Xingya Ren
260784116

COMP251 Assignment3 Long Answer

I. Correctness of dynamic programming algorithm.

Prove by Strong Induction

First, we define **States** (sub-problems) as the maximum value for a knapsack with weight limit j, and we are given the first i items.

Then, we have for all i, j, the corresponding state

$a(i,j) = \max\{a(i-1, j-W_i) + V_i, a(i-1, j)\}$

where the first option in max is when the value if item i is in the knapsack and the second option is the value when i is not in the knapsack. And, as seen in class, W_i is the weight of i, V_i is the value of i.

Sorry, I couldn't erase this. (free app.)

Other than the OPT seen in class, we introduce another algorithm — calculating the state a — and then use another algorithm to convert a to the OPT.

```
calculatestate (W, V);
Initialize a(i,0) = a(0,j) = 0 as base case;
for i = 1 → n do
    for j=1 → W do
        a(i,j) = a(i-1, j);
        if j ≥ w[i] and a(i-1, j - w[i] + v[i] > a(i,j) then
            a[i,j] = a[i-1, j - w[i] + v[i];
    end
end
end
```

```
findoutput(n, W, a);
if n=0 or W=0 then
    return
end
if a[n, W] = a[n-1, W] then
    findoutput(n-1, W, a);
end
else
    findoutput(n-1, W-w[n], a);
    print(n);
end
```

Proof by Strong Induction:

Let (i, j) be a pair representing an item that is to be added to the sack, where i is the value of the item and j is its weight.

We say $(i, j) < (i', j')$ if $i < j'$ or $i = i'$ and $j < j'$.

This way we can order the items i.e.

$(0,0) < (0,1) < (0,2) < \dots < (1,0) < (1,1) < \dots$

Base case:

$$a[i, 0] = a[0, j] = 0 \quad \text{for } \forall i, j$$

Induction Hypothesis:

Assume the algorithm is correct for all values of $a[i, j]$ where $(i, j) < (i', j')$, that is all items that come before (i', j'), using the algorithm, are correct.

Induction step:

When computing $a(i', j')$, by the induction hypothesis, we have $a(i'-1, j')$, $a(i'-1, j'-w_{i'})$ already computed correctly and put into the sack.

Then in the case where i is in the knapsack, the algorithm considers $(a(i'-1, j'-w_{i'}) + v_{i'})$

otherwise it considers $(a(i'-1, j'))$.

Since the above two values are correct according to the induction hypothesis, the algorithm is correct for $a(i', j')$. □

A similar proof can be done for OPT.

Let (i, W) be a pair representing an item that is to be added to the sack, where i is the value of the item and j is its weight.

We say $(i, W) < (i', W')$ if $i < i'$ or $i = i'$ and $W < W'$. (i', W') is a pair representing another item to be added to the knapsack.

This way we can order the items (value, weight) pairs, i.e. $(0,0) < (0,1) < (0,2) < \dots < (1,0) < (1,1) < \dots$

Base case:

When $i' = 0$, $OPT(i', W') = 0$ by the first case in the given algorithm.

Induction Hypothesis:

Assume the algorithm is correct for all values of $OPT(i, W)$ where $(i, W) < (i', W')$, that is all items that come before (i', W') are correctly computed and added.

Induction Step:

When computing $OPT(i', w')$, if $w' > w$, that is case 2 of the algorithm $OPT(i', w') = OPT(i-1, w)$.

$OPT(i-1, w)$ is already computed correctly according to the algorithm because $i-1 < i$.

The algorithm thus holds true for $OPT(i', w')$.

Now consider the third case in the algorithm, we have $OPT(i'-1, w')$, $OPT(i'-1, w'-w_{i'})$ already computed correctly and put into the sack.

Since by the way we order the pairs, these pairs come before (i', w'), because $i'-1 < i'$ and $w'-w_{i'} < w'$, as weight can not be negative.

Then in the case where i is in the knapsack, the algorithm considers $(OPT(i'-1, w'-w_{i'}) + v_{i'})$ otherwise it considers $(OPT(i'-1, w'))$.

Since the above two values are correct according to the induction hypothesis, and max {...} function works as expected, the algorithm holds true when we add (i', j') to knapsack.

Therefore, the given algorithm of the knapsack problem is correct by strong induction. □

II. Bounded Knapsack Problem

Let C_i be the number of copies of i and C_i is an integer greater than 0.

And X_i, as used in the question, is the number of item i, in this case $0 \leq X_i \leq C_i$, in the optimal solution.

Therefore, this knapsack problem is different from the original (previous) one as we may consider adding multiple copies of i into the sack while keeping the weight under limit.

Thus, the algorithm can be modified in the following way

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max \left\{ \sum_{x_i=0}^{C_i} (x_i \cdot v_i) + OPT(i-1, w - x_i \cdot w_i) \right\} & \text{otherwise} \end{cases}$$

III. Unbounded Knapsack Problem

Similar to the previous problem, but with x_i being able to approach positive infinity. The proof is omitted since the question only asks to provide a dynamic programming algorithm.

Use a memoization array `dp[values.length][w+1]` where w is the capacity of the knapsack, to store values of our subproblems. Populating the array, we want the maximum value for every sub-array and for every possible w.

There are 2 possible cases:

1) exclude the item. `dp[i-1][w]`
2) include if its weight $\leq w$. $V_i + dp[i][W - W_i]$ where V_i is the value of the item.

Taking the max of the above two:

$$dp[i][w] = \max\{dp[i-1][w], w[i] + dp[i][W - w[i]]\}$$

Thank you for your time! :)