The Partition Problem A Dynamic Problem

Cemal Arican

February 2020

1 The Problem

Sometimes also known as the splitting the loot problem. The question is, if there is a way to create a subset $S \subseteq \{1,...,n\}$ for the items with weights $w_j, \forall j \in \{1,...,n\}$ such that $\sum_{j\in S} w_j = \sum_{j\notin S} w_j = \frac{1}{2}\sum_{j=1}^{n} w_j$.

2 The Dynamic Program

The idea of a dynamic problem is to iteratively update your answer/solution as you go along a given array or matrix that you are given as an input. In a dynamic program we need to first identify a sub problem. Once we have identified our subproblem, we follow up with 3 steps. 1) Initialization, 2) Recursion and 3) Output along with the input size and running time.

Subproblem:

Let $\frac{1}{2}\sum_{j}^{n} w_j = B$, and variable v such that $v \in \{0,...,B\}$ and the i-th item such that there exists a set $S = \{1,...,i\}$ and consider the following function:

$$F(i,v) = \begin{cases} 1 & \text{if } \exists S \subseteq \{1,...,i\} \text{ and } v \in \{0,...,B\}, \\ 0 & \text{otherwise.} \end{cases}$$

Algorithm 1: Part One: Initialization

```
for $v \in \{0,...,B\}$ do
    // $O(B)$
    if $v = 0$ or $v = w_1$ then
        // $O(1)$
        $F(1,v) = 1$
    else
        $F(1,v) = 0$
    end
end
```

Algorithm 2: Part 2: Recursion

```
for $i \in \{2, ..., n\}$ do
    // $O(n-1)$
end
for $v \in \{0, ..., B\}$ do
    // $O(B)$
    if $v \geq w_i$ // $O(1)$
     then
        $F(i,v)$=$F(i-1,v)$ or $F(i-1,v-w_i)$
    else
        $F(i,v) = F(i-1,v)$
    end
end
```

Algorithm 3: Part Three: the output

```
Result: return for any set $S \subseteq \{1, ..., n\}$ where $F(n,B) = 1$, depending on your
  specification, you could return any set of item where the function is true as
  long as it equals $B$, // $O(1)$
```

Lets look at an example, we shall create a table with the results of the algorithm as we iterate over all n items. Consider an array as input of $n = 7$ number of jobs with weights $w_i = \{2,2,5,8,3,1,1\}, \forall i \in n$. The weight of all the jobs is 22, and thus we want to find a set, S such that the sum of the number of items in our set S equals 11, set $B = 11$.

Initially the table looks like this:

Algorithm Results												
$(i) \in n, w_{(i)}$	0	1	2	3	4	5	6	7	8	9	10	$11 = B$
(1), {2}	1	0	1	0	0	0	0	0	0	0	0	0

when v = 2:

Algorithm Results												
$(i) \in n, w_{(i)}$	0	1	2	3	4	5	6	7	8	9	10	$11 = B$
(1), {2}	**1**	0	1	0	0	0	0	0	0	0	0	0
(2), {2, 2}	1	0	1	0	**1**	0	0	0	0	0	0	0

quick explanation, for $F((2), v = 4)$ has to equal $F((2-1),4) = F((1),4) = 0$ or $F((1),4-2) = F((1),2) = 1$. Hence, we set $F((2), v = 4) = 1$.

Continuing untill the end:

Algorithm Results												
$(i) \in n, w_{(i)}$	0	1	2	3	4	5	6	7	8	9	10	11 = B
(1), {2}	1	0	1	0	0	0	0	0	0	0	0	0
(2), {2, 2}	1	0	1	0	1	0	0	0	0	0	0	0
(3), {2, 2, 5}	1	0	1	0	1	1	0	1	0	1	0	0
(4), {2, ... , 8}	1	0	1	0	1	1	0	1	1	1	1	0
(5), {2, ... , 3}	1	0	1	1	1	1	0	1	1	1	1	1
(6), {2, ... , 1}	1	1	1	1	1	1	1	1	1	1	1	1
(7), {2, ... ,1, 1}	1	1	1	1	1	1	1	1	1	1	1	1

So the algorithm has to return where $F(n,B) = 1$ We see that by using by using item 1 to 5,6 or 7 we can split the loot depending on the preference. Next, we will investigate the running time of the whole algorithm. The worst case running times are mentioned below the for loops or the if statements in red. Thus putting all these running times, the initialization part, recursion and the output together. We have $O(B \times 1) + O(B \times (n-1) \times 1) + O(1) = TR(nB) \in O(nB)$, respectively.

The input size is dependent on the size of the array and n times the $\log max(w_i)$. The reason why we consider the log time of a value if because the memory it takes to program the number, and as we are considering the worst case running time, we take the max. So let us denote the log max of the array as $log(max_i(w_i)) = W$. Moreover we denoted B as the sum of the all the values. However, it should be noted that in the sum of the array is dependent also the largest value, hence $O(n + nW) \in O(nW)$.

In conclusion as the total running time is dependent of the size of the array and the maximum value within that array, hence it is pseudo-polynomial rather than polynomial.