Dynamic programming

Research · November 2018

CITATIONS
0

READS
10,149

1 author:

Sajil Neissery Abdulla

1 PUBLICATION **0** CITATIONS

SEE PROFILE

Dynamic programming

Sajil N A

November 17 2018

1 Overview

Section 3 introduces dynamic programming, an algorithm used to solve optimization problems with overlapping sub problems and optimal substructure. Section 4 discusses two important characteristics that a problem should have to be solved using dynamic programming. In section 5, I have discussed about one of the possible way of storing the solution of subproblems to a given problem in tabular form. Section 6 Shows how to find the optimal solution for a longest common subsequence problem using dynamic programming. Section 7 deals with memoization which can be of interest to the reader.

2 Prerequisites

This paper is expected to be accessible for someone who have done a course in discrete mathematics. Formulating recursion and a basic understanding of graph theory is required for better understanding this paper. Most of the concepts and terminologies used in this paper are explained in great detail for reaching out to maximum audience. I have included references for some of the subtopics used in this paper to have a clear picture of the main topic. Going through these references before reading this paper will help the reader to have a better experience.

1.Recursion

2.Sequence and string

3 Introduction

Dynamic programming is both a mathematical optimization and computer programming method developed by an American mathematician Richard Bellman. The term dynamic programming was coined by Richard Bellam in 1940s. He used this algorithm to describe the process of solving problem where one is asked to find best decision at each stage. He used the word dynamic to show the time-varying aspect of the problems and also it sounded impresseive for him. The word programming is used in the same sense as that it is being used in linear programming or mathematical programming.

Dynamic programming is a method for solving a complex problem by breaking down the given problem into a number of sub problems and solving these sub problems once and storing the solution to these sub problems in a table. Generally, dynamic programming is applied to optimization problems. Dynamic programming is applied when there is an overlapping between sub problems of the same problem. If applicable to some problem, it takes less time than naive methods. It can be used to solve problem in time polynomial time for which a naive approach would take exponential time. Dynamic programming is one of the most popular

algorithm used in bioinformatics. Dynamic programming is used in various process including sequence comparison, gene recognition and many other problems.In this paper we will see how to compare DNA of two organism using dynamic programming.

3.1 Subsequence

Definition 1 : If we define a sequence P as $P =< p_1, p_2, ..., p_m >$, then we can define a subsequence R as $R =< r_1, r_2, ..., r_k >$ for the given sequence P. This is true if and only if R is derived from P. That is, elements of the subsequence R is chosen from the sequence P in a strictly increasing order.

3.2 Common subsequence

Definition 2 : Let us consider two sequences P and Q where P is defined as $P =< p_1, p_2, ..., p_m >$ and Q as $Q =< q_1, q_2, ..., q_n >$. Then the common subsequence of P and textitQ is defined as $R =< r_1, r_2, ...r_k >$, where R is a subsequence of both P and Q.

For example, if $P =< AGCGTAG >$ and $Q =< GTCAGA >$, then by inspection we get the length of the common subsequence as 4 and one of the common subsequence as GTAG.

3.3 Four steps involved in developing a dynamic programming algorithm:

1.Identify the structure of the optimal solution in a given problem.

2.Use recursion to define the value of an optimal solution.

3.Find the optimal value using bottom up approach.

4.Construct an optimal solution for the given problem using the above information.

4 Elements of dynamic programming

In order to apply dynamic programming algorithm in a problem, the problem must satisfy two properties. They are optimal substructure and overlapping subproblems. Subsequent sections discuss about these two properties.

4.1 Optimal Substructure

A problem is said to have an optimal substructure if the problem has an optimal solution which is contained in the optimal solution of the subproblmes. ie, an optimal solution to the problem is obtained by combining the optimal solutions of the subproblems. While solving the problem using dynamic programming we must ensure that the range of subproblems we consider includes those that can be used in an optimal solution.

This property can be understood by the given example from graph theory. The shortest path p from a vertex a to a vertex c in a given graph exhibits optimal substructure. Take any intermediate vertex b on this shortest path p. If the initial choice that p is the shortest path between the vertex a and c is true, then there exists intermediate shortest paths between ab (shortest path between a and b) and bc (shortest path between b and c).

4.2 Overlapping subproblems

An optimization problem have overlapping subproblems if the recursive algorithm repeatedly solves the same subproblem. As mentioned before the advantage of dynamic programming is that it solves the overlapping subproblem only once and then store the solution of the same in a table which can be used later. This can be understood by the problem of computing Fibonacci sequence where one compute Fibonacci number recursively. The problem of computing n^{th} Fibonacci number F_n includes computing F_{n-1} and F_{n-2} and adding the two. In computing F_{n-1} we again need to compute F_{n-2} which is already computed to find F_n.Therefore by computing F_{n-2} for F_{n-1} and by storing the same, we avoid the repetition of computing it.

