CSCI 2300
Introduction to Algorithms: HW7

Sean Hyde
RIN: 662096071

Due: October 31th, 2024

1 (6.17)

1.1 Subproblem

$L[v]$ represents the maximum value that can be achieved given the denominations $X_1, X_2, \ldots, X_n$ that doesn't exceed $\bar{V}$, where $\bar{V}$ is the given value we wish to make change for.

1.2 Recurrence

$$L[v] = \max_{i:X_i<v} \{L[v - X_i] + X_i\}$$

This recurrence is correct as it calculates the max value that is possible for each capacity from $1 \ldots \bar{V}$, checking every potential combination of dominations that don't exceed the current v. The solutions are stored at each v to calculate for larger values of v.

1.3 Algorithm

Algorithm 1 Compute $L[v]$

```
1: $L[0] = 0$
2: for $v = 1$ to $\bar{V}$ do
3:     $L[v] = \max_{i:X_i \leq v} \{L[v - X_i] + X_i\}$
4: end for
5: return $L[\bar{V}] == \bar{V}$
```

The algorithm will return a boolean value, true if the value of $L[\bar{V}] == \bar{V}$. This is the final subproblem for the desired value we want to make change for, if they are equal, then we know that we are able to make change for this value with the given denominations $X_1, \ldots, X_n$ It will return false if this case is not satisfied.

1.4 Runtime

This algorithm satisfies the constraint of the $O(nv)$ runtime as for each value v, we are iterating through every potential combination of dominations that don't exceed the current v. Resulting in an $O(n\bar{V})$ runtime.

2 (6.19)

2.1 Subproblem

$L[v, p]$ represents the maximum value that can be achieved given the denominations $X_1, X_2, \ldots, X_n$ that doesn't exceed $\bar{V}$, where $\bar{V}$ is the given value we wish to make change for and p represents the amount of coins that we are allowed to use, $\bar{K}$, in our solution.

2.2 Recurrence

$$L[v, p] = \max_{i: X_i \leq v} \{L[v - X_i, p - 1], L[v, p - 1]\}$$

This recurrence is correct because at every level of recurrence we are choosing between including X_i in the solution or not for every denomination, $X_1, \ldots, X_n$ that doesn't exceed $\bar{V}$ based on which choice maximizes the achievable value.

2.3 Algorithm

Algorithm 2 Compute $L[v, p]$

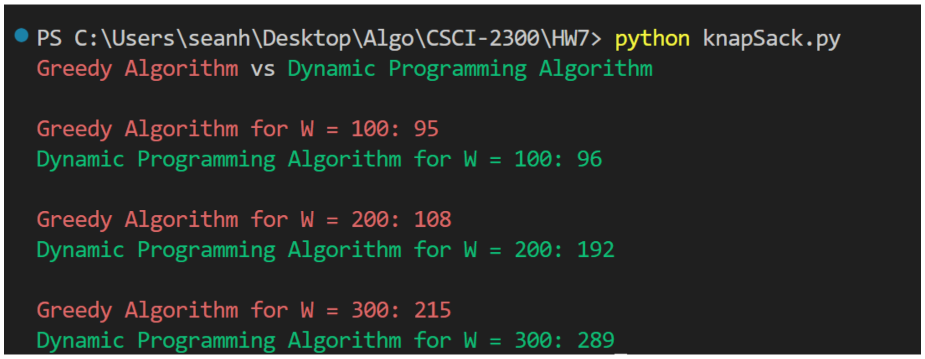
```
1: $L[0, p] = TRUE, L[v, 0] = FALSE$
2: for $p = 1$ to $K$ do
3:     for $v = 1$ to $\bar{V}$ do
4:         $L[v, p] = \max_{i: X_i \leq v} \{L[v - X_i, p - 1], L[v, p - 1]\}$
5:     end for
6: end for
7: return $L[\bar{V}, K]$
```

For the base case where we are assigning initial values, we assign $L[0, p$ to true since for any value 0 you can make change for any number of coins since you have no value to make change for. For $L[v, 0]$, this is false since if you are not given any coins to use, you can not make change for any value, other than $L[0, 0]$. We are returning the value for $L[\bar{V}, K]$, as this is the corresponding element we want since it is the value we want to achieve with the limit of k coins.

2.4 Runtime

This algorithm runs in $O(K\bar{V}n)$ time. This is because in the outer loop, we go from $p = 1$ to $\bar{K}$, giving us $\bar{K}$ iterations. In the second loop we go from $v = 1$ to $\bar{V}$, giving us $\bar{V}$ iterations. Within each iteration of the inner loop we go through each potential denomination $X_1 \dots X_n$ where $X_i \leq v$ giving us n runtime. For a final runtime of $O(K\bar{V}n)$.]

3 (3)



```
● PS C:\Users\seanh\Desktop\Algo\CSCI-2300\HW7> python knapSack.py
  Greedy Algorithm vs Dynamic Programming Algorithm

  Greedy Algorithm for W = 100: 95
  Dynamic Programming Algorithm for W = 100: 96

  Greedy Algorithm for W = 200: 108
  Dynamic Programming Algorithm for W = 200: 192

  Greedy Algorithm for W = 300: 215
  Dynamic Programming Algorithm for W = 300: 289
```

Figure 1: Results for Greedy Algorithm VS. Dynamic Programming Algorithm for given dataset and W = 100, 200, 300