Dynamic Programming Examples

Table of Contents:

Rod Cutting:

- Given a rod of length n and a table that shows the prices of a rod from length 1 to length n, determine the max profit we can make by cutting up the rod and selling the pieces.

- E.g. Say our rod is of length 5 and we have the table

length i	1	2	3	4	5
Price P_i	1	5	8	9	10

We see that to max profit, we should cut the rod into 2 rods of length 2 and 3 respectively. This will give us a profit of $13.

Bellman Eqn:

- The optimal substructure for the problem is the max profit we can make when the rod is of length 1 to length n-1.

- Hence, the bellman eqn for this problem is:

$$OPT[i] = \begin{cases} P[i], & \text{if } i=1 \\ \max(P[i], \overset{\max(}{{}_{j}} OPT[j] + OPT[i-j])), & \text{if } i>1 \\ \quad j \in 1, \ldots, i-1 & \end{cases}$$

If the length of the rod is 1, we can't cut it any further, so its max profit is just its price.

If the length of the rod is greater than 1, then we see if cutting it or not cutting it yields a higher profit.

Hence, we first compute the max profit we can make from any cut. This is what $\max_{j,\ j \in 1,\dots,i-1} (OPT[j] + OPT[i-j])$ does.

It loops from 1 to i-1 and gets the max profit that can be made with any cut.

Then, we compare that with $P[i]$, the price of selling the rod as a whole. Again, we take the higher of the 2.

The time complexity is $O(n^2)$.

Longest Simple Path in a Directed Acyclic Graph

- Suppose we are given a directed acyclic graph $G=(V,E)$ with real-valued edge values and 2 distinguish vertices s and t. We want to use DP to find a longest weighted simple path from s to t.

Bellman Eqn:

- The substructure for this problem is to see the longest weighted simple path from S to u where u is an intermediary vertex.

- If $s=t$, the distance $=0$.

- If $s \neq t$, the distance $= \max\left(w_j + OPT[j]\right)$

where j is any vertex that directly precedes t.

- Hence, the bellman eqn is:

$$OPT[i] = \begin{cases} 0, & \text{if } i=s \\ \max(w_j + OPT[j] & , \text{if } i \neq s \\ \text{where } j \in \{\text{nodes that lead to } i\} & \end{cases}$$

The time complexity is $O(|V|)$.

Fibonacci Sequence:

- We want to compute the n^{th} number in the fibonacci seq.

Recall: The first 2 numbers of the sequence are given. They are 0 and 1, respectively.

Recall: $Fib(n) = Fib(n-1) + Fib(n-2)$

Bellman Eqn:

- The substructure is finding the fibonacci values for $i = 2, \ldots, n-1$.

- Hence, the bellman eqn is:

$$OPT[i] = \begin{cases} i & \text{, if } i=0 \text{ or } 1 \\ OPT[i-1] + OPT[i-2] & \text{, if } i>1 \end{cases}$$

- The time complexity is $O(n)$.

Longest Palindrome Subsequence:

- Given a string, return the longest palindrome that is a subsequence of the string.

Note: A palindrome is a string that's read the same forward and backward. Examples include racecar, civic, mom.

Note: All strings of length 1 are palindromes.

Note: A subsequence of a string is a subset of that string where the letters don't have to be consecutive but must follow the same order.
E.g. If the string is apple, then a, app, ale, ae are all subsequences but el isn't because the order changed.

Bellman Eqn:

- The substructure is knowing whether or not the substring from the second char to the second-last char is a palindrome.

E.g. Suppose we have the string abcba. There are 2 conditions we need to check to see if the string is a palindrome:

1. S[0] == S[n-1] where n is the length of the string
2. S[1,. n-2] is a palindrome.

The first condition checks if the first and last letters/chars are the same.

The second condition checks if the inner string is a palindrome.

If both conditions are satisfied, then the string is a palindrome.

We only need to check for this condition if the length of the string is greater than 2.

- E.g. Suppose the string is character. The longest palindrome subsequence is carac.

End Index

Start Index →

	0	1	2	3	4	5	6	7	8
0	1	1	1	1	1	5	5	5	5
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	3	3	3	3	3
3	0	0	0	1	1	1	1	1	2
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	1	1	1	1
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	1

The idea behind this is we have a start index and an end index and we look at the string btwn those indices. E.g. (3,5) corresponds to "rac" in character above.

Let s = start index, e = end index

If s=e, then we have a palindrome of length 1.

If s+1=e, then we have a string of length 2. This string is a palindrome only if the first and last chars are the same.

Now, if the string is of length 3 or greater, we first check to see if the first and last char are the same. If they are, then we do 2 + max palindrome substring length.

The red numbers in the table above indicate the max length of the palindrome that can be made by a string starting at index s and ending at index e.

- Hence, the bellman's equation is:

$$OPT[s.,e] = \begin{cases} 1, & \text{if } s=e \\ 1, & \text{if } s+1=e \text{ and } str[s]=str[e] \\ 0, & \text{if } s+1=e \text{ and } str[s] \neq str[e] \\ \max(0,\ 2+{}_{\wedge}^{\max}OPT[s+1,e-1]), & \\ \quad \text{if } e-s \geq 2 & \end{cases}$$

Breaking a String:

- Given a string of length n and a list of indices on where to split the string, we want to find the sequence of cuts that minimizes the operation cost. It costs n time to split a string of length n into 2.

- E.g. Say we're given a string of length 20 and we want to make the cuts at after index 2, 8 and 10.

Cut #1: 2 → 8 → 10

If we cut on index 2 first, the cost is 20.
If we cut on index 8 second, the cost is 18.
If we cut on index 10 third, the cost is 12.
The total cost is 50.

Cut #2: 10 → 8 → 2

$$\left.\begin{array}{l}\text{Cut on index } 10 \rightarrow \text{Cost} = 20 \\ \text{Cut on index } 8 \rightarrow \text{Cost} = 10 \\ \text{Cut on index } 2 \rightarrow \text{Cost} = 8\end{array}\right\} \text{Total Cost} = 38$$

Bellman's Equation:

$$OPT[i,j] = \begin{cases} 0, & \text{if } i=j \\ \min\limits_{i<k<j} \{OPT[i,k] + OPT[k,j] + (c_j - c_i)\} & \end{cases}$$

The idea behind this is this: We add the first and last index to L and that will be our base case.

E.g. If $L = [2, 8, 10]$, then $L' = [0, 2, 8, 10, 19]$

Next, we find the min cost of cutting between every pair of indices.

Going back to the example:

	0	2	8	10	19
0	0	1	2	3	4
2	0	0	1	2	3
8	0	0	0	1	2
10	0	0	0	0	1
19	0	0	0	0	0

Num of possibilities

If we look at the cut at (0,2), there's only 1 choice as there's no intermediary cut index. With (0, 8), we can either do (0, 2) followed by (2, 8) or we can just do (0, 8).