HBRP PUBLICATION

Journal of Network Security and Data Mining
Volume 6 Issue 2

Deepen Your Knowledge of Dynamic Programming in Data Science

***Deepak Prajapat*[*1], *Akanksha Kulkarni*[2]**
[1] *Student, Dept. School of Engineering, Ajeenkya D.Y. Patil University, Pune, India*
[2]*Professor, Dept. School of Engineering, Ajeenkya D.Y. Patil University, Pune, India*

***Corresponding Author**
E-mail Id:- *deepak27prajapat@gmail.com*

ABSTRACT
Dynamic programming is an area that is often not well understood by those learning algorithms for the first time, but it is a crucial part that should be studied. This technique has been effectively used in numerous fields, including controlling human movement, distributing hydroelectric resources, and gene sequencing. This article provides a detailed explanation of the dynamic programming principle, comparing it to other algorithms to help readers understand its nature, benefits, and drawbacks compared to alternative problem-solving techniques. Using relevant application examples, it explores the stages and techniques involved in dynamic programming problem-solving.

Keywords:-*Knapsack problem, Memory recursion, and Dynamic programming.*

INTRODUCTION

Using the dynamic programming technique, it is possible to solve the optimal solution problem for multi-stage decision-making. Despite its name, it is not as "dynamic" as one might assume. To tackle a practical problem, this technique establishes a starting point and breaks the larger problem down into smaller subproblems.

The solution to the previous subproblem can then be used to solve the current subproblem. The state transition equation, which describes the relationship between the previous and current subproblems, is at the heart of this approach and is also the source of its difficulty.

Once the state transition equation has been determined, the sub-solutions are gradually computed from the bottom to the top of the problem's original state in order to resolve the larger overall issue.

ESSENTIAL CONCEPT OF DYNAMIC PROGRAMMING

Before applying dynamic programming to solve a problem, it is necessary to determine whether the problem has ideal substructure features, overlapping subproblem characteristics, and the absence of consequences.

"Optimal substructure" refers to the property that the best solution to a problem includes the best solutions to all its subproblems. When a problem is divided into subproblems, overlapping subproblems indicate that there are no aftereffects, meaning that the subsequent decisions made in that state will not have any impact on it.

Dividing a task into several stages is the fundamental concept of using dynamic programming to solve problems, and each stage can have multiple states. These states can be used to determine the outcome of the current stage and the values of each

state in the following stage. This process continues until the solution to the final stage is found, which represents the solution to the entire problem.

In general, when addressing an issue using dynamic programming, the approach should be top-down. We need to solve the problem by first addressing the earliest stage of the issue, where there may be multiple states. The selection of any one of these states could potentially be the solution to the initial issue, and this is what the transfer equation needs to evaluate.

These states are determined by the final step, and this process is repeated until the starting state is reached. However, the actual calculations are done in a bottom-up manner, starting with the initial state. Calculations are made to determine the solution for each state in the first stage, and then using these conclusions, the solutions for the subsequent states can be determined until the solution for the final stage is obtained.

CONNECTED OTHER ALGORITHMS

Greedy Method

In addition to dynamic programming, using the greedy approach is another highly effective method for solving optimization problems. However, the problem must meet the criteria for greedy selection to be used. Local optimal selection is stricter compared to the application of dynamic programming, and it can yield the overall best solution. While the dynamic programming approach can typically solve the issues that can be resolved by the greedy technique, the greedy method might not be able to tackle all the problems that dynamic programming can solve. It can be thought of as a unique instance of dynamic programming, where the greedy approach only considers the current state, while dynamic programming also considers the past states.

