Saffron Livaccari 1-12-18

Dynamic Programming

Dynamic programming is a subject that bridges math and computer science. Dynamic programming is most frequently used for optimizing, but is adaptably to algorithms for computer science, and a common method to solve complex problems in pure mathematics. The adaptability and speed of this theory makes this arguably the most important theory to know in computer programming or applied mathematics.

Dynamic Programming does not have a specific linear history, like Pythagorean Theorem or Riemann Integrals. This method has been around for an indefinite time, and cannot be traced to just one person. However, the name of the method was determined in the 1950's by a highly distinguished professor of Mathematics at Stanford University, Richard Bellman. He chose the name because he wanted to show that the method was very dynamic and useful for many different topics (Dreyfus 48).

To review what Dynamic Programming is, it is simply breaking down a seemingly complex problem and solving each sub-problem, building up to the end computation. A good starting example is that of a game. There are 30 matchsticks and two players. Each player can only pick up 1, 2, or 3 matchsticks during their turn. The last player to pick up the last remaining matchstick loses the game. How can I be guaranteed to win the game?

To solve this problem, start with the ending subset. Two facts must be upheld at the end of the game to ensure I win; first, that there is only one matchstick on the table, and second, that it is my opponent's turn. Knowing how the game must end, think of the scenario right before that ending; there are two matchsticks on the table and it is my turn. I know in order to win, I must take one matchstick. Going back one more step, there are three matchsticks on the table, then I know I must take two matchsticks. If there are four matchsticks on the table, then I must take three matchsticks to ensure the opponent loses. If there are 5 matches left. The opponent takes one, then I know I have to take three to win. If my opponent takes two, then I know I have to take two. If my opponent takes three sticks, then I take one. So far, we know that the two scenarios that have to succeed to ensure I win is that there is one matchstick on the table and it is my opponent's turn, and when there are five matchsticks left, it is my opponent's turn. Now, think of the next sub-problem. There are 6 matches. If my opponent takes one, there are 5 matches left. If I take three, then my opponent takes one, then I lose. If I take two matches, then my opponent takes two and I lose. No matter how many my opponent takes or how many I take, I will lose. Think of the next scenario, if there are seven matchsticks on the table and it is my opponent's turn. Since my opponent can take either one, two, or three matchsticks, there is only a slight probability that my opponent will take one matchstick, so that I may take one to force my opponent's turn to be with five matchsticks left, which is something I know I am guaranteed to win. Since there is only a 33% probability that my opponent will take only one match, this is not certainty. Therefore, having seven matchsticks left on the table with my opponent's turn is not an option. Think of eight matchsticks. The same idea occurs, there is a 66% chance that my opponent will take 1 or 2 matches leading us to set up the 5 matches scenario. Let's skip to 9 matches left. My opponent may take 1, 2, or 3 matches, and with any option, I can take 1, 2, or 3 matches to get to the five matches scenario, guaranteeing I win. I must always balance it so during one turn, four matches are taken by both me and my opponent. Repeating this same process, I can guarantee I win if my opponent's turn lands on when there are 1,5,9,13,17,21,25,29 matches. To win, I must take 1 match at the beginning of the game (Winston 961).

This is the basic idea of Dynamic Programming. Start with a known ending subset, and use that subset answer to work back up each subset to answer the complex problem. A more applicable to the real-world would be a transportation problem. For example, minimizing the number of miles traveled if one is traveling from New York City to Los Angeles and stopping three times along the way. First stopping in either Columbus, Nashville, or Louisville, then stopping in either Kansas City, Omaha, or Dallas, then San Antonio or Denver. The distance three times along the way. First stopping in either Columbus, Nashville, or Louisville, then stopping in either Kansas City, Omaha, or Dallas, then San Antonio or Denver. The distance between every city is known and represented in the chart on the Reference Page. One may decide to start from New York, and calculate every combination of cities and distances and get the right answer. However, the advantage to dynamic programming is its easiness and speed.