5 Storing the values of the optimal solution

In general, there are two ways by which we can store the solution to the subproblems of a given problem:

1. Bottom up approach

2. Memoization (This method is discussed in section 7 of this paper)

5.1 Bottom up approach

In dynamic programming, we use a table to store the computed values of the optimal solutions to the subproblems. Once we have formulated the solution to a given problem recursively using the solutions to the subproblems, using this we solve the subproblems first and combine the solution to the subproblem to arrive at the solution to a given problem. This is usually done in a tabular form by using the computed solution of the subproblems from the table.

6 Longest common subsequence (LCS)

Longest common subsequence problem is the problem of finding the longest common subsequence in given two sequences in same relative order. Bioinformatics is one of the important area where comparison of two sequences is used. Comparing DNA of two organism to understand the similarity between two organism is often required in the field of bioinformatics.

As mentioned before the brute force way of finding the longest common sequence will take exponential time. If we have a sequence P with m elements then it will have 2^m subsequences. Comparing each subsequence of P with another subsequence Q will result in exponential time. This is practically impossible for longer sequences. In the subsequent sections we shall see that the LCS problem satisfies optimal substructure property and overlapping subproblems.

STEP 1: Identifying the structure of the LCS problem

Theorem 1: Optimal substructure of an LCS

Let $P_m =< p_1, p_2, ..., p_m >$ and $Q_n =< q_1, q_2, ..., q_n >$ be two sequences and the LCS of P and Q is given by R, where $R =< r_1, r_2, ..., r_k >$.

1. If $p_m = q_n$, then $r_k = p_m = q_n$ and R_{k-1} is an LCS of P_{m-1} and Q_{n-1}

2. If $p_m \neq q_n$, then $r_k \neq p_m$ implies that R is an LCS of P_{m-1} and Q

3. If $p_m \neq q_n$, then $r_k \neq q_n$ implies that R is an LCS of P_m and Q_{n-1}

Proof :

(1)Assume to the contrary, that is, if $r_k \neq p_m$ then by adding $p_m = q_n$ to the sequence R we obtain common subsequence of P and Q. This common subsequence will be of length k+1, which is a contradiction. Since maximum length of the common subsequence is k. Thus $r_k = p_m = q_n$.Consider the common subsequence(R_k) of P_{m-1} and Q_{n-1} with length k-1. Now we have to show that this is an LCS. In order to show that this is an LCS, assume to the contrary that there exists a common subsequence S of P_{m-1} and Q_{n-1} with length greater than k-1. But we know that if we add $p_m = q_n$ to S, it will give a common subsequence of length greater than k. This is a contradiction. Therefore, R_{k-1} is an LCS of P_{m-1} and Q_{n-1}

(2)Suppose for the purpose of contradiction assume that there exists a common sebsequence(S_k) of P_{m-1} and Q with length greater than k, then S would be a common subsequence of P_m and Q also. This contradicts our earlier assumption that R is an LCS of P and Q. Therefore R is an LCS of P_{m-1} and Q.

(3)Suppose for the purpose of contradiction assume that there exists a common sebsequence(S_k) of P and Q_{n-1} with length greater than k, then S would be a common subsequence of P and Q_n also. This contradicts our earlier assumption that R is an LCS of P and Q. Therefore R is an LCS of P and Q_{n-1}.

Overlapping subproblems of an LCS

The argument given below shows that LCS problem satisfies the overlapping subproblem property.

In order to find the LCS of P and Q, we have to find the LCS of sequences P & Q_{n-1} and P_{m-1} & Q (by case 2 3 of theorem 1). But we can see that finding an LCS of P & Q_{m-1} and P_{n-1} & Q involves finding an LCS of P_{m-1} and Q_{n-1}. Therefore LCS problems satisfies the second property to be solved using dynamic programming.

STEP 2: Defining a recursion for the optimal solution

Given two sequences P_i and Q_j, where $P_i =< p_1, p_2, ..., p_i >$ and $Q_j =< q_1, q_2, ..., q_j >$. Now, let us define LCS$[i, j]$ to be the length of an LCS of the sequence P and Q. The optimal structure of the LCS problem is given by the following recursion.

$$\text{LCS}[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0 \\ \text{LCS}[i-1, j-1] + 1, & \text{if } i, j > 0 \text{ and } p_i = q_j \\ \max(\text{LCS}[i, j-1], LCS[i-1, j]), & \text{if } i, j > 0 \text{ and } p_i \neq q_j \end{cases} \quad (1)$$

Theorem 1 and the above recursion (1) can be well understood by the below example.

Lets look at the LCS tree Of two sequences P and Q, where $P =< AGCGTAG >$ AND $Q =< GTCAGA >$.

LCS[AGCGTAG,GTCAGA] = max(LCS[AGCGTA, GTCAGA],LCS[AGCGTAG,GTCAG])

(a) LCS[AGCGTA,GTCAGA] = max(LCS[AGCGT, GTCAG] + A (theorem 1)

(b) LCS[AGCGTAG,GTCAG] = max(LCS[AGCGTA, GTCA] + G (theorem 1)

Then again applying theorem 1 on (a) and (b) we get,

LCS[AGCGT,GTCAG] = max(LCS[AGCG,GTCAG],LCS[AGCGT,GTCA])

LCS[AGCGTA,GTCA] = LCS[AGCGT,GTC] + A.

and so on.

Thus by applying theorem 1 to the above problem further we can see that the LCS problem satisfies optimal substructure property. Since the LCS problem satisfies both optimal substructure and overlapping subproblems property we can apply dynamic programming to LCS

Solving LCS problem using dynamic programming

Now we will see a worked example of longest common subsequence for two given sequences P' and Q' using dynamic programming, where $P' =< AGCGT >$ and $Q' =< TGCAT >$

STEP 3: Finding an optimal value of the LCS

The pictorial representation of the above defined recursion (1) for sequence P' and Q' with P' along the top row and Q' along the leftmost column is shown below.

Table 1: **LCS Table**

	0	A	G	C	G	T
0	0	0	0	0	0	0
T	0	0	0	0	0	1
G	0	0	1	1	1	1
C	0	0	1	2	2	2
A	0	1	1	2	2	2
T	0	1	1	2	2	3

We start by filling zeros in the first row and first column of the table 1. Next we will see if the second element on the left most side of the table is same as the second element of the sequence given above the table. If the elements are different then the value entered in the corresponding cell will be the maximum of the values in the cells right above and left to the respective cell. If the elements matches then the value entered in the corresponding cell will be the value of the diagonal value plus one. We continue the same process till we reach the right bottom corner of the table. The optimal value of LCS will be present at the bottom right corner of the table.

Optimal value of the LCS for the sequences P' and Q' is 3.

STEP 4: Computing the optimal solution from the above information

To find the optimal solution for the above sequences we trace back from the lower right corner of table 2. we will look at the cell directly above or directly to the left and see if the value in those cells are equal or greater than the value of the current cell. If both the cells have equal value, choose one of those cells and move up. If both such cells have values lesser than the current cell then move diagonally up-left. Continuing the same process will give the longest common subsequence in reverse order. The table below shows the optimal solution for the sequences P' and Q' is obtained using the above defined technique.

LCS table showing optimal solution of P' and Q'

Table 2: **LCS Table showing the bottom-up approach**

	0	A	G	C	G	T
0	0	0	0	0	0	0
T	0	0	0	0	0	1
G	0	0	1	1	1	1
C	0	0	1	2	2	2
A	0	1	1	2	2	2
T	0	1	1	2	2	3

One of the LCS for the given sequences would be GCT (table 2). To explain in simpler terms, the element corresponding to the cell where the tail of the arrow begins is a member of LCS in reverse order, if one use the bottom up approach to find LCS.

7 Additional Exploration

7.1 Memoization/Top down approach

This is an alternate method for bottom up approach for solving a problem using dynamic programming. Memoization can be used if a problem can be solved recursively using the solution to its subproblems and if the subproblems are overlapping. This method also requires storing the values in a table but in a top down approach. Unlike tabulation, memoization does not fill up all cells unless it it definitely required. Each cell is filled based on demand. ie, Memoization does not ask for filling up of all cells of a table to reach the final answer.

8 References

1.Thomas H Cormen,Charles E. Leiserson,Ronald L. Rivest and Clifford Stein. Introduction to algorithms,third edition. pp 360-395

2.Longest common subsequence—Introduction LCS length,Techie delight,Coding made easy

3.MIT course on algorithms-Includes a video lecture on DP along with lecture notes

4.Dynamic programming and Longest common subsequence, Geeks for geeks,computer science portal for geeks

5.Richard bellman on the birth of dynamic programming,Stuart Dreyfus,university of California, Berkeley.