CS 3510 — Algorithms, Spring 2022

Lecture 10: Even More DP

February 17, 2022

Lecturer: Frederic Faulkner *Scribe: Aditya Diwakar*

1 Challenge from Tuesday

Let's start with an $n \times n$ matrix of integers and the goal is to get from the top left to the bottom right and we want the path with the smallest sum (we can only move right and down). Design a DP algorithm to solve this problem.

We can start out by letting $T[i, j]$ represent the minimum cost it takes to get to row i and column j. This is a good starting point for DP.

For the recurrence, remember that with dynamic programming, we don't want to look at the beginning of the solution, only a few intermediate steps from a subproblem.

For any square in the matrix, you are either coming from the left or coming in from the right. There is no other way to get into a square. In terms of smaller subproblems, how long does it take to get here? This is, by definition, $T[i, j]$ and we can look at the table to fill it in.

If we come from the top, it costs us $T[i-1, j]$ to get to the cell above and $T[i, j-1]$ to get to the cell to the left. It also costs us $M[i, j]$ to traverse the current square. Hence, the recurrence is:

$$T[i, j] = \min\left(T[i-1, j], T[i, j-1]\right) + M[i, j]$$

The last step of defining a recurrence relation is what are our base cases? $T[1, 1] = M[1, 1]$ is an obvious one. There are more base cases, however. We want to initialize $T[1, j]\ \forall j$ to be the sum of the row up until the jth column. The same is

true for $T[i, 1]$ $\forall k$. In more exact terms:

$$T[1, j] = M[1, j] + T[1, j-1] \qquad T[i, 1] = M[i, 1] + T[1, i-1]$$

2 Knapsack Problems

We are robbing a house, but can only grab 25 pounds worth of stuff. We want to maximize the value of stuff we steal while staying below the 25 pound limit. We have the following items (with weights being in pounsd and values being dollars):

	painting	bike	change jar	marbles
weight	15	20	10	5
value	\$130	\$200	\$90	\$10

An option is to steal the bike and marbles which is 20 pounds and 5 pounds maximizing our weight to 25 pounds, and has a value that adds to \$210.

Another option is to take the painting and change jar which has a combined weight of 25 pounds and adds up to a value of \$220. This is evidence that a greedy approach does not work.

2.1 Generalized Knapsack

A knapsack problem is a problem in which we have n items where each item can be defined by $(\text{weight}, \text{value})$ and we want to find the subset of items that has the highest total value subject to total weight being less than B, where B is our budget.

This problem has two variations: with and without replacement. We start by exploring the without replacement case.

Let's solve this problem using dynamic programming! First, let us try setting $T[i]$ to be the max value for items $1, \ldots, i$ that has a total weight $\leq B$.

How do recursively define $T[i]$? How can we discuss the inclusion of item i in our knapsack? If item i is not going to be used, we can simply look at $T[i-1]$.

If item i is used, what should $T[i]$ be? We can't use any previous i values because it would not account for the budget being reduced. For example, if we let $T[i] = T[i-1] + v_i$, then $T[i-1]$ could also go over budget.

This means that this entry definition does not work. Instead, let us try $T[i, b]$ which represents the maximum value from items $1, \ldots, i$ with remaining capacity b.

For a recurrence relation, we have two cases. Either i is included or i is not used. In the case it is not used, then the maximum value from items $1, \ldots, i$ is simply the maximum of $1, \ldots, i-1$ with the same capacity. Hence, $T[i, b] = T[i-1, b]$.

If we do use i in the solution, then the value comes from the maximum from the previous $i-1$ items but only with capacity b minus the weight of the recurrence option. Let w_i, v_i be the weight and value of the ith item. Hence, $T[i, b] = T[i-1, b-w_i] + w_i$. There is a special case where if $w_i > b$, then $T[i, b] = T[i-1, b]$ as we cannot afford this item.

We also have two base cases: $\forall i, b$ we set $T[i, 0] = T[0, b] = 0$. All in all, the entire recurrence relation becomes:

$$T[i, j] = \begin{cases} 0 & \text{if } j = 0 \vee i = 0 \\ T[i-1, b] & \text{if } w_i > b \\ \max\left(T[i-1, b-w_i] + v_i, T[i-1, b]\right) & \text{otherwise} \end{cases}$$

In order to implement this algorithm, we can write the following pseudocode:

```
1  Function Knapsack(W = w_1, ..., w_n, V = v_1, ..., v_n, B):
       /* let T be an empty (n+1) × (B+1) 2D table                    */
2      T ← [][]
       /* base cases for first row and column                         */
3      for b ∈ 0 → B do
4          T[0, b] ← 0
5      for i ∈ 0 → n do
6          T[i, 0] ← 0
7      for i ∈ 1 → n do
8          for b ∈ 1 → B do
9              if w_i > b then
                   /* item is too heavy                               */
10                 T[i, b] ← T[i − 1, b]
11             else
                   /* take max of either including item or not        */
12                 T[i, b] ← max(T[i − 1, b − w_i] + v_i, T[i − 1, b])
13     return T[n, B]
```

The runtime of this algorithm is $O(nB)$ as we have a nested for loop that first iterates $O(n)$ times where each loop has $O(b)$ iterations giving $O(nB)$.

Challenges

1. How do we modify the above to consider the case where items are replaced?

   Solution. A very small change can be made to perform replacements. The new recurrence relation becomes the following:

$$T[i,j] = \begin{cases} 0 & \text{if } j = 0 \vee i = 0 \\ T[i-1,b] & \text{if } w_i > b \\ \max\left(T[i, b-w_i] + v_i, T[i-1,b]\right) & \text{otherwise} \end{cases}$$

   In the above, we only changes the first part of the $\max$ such that if we pick an item, then we reduce the budget and add the value, but allow us to consider the same item again by not changing the index i (it was $i-1$ before).

2. What if we also limited the amount of total items? Now, we have both a weight restriction as well a count restriction.

   Solution. We can add a third dimension to the Knapsack table which consider the number of items taken so far. The recurrence relation becomes:

$$T[i,j,k] = \begin{cases} 0 & \text{if } i, j, \text{or } k = 0 \\ T[i-1,b,k] & \text{if } w_i > b \\ \max\left(T[i,b-w_i,k-1] + v_i, T[i-1,b,k]\right) & \text{otherwise} \end{cases}$$

   In the above, we add the base case for when $k = 0$ since we cannot take any items if we allow 0 total items. Otherwise, in any situation where we reduce the weight, we reduce the item count allowed by 1. We can think of an item limit as a weight limit where each item has a weight of 1.