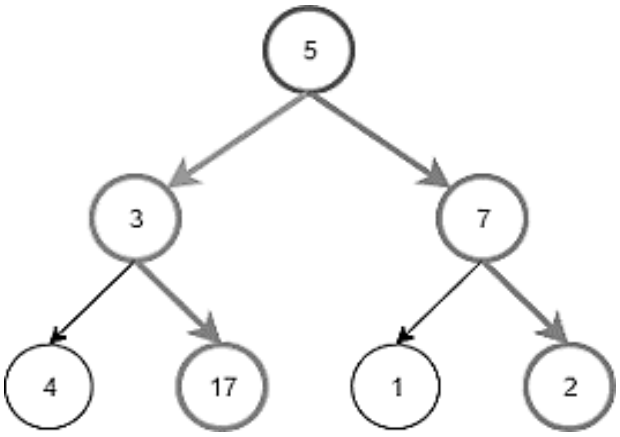

Fig.1:*-Example of Greedy Method*

Divide and Conquer

The dynamic programming algorithm is essentially a variation of the divide and conquers strategy, as they both break down a major problem into smaller ones and deal with each one separately. However, the dynamic programming method differs in that a sub-problem may occur many times, and solving the latter problem also necessitates solving the first due to the overlap of sub-problems. Hence, storing these subproblems is considered, so that their solutions can be easily accessed while tackling larger sub problems,

eliminating redundant calculations to improve algorithm efficiency.

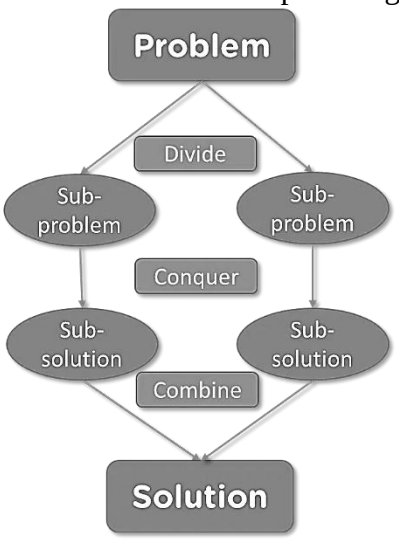

Fig.2:*-Example of Divide and conquer*

Memory Recursion

Memory recursion is also utilized to tackle the problem with the concept of the space-for-time algorithm, similar to the dynamic programming approach, and they actually have the same essence. Yet, the dynamic programming approach works from the bottom up whereas the memory recursion solves problems from the top down. The two can typically be used interchangeably. The cache in memory recursion is analogous to the dp table in dynamic programming, thus in dynamic programming, the state transition equation is the same as a recursive calling. The conversion between the two is somewhat comparable to that between recursion and loop.

APPLICATION

Steps for Resolving Issues

Once an issue arises, the first thing we must consider is if dynamic programming can be used to address it. Considering if the answer is optimal is necessary if dynamic programming can be used to resolve the issue. Dynamic programming can be used to solve the problem if it meets the prerequisites of the ideal sub-structure, the similarity of sub-problem characteristics, and the absence of a consequence. We are now considering if it is best to employ dynamic programming to resolve this issue. Assume that there are n phases to this problem, and each stage contains m states. Recursion can be used to solve this problem. This issue can be resolved using the greedy method when m is equal to 1 if each stage's ideal state is derived from the optimal state of the stage before it; Dynamic programming can be used to address this problem if a state from a previous step serves as the foundation for the ideal condition at each level.

Once it has been determined that this problem can be solved using dynamic programming, it is broken down into many steps based on its specific characteristics. We must employ various states to reflect the problem's current objective reality once it reaches a given level of development. The transition equation, or link between a stage's current state and its predecessor stage's current state, is what we need to discover. Prior to that, we must first determine the beginning state and make sure the state we choose has no consequences. Find the best solution at each level in accordance with the transfer

equation, and then locate the answer to the initial problem by finding the best solution at the last stage.

Application Examples

A well-known issue with dynamic programming is the 0-1 knapsack problem, which is also worthwhile understanding because it may be used to solve a variety of other problems. The issue is described in the following way: Given a rucksack with a capacity of W, n objects with weights w1, w2, and wn and values v1, v2, vn are present. Create a strategy for choosing a few of these goods to put in the rucksack. Either one of the items is chosen or not. The chosen goods must have the highest worth in addition to being able to fit in the rucksack. The first thought that comes to mind is typically a pretty violent recursive one. There are two options for each item: either place it in the backpack or do not place it in the backpack. This gives us the occurrence: f(n,W)=max(f(n-1,W),f(n-1,W-wn)+Vn). The largest value that is possible after packing the first n things into a bag with a W-liter capacity is represented by f(n, W) among them. We have made decisions for each recursive stage, including whether to select, which means Citