Tutorial: Dynamic Programming

Difference between Dynamic Programming and Greedy Algorithm:

Greedy Algorithm:

1. Know only partial information in each decision
2. May not guarantee the optimal solution (You need to prove it is optimal!)
3. Usually fast

Dynamic Programming:

1. Know the whole decision space
2. Can guarantee the optimal solution **once you formulate the problem** correctly
3. Usually slower than Greedy Algorithm

Characteristics of Dynamic Programming:

1. Optimal Substructure Property (**Combine the optimal solution of sub-problems** to **construct the optimal solution of the problem**)
2. Usually the sub-problems may overlap with each other (No overlap→ Divide and Conquer)

Example:

Given two strings X and Y, two operations can be used to convert from string X to string Y.

Op1: Delete a letter Example: “applev” to “apple” (Delete “v”)

Op2: Replace a letter Example: “lead” to “read” (Replace “l” with “r”)

The edit-distance between two strings is defined by **the minimum number of operations** to convert X to Y. This problem is shown as follows:

Input: Two strings X[1..m] and Y[1..n]
Output: The edit-distance between X and Y

What is the optimal substructure of this problem?

Ans: The edit-distance between two substrings X[1..i] and Y[1..j]

Now, we formulate this problem as follows:

Let C(i,j) be the edit-distance between X[1..i] and Y[1..j],

<u>Case 1: X[i]=Y[j]</u>

$$C(i,j) = C(i-1,j-1)$$

<u>Case 2: X[i]≠Y[j]</u>

Case 2a: Replace X[i] with Y[j]

$$C(i,j) = C(i-1,j-1) + 1$$

Case 2b: Delete X[i]

$$C(i,j) = C(i-1,j) + 1$$

Case 2c: Delete Y[j]

$$C(i,j) = C(i,j-1) + 1$$

Recurrence Case:

$$C(i,j) = \begin{cases} C(i-1, j-1) & if\ X[i] = Y[j] \\ min\{C\ \ i, j-1), C(i-1, j), C(i-1, j-1)\} + 1 & if\ X[i] \neq Y[j] \end{cases}$$

Base Case:

$$C(i, 0) = i$$

$$C(0, j) = j$$