COMPSCI 330: Design and Analysis of Algorithms Spring 2019

Lecture 6: Dynamic Programming III

Lecturer: Rong Ge *Scribe: Shweta Patwa*

1 Review

1. States: Have an English description of the states. We have seen an example using Knapsack. There, we defined states $a[i, j]$ to be the optimal value (or boolean value) for a knapsack of capacity j using the first i items.

2. Transition function and base cases: Often not independent. Can define out of bound cases in either the transition function or the base case(s).

   In the case of Knapsack, we need to check if $j - w_i \geq 0$ for the second case in our transition function you saw in class earlier. The bare minimum you need for the base case is $a[0, j] = 0$ for all j. Alternatively, you can say

   $$a[1, j] = \begin{cases} 0 & \text{if } w_1 > j \\ v_1 & \text{otherwise} \end{cases}$$

3. Algorithm: In English or pseudocode (that does not need to be very precise). Demonstrate how the algorithm should work.

4. Proof of correctness: Prove by induction. Need base case(s), IH and induction step. In the induction step, it is important to go over the order in which you are computing your DP table values.

2 Viterbi's Algorithm

This is a simplified version of voice recognition. Suppose each chunk/segment of sound make up a phoneme, and you are given k phonemes.

Input: For each sound segment, you are given a list of scores for the phonemes. For every pair of phonemes, you are given a score of high likely one is to follow the other. Score is higher if two phonemes are more likely to be connected, and it is low if the two phonemes are less likely to be connected.
Output: Find the sequence of phonemes with the highest score.

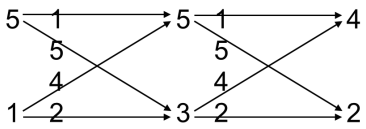
Consider the following example: $n = 3, k = 2$.

Phoneme/sound	1	2	3
1	5	5	4
2	1	3	2

Phoneme/Phoneme	1	2
1	1	5
2	4	2

Optimal solution would be 121 with a score of $(5+3+4)+(5+4)$, where first part follows from the first table and the second from the second table. Score for 2 following 1 is 5. 111 is less favorable because phoneme 1 following phoneme 1 is least likely out of other possibilities.

We can define paths between values in the first table such that the edges have weights given by the second table. Thus, the goal becomes to find the path in this tree that maximizes the sum of vertex and edges weights along that path.



Nodes of graph would be of the form (i, j), where i is a sound segment and j is a phoneme, and edges go from (i, j_1) to $(i+1, j_2)$ where score for the edge comes from the second table. Let us denote this edge weight by $b[j_1, j_2]$.

How do we design a dynamic programming algorithm to solve this problem?

How do I simulate it as a sequence of decisions? For each sound segment, we want to assign a phoneme.

We relate two smaller subproblems as

- Need to know the phoneme of the previous sound in order to evaluate the score on the edges.
- Naive subproblem: max possible score for the first i sounds. But this does not give us enough information. What is the last phoneme? Need to store this information in the state somehow!
- Refined subproblem: $m[i, j] :=$ max possible score for the first i sounds if the phoneme for sound #i is j.
- $m[i, j] = \max_{j'=1,2,\ldots,k} (m[i-1, j'] + b[j', j]) + a[i, j]$, where $a[i, j]$ is the score of assigning phoneme j to sound #i.
- Base case: $m[1, j] = a[1, j]$, if you have one sound and know it is phoneme number j.

  $m[0, j] = 0$ is incorrect because it means that we get an extra transition from sound #0 to #1 but #0 does not exist.
- Algorithm:

  $m[1, j] = a[1, j]$
  for $i = 2$ to n:
  for $j = 1$ to k:
  Evaluate transition function at $m[i, j]$
- Runtime = # states $\times$ time of transition. This gives us $O(nk^2)$.

3 Maximum Independent Set on Trees

Independent set is a set of nodes in the tree that are not connected by any edges.

Goal: find an independent set of maximum size.

- Sequence of decisions: for each node, whether we want to include it in the independent set or not.

- What is the last decision we are making? What is the order of decisions we want to make? To solve this, we fix an arbitrary node to be the root of the tree. The last decision we make will be for the root of this tree.
- Decisions for the root: One case is when the root is not in the independent set. This is an easy case because then we can safely remove the root from the tree, and consider nodes at the next level, which also happen to be the new roots in the tree upon deleting the root. If we can solve the maximum independent set problem in each subtree, then we can simply combine these solutions to get the final answer in the original tree.

  $$\text{Max independent set in this case} = \sum_{i:\text{child of root}} \text{max independent set of the subtree rooted at } i.$$

  However, if the root is in the independent set, then the set we are building cannot contain any of the children of the root. This puts an additional constraint compared to the previous case.

  $$\text{Max independent set in this case} = \Big(\sum_{i:\text{child of root}} \text{max independent set of the subtree rooted at } i \text{ but } i \text{ cannot be in the independent set} \Big) + 1.$$

- For each node i, we have two states.
  $F(i) :=$ size of the maximum independent set for the subtree rooted at i.
  $G(i) :=$ size of the maximum independent set for the subtree rooted at i, but i cannot be in the independent set.
- Transition function:

  $$F(i) = \max \begin{cases} \sum_{j:\text{child of } i} F(j) & (i \text{ is not in the indep set}) \\ 1 + \sum_{j:\text{child of } i} G(j) & (i \text{ is in the indep set}) \end{cases}$$

  $$G(i) = \sum_{j:\text{child of } i} F(j)$$

- Base case: If i is a leaf of the tree, then $F(i) = 1$, and $G(i) = 0$.
- What is the ordering of computation we do? Simple choice is by decreasing depths (root has depth 0). Hint: Breadth-first search. Can instead do this recursively and memoizing values.
- Algorithm:

  $F(i)$
  If i is a leaf, return 1
  If $F(i)$ has been computed, return stored number.
  Evaluate transition function for $F(i)$ and store the result.