int i = 0; i < 3; i++)
			{
				sc = new Scanner(eb.findBiggest());
					int biggesty = sc.nextInt();
					int biggestx = sc.nextInt();

				eb.removeBinToCheck(biggesty);

				eb.transfer(biggesty, biggestx);

			}

			System.out.println(eb.output());
		}

	}

	int[][] bins = new int[3][3];
	ArrayList<Integer> binsToCheck = new ArrayList<Integer>();
	int counter = 0;

	public EcoBins(int[] inputs)
	{
		binsToCheck.add(new Integer(0));
		binsToCheck.add(new Integer(1));
		binsToCheck.add(new Integer(2));

		int i = 0;

		for(int y = 0; y < 3; y++)
		{
			for(int x = 0; x < 3; x++)
```

```
                {
                        bins[y][x] = inputs[i];
                        i++;
                }
        }
}

public String findBiggest()
{
        int biggest = -1;
        int biggestx = -1;
        int biggesty = -1;

        for(int i = 0; i < binsToCheck.size(); i++)
        {
                for(int x = 0; x < 3; x++)
                {
                        int num = bins[binsToCheck.get(i)][x];

                        if(num > biggest)
                        {
                                biggest = num;
                                biggestx = x;
                                biggesty = binsToCheck.get(i);
                        }
                }
        }

        return "" + biggesty + " " + biggestx;
}

public void removeBinToCheck(int rbin)
{
        binsToCheck.remove(binsToCheck.indexOf(new Integer(rbin)));
}

public void transfer(int biggesty, int biggestx)
{
        for(int y = 0; y < 3; y++)
        {
                if(y == biggesty)
                        continue;

                int temp = bins[y][biggestx];

                bins[biggesty][biggestx] += temp;
                bins[y][biggestx] -= temp;
                counter += temp;
        }
}

//This method was only used for debugging
```

```
	/*
	public void print()
	{
		for(int y = 0; y < 3; y++)
		{
			for(int x = 0; x < 3; x++)
			{
				System.out.print(bins[y][x] + " ");
			}

			System.out.println();
		}
	}
	*/

	public String output()
	{
		String output = "";

		for(int y = 0; y < 3; y++)
		{
			if(bins[y][0] > 0)
				output += "B";
			else if(bins[y][1] > 0)
				output += "G";
			else if(bins[y][2] > 0)
				output += "C";
		}

		output += " " + counter;

		return output;
	}
}
```