Share this page (https://twitter.com/intent/tweet?url=https%3A%2F%2Flearnxinyminutes.com%2Fdocs%2Fdynamic-programming%2F&text=Learn+X+in+Y+minutes%2C+where+X%3DDynamic+Programming)

Learn X in Y minutes (/)

Where X=Dynamic Programming

Dynamic Programming

Introduction

Dynamic Programming is a powerful technique used for solving a particular class of problems as we will see. The idea is very simple, If you have solved a problem with the given input, then save the result for future reference, so as to avoid solving the same problem again.

Always remember! “Those who can’t remember the past are condemned to repeat it”

Ways of solving such Problems

1. *Top-Down* : Start solving the given problem by breaking it down. If you see that the problem has been solved already, then just return the saved answer. If it has not been solved, solve it and save the answer. This is usually easy to think of and very intuitive. This is referred to as Memoization.

2. *Bottom-Up* : Analyze the problem and see the order in which the sub-problems are solved and start solving from the trivial subproblem, up towards the given problem. In this process, it is guaranteed that the subproblems are solved before solving the problem. This is referred to as Dynamic Programming.

Example of Dynamic Programming

The Longest Increasing Subsequence problem is to find the longest increasing subsequence of a given sequence. Given a sequence `S= {a1 , a2 , a3, a4, ............., an-1, an }` we have to find a longest subset such that for all `j` and `i`, `j<i` in the subset `aj<ai`. First of all we have to find the value of the longest subsequences(LSi) at every index i with last element of sequence being ai. Then largest LSi would be the longest subsequence in the given sequence. To begin LSi is assigned to be one since ai is element of the sequence(Last element). Then for all `j` such that `j<i` and `aj<ai`, we find Largest LSj and add it to LSi. Then algorithm take *O(n2)* time.

Pseudo-code for finding the length of the longest increasing subsequence: This algorithms complexity could be reduced by using better data structure rather than array. Storing predecessor array and variable like largest*sequences*so_far and its index would save a lot time.

Similar concept could be applied in finding longest path in Directed acyclic graph.

```
for i=0 to n-1
    LS[i]=1
    for j=0 to i-1
        if (a[i] >  a[j] and LS[i]<LS[j])
            LS[i] = LS[j]+1
for i=0 to n-1
    if (largest < LS[i])
```

Some Famous DP Problems

- Floyd Warshall Algorithm - Tutorial and C Program source code:http://www.thelearningpoint.net/computer-science/algorithms-all-to-all-shortest-paths-in-graphs—floyd-warshall-algorithm-with-c-program-source-code
- Integer Knapsack Problem - Tutorial and C Program source code: http://www.thelearningpoint.net/computer-science/algorithms-dynamic-programming—the-integer-knapsack-problem
- Longest Common Subsequence - Tutorial and C Program source code : http://www.thelearningpoint.net/computer-science/algorithms-dynamic-

programming—longest-common-subsequence

Online Resources

- codechef (https://www.codechef.com/wiki/tutorial-dynamic-programming)

Got a suggestion? A correction, perhaps? Open an Issue (https://github.com/adambard/learnxinyminutes-docs/issues/new) on the Github Repo, or make a pull request yourself!

Originally contributed by Akashdeep Goel, and updated by 2 contributor(s) (https://github.com/adambard/learnxinyminutes-docs/blame/master/dynamic-programming.html.markdown).