CS 124 Programming Assignment 3

Britteny Okorom-Achuonye and Vanessa Chambers

April 2020

1 Dynamic Programming Solution

1.1 Definition

We can define a two dimensional array F. F will be a n by b sized array,indexed at zero, where $F[i][j]$ is True if a subset of the first i numbers in our input array is equal to j and False otherwise. In order to find an answer we should love at the cell $F[n-1][b/2]$. if this cell has a True value that means that the entire list of numbers can be split into a subset equal to half of the sum of the total list, which implies that the other subset is also equal to half of the sum. This case would mean that the list can be partitioned with a residue of zero. If this cell is not true we can go backward looking at each cell $F[n-1][(b/2)-r]$ where r will be our residue. This is because if we find a cell of our array $F[n-1][(b/2)-r]$ that is True that means that the entire list A has a subset equal to r less than half of the total sum, this implies that the other subset is equal to $b/2+r$ since both subsets must sum together to b.

1.2 Recursion:

When we create the two dimensional array F we can initialize all $F[i][0]$ values to True since the empty set can always be a subset tat is equal to 0. We can also set all $F[0][j]$ to False except $F[0][0]$ since an empty array can only have a subset sum of zero.

When it comes to filling other values of the array we know that if $F[i-1][j]$ is True than this implies that $F[i-1][j]$ must also be true since if we have already found a subset of the list that is equal to j adding more numbers won't change this.

We also know that if $F[i-1][j]$ is true than the we know that $F[i][j+a_i$ must also be true. This is because we if we have a subset of value j in the first $i-1$ values of the list A than if we add the next number, a_i, to that subset it will have a value of $j+a_i$.

The rules listed are exhaustive, meaning there is no other way to get a True value for an entry of the array outside of these, so we should set all other entries of the array to False.

1.3 Analysis:

The space complexity of this algorithm will be $O(nb)$ since this is the size of our array F. The time complexity of the algorithm will also be $O(nb)$ since we will in the worst case need to fill up the entire array and go through $b/2$ cells of the array.

2 Karmarkar-Karp Algorithm Runtime

The Karmarkar-Karp Algorithm can be implemented in nlogn steps. This algorithm first has to find the two max numbers and then take the difference of the two and update the two previous max values with the difference and a zero. The time it takes to extract the ax of a list is just $logn$ since we will use a binary method. We can do this twice to find the two highest numbers and then perform a constant time subtraction on the numbers inserting the difference as well as a zero back into the list. We will repeat this at most n times since on every extraction of two numbers we are putting back at least one zero. since we are doing $O(logn)$ steps n times we know that the entire algorithm can be implemented in $O(nlogn)$ steps.

3 Partition Program

See Java file.

4 Random Trials

100 random instances of 100 numbers were generated and solved using the Karmarkar-Karp, Repeated Random, Hill Climbing, and Simulated Annealing algorithms in their standard and pre-partitioned form. The average residual for all 100 instances was calculated along with run-time for each algorithm (Table 1).

According to Table 1, we can see that the Karmarkar-Karp produces lower, and therefore more optimal residual values than the three algorithms in their standard forms. This could be because each of these algorithm randomly chooses a solution to start with, and depending on how the "neighborhood" of the problem is set up, this could be at a position that makes the optimal difficult to reach in a reasonable amount of time, or near a local optima that proves hard to move on from. The Repeated Random algorithm serves to avoid this by producing a random solution upon each call; a drawback to this is that when close to the optimal, the algorithm could move to a spot that is very far from the optimum–there is no guarantee that it will move closer. On the other hand, the hill climbing problem has no way of addressing this issue, as a neighbor of the randomized solution is always selected upon each iteration. Thus, the simulated annealing algorithm deals with this, allowing for locally less optimal decisions to be made in order to avoid falling into a local minimum.

The pre-partitioned implementations of Repeated Random, Hill Climbing, and Simulated Annealing, using the KK algorithm resulted in lower residual values overall. This follows because as shown in Table 1, Karmarkar-Karp algorithm generated lower residual values when compared to the standard implementations of the other algorithms; thus an optimization of the Karmarkar-Karp algorith should produce increasingly optimal residual values. Thus, we can conclude that the Karmarkar-Karp algorithm used with Repeated Random, Hill Climbing, and Simulated Annealing implementations generate lower residual values. The Pre-Partitioned Repeated Random algorithm provided the most optimal result and had similar running times to the other pre-partitioned algorithms.

Although they produce more optimal residues, a drawback to these pre-partitioned algorithms is that each pre-partitioning requires additional time, thus increasing the running-time of the algorithm. This is evidenced by the running-time results in Table 1. Additionally, more space was used as the pre-partitioned lists had to be stored in addition to the original lists.

Algorithm	**Mean Residual**	**Running-Time (nanoseconds)**
Karmarkar-Karp	230626	19085553
Repeated Random	239560365	10814094129
Pre-Partitioned Repeated Random	174	141694697509
Hill Climbing	297944323	1314432604
Pre-Partitioned Hill Climbing	939	135506596329
Simulated Annealing	12647149083	377002095187
Pre-Partitioned Simulated Annealing	870	139975084370

5 Randomized Algorithms

If we were to start our randomized algorithm with the output of the Karmarkar-Karp Algorithm we will be more accurate than if we were to start with a random one. If we look at the standard representation we are guaranteed to have a result that is at least as good as one that would result from using Karmarkar-Karp. In the worst case where the result is not improved it will be faster since we do not have to use as many iterations to get to our solution