CSC301 HW8

Alex Zhang

March 2023

Question 1

The updated prgram is called "knapsackOut.cpp" and there are three output files called "smallout.txt", "mediumout.txt", and "largeout.txt" which writes the output in the format.

Question 2

Space In the file "knapsackCap"'s knapscakCap method, I created two constant dimension arrays "bag" and "m". Both of them are actaully $(W+1) \times 2$ matrices. So the overall space complexity will be $O(2*2*(W+1)) = O(4W+4)$. For each calling of this function W is the leading term and 4 is a constant, the overall space complexity will be

$$O(W)$$

Time In "knapsackCap" method, first there are loops for initialization, which takes $O(W)$. Further, there is a nested loop which the outer loop iterates n times, and inner loop iterates W times. The time complexity for this loop is $O(nW)$. Assume that creating variables and comparision take constant time. The overall time complexity will be $O(W+nW)$, which is

$$O(nW)$$

Question 3

Space In the file "knapsackDC.cpp", method knapsac dc first calculate the capacity k, which requires at most $O(W)$ space. For each recursion calls, there are two sub recursion. Therefore in each level, the cost for calculating k will be $O(2W) = O(W)$.

Following that, there are two function calls used for recording values and element number. Each of them requires at most $O(n)$ space. The divide and conquer recursion goes from n until the base case. So the depth of this divide and conquer method is $\log n$ because each time it is divided by half. So the overall cost of space for these function calls is $O(2n \log n) = O(n \log n)$.

The following section is used for storing the current value and capacity, which needs at most $O(W)$. However, this memeory will be collected after the recursion so total memory useage will be $O(W)$.

The last is creating a solution array that stores index in each recursions. This overall only needs $O(n)$ because the worst case is when every item is picked which gives you $O(n)$ complexity.

The overall memory complexity should be $O(W + n\log n + n)$. Since W is the dominant term, the complexity will be

$$O(W)$$

Time The majority cost of time is taken by the calculation of capacity k. In 0 level recursion, it needs $O(nW)$. In l level of recursion it needs $O(\frac{nW}{2^l})$. So the total time complexity for calculating k will be $\sum_{l=0}^{\log n} O(\frac{nW}{2^l}) = O(nW)$ if W is super big.

The following code chunk of getting correspondence values and elements has the same time complexity with calculating k. And stroing the capacity used a for loop takes $O(W)$.

So the overall time complexity will be $O(nW + nW + W)$. This can be generalized into,

$$O(nW)$$

There are also three output files call "smallDCout.txt", "mediumDCout.txt", and "largeDCout.txt".

Reference

Xing, Feifan. "A Hybrid Dynamic Programming Algorithm for Solving the 0-1 Knapsack Problem." (2022).
Justin. Reconstructing the List of Items from a Space Optimized 0/1 Knapsack Implementation, 25 Apr. 2016, https://stackoverflow.com/questions/36834028/reconstructing-the-list-of-items-from-a-space-optimized-0-1-knapsack-implementat.