One method of dynamic programming is to think about each step and write it all out. For example, since the goal is to end in Los Angeles. there are two choices of cities for the next subset, Denver and San Antonio. From Denver to Los Angeles is 1,030 miles, and from San Antonio to Los Angeles is 1,390 miles. The next subset is from either Kansas City, Omaha, or Dallas. Writing every city name out and calculating each distance will work, however, the process can be simplified to a simple algorithmic mathematical expression. First, simplify the cities to numbers instead of names, and label each stage the city options he can drive to. The figure of this is on the Reference Page. The variables c_{ij} and $f_t(i)$ are introduced. The road mileage between city i and city j is c_{ij}, and the length of the shortest path from city i in stage t to Los Angeles is $f_t(i)$.

Now to solve this problem, start off with the Stage 5, whose distances to city 10 are known. The distance from city 8 to 9 is c_{89} $+f_4(10)$, 1030+0= 1030 miles. The distance from city 9 to city 10 is $c_{9,10}$ $+f_5(10)$, 1390+0= 1390miles. The next subset is Stage 3, from cities 5, 6, or 7. The sub-problem from city 5 to either city 8 or 9 is represented by the following expression;

$$f_3(5) = \min \begin{cases} c_{58} + f_4(8) = 610 + 1{,}030 = 1{,}640^* \\ c_{59} + f_4(9) = 790 + 1{,}390 = 2{,}180 \end{cases}$$

The * indicates the shortest path. Solve the similar algorithm for each city 6, and 7. For the next stage, use the shortest distance from stage 3;

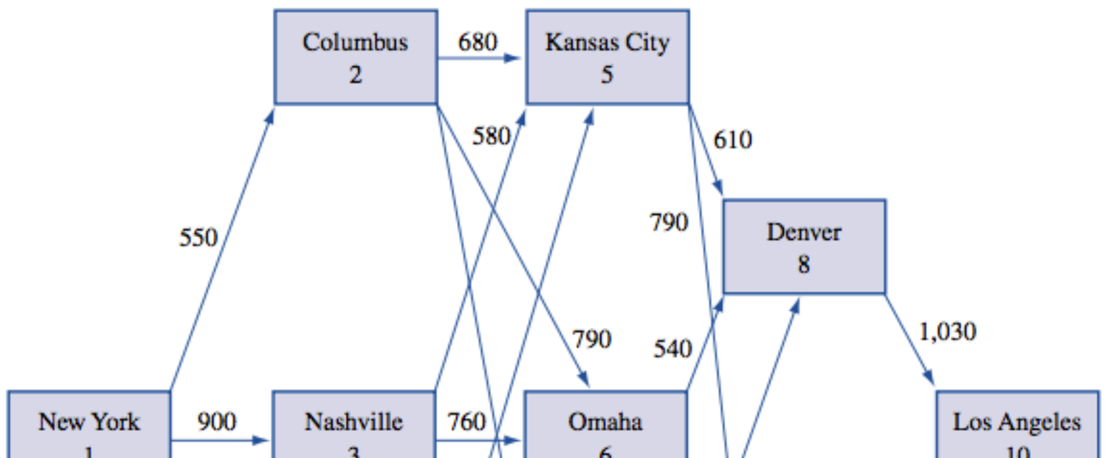
$$f_2(2) = \min \begin{cases} c_{25} + f_3(5) = 680 + 1{,}640 = 2{,}320^* \\ c_{26} + f_3(6) = 790 + 1{,}570 = 2{,}360 \\ c_{27} + f_3(7) = 1{,}050 + 1{,}660 = 2{,}710 \end{cases}$$

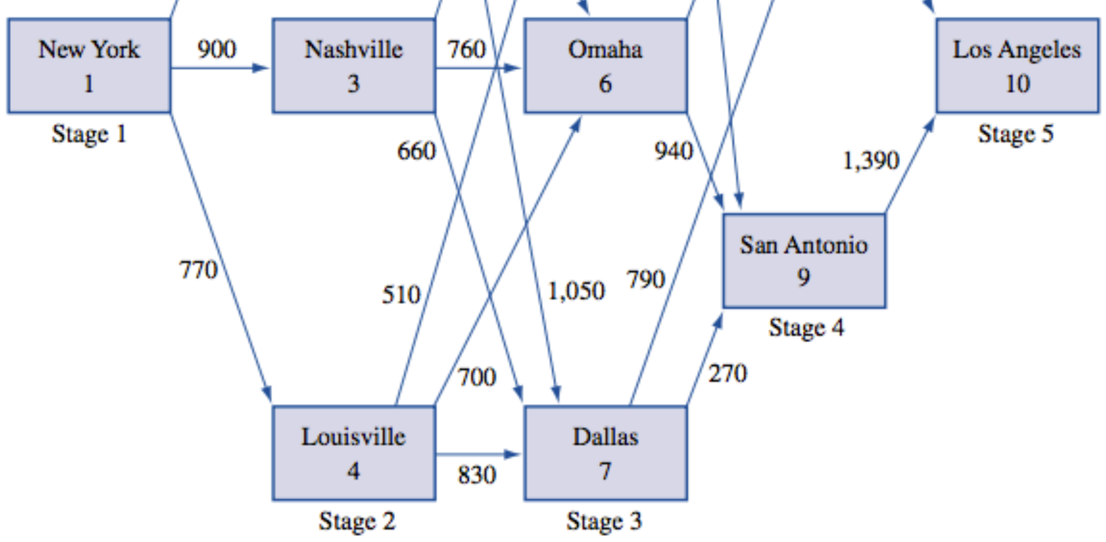
Repeat this algorithm until the shortest path from New York to Los Angeles is known to be New York, Columbus, Kansas City, Denver, and Los Angeles. This path has a length of 2,870 miles (Winston 963-965). Using this mathematical algorithm, this problem can be quickly solved. The other method of solving this problem would be to run through every iteration starting from New York. Although one will eventually get the right answer, the process will take considerably longer than dynamic programming.

Books on dynamic programming go over the topics of Fibonacci Sequence, Markovian Decision Process, Games, Loop Optimization, Deriving the Euler Condition, Lagrange Multipliers, The Euler Equation, Boundary Problems, Hamilton-Jacobi Equation, Satellite Trajectory Problems, etc. The list continues indefinitely. From these examples, it is clear that dynamic programming is very useful for a plethora of topics (Nemhauser) (Bellman).

The name dynamic programming is quite fitting for this adaptable method, although no programming experience is necessary. This method of solving a problem is quick and speedy, faster that iterating, and can take a seemingly difficult problem and turn it into a quickly solvable and programmable question. Its widespread use in Computer Science, Applied Mathematics, and Pure Mathematics, this technique is incredibly important to understand.

Reference Page





Works Cited

Bellman, Richard Ernest and Stuart E. Dreyfus. *Applied Dynamic Programming, by Richard E. Bellman and Stuart E. Dreyfus*. Princeton, N.J., Princeton University Press, 1962., 1962. EBSCO*host*, proxy.library.stonybrook.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=cat03000a&AN=STB.000608912&site=eds-live&scope=site.

Dreyfus, Stuart. "Richard Bellman on the Birth of Dynamic Programming." *Operations Research*, vol. 50, no. 1, 2002, pp. 48–51. *JSTOR*, JSTOR, www.jstor.org/stable/3088448.

Nemhauser, George L. *Introduction to Dynamic Programming [By] George L. Nemhauser*. New York, Wiley [1966], 1966. Series in decision and control. EBSCO*host*, proxy.library.stonybrook.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=cat03000a&AN=STB.000361526&site=eds-live&scope=site.

Winston, Wayne L. and Jeffrey B. Goldberg. *Operations Research : Applications and Algorithms / Wayne L. Winston ; with Cases by Jeffrey B. Goldberg*. Belmont, CA : Thomson/Brooks/Cole, ©2004., 2004. EBSCO*host*, proxy.library.stonybrook.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=cat03000a&AN=STB.001692238&site=eds-live&scope=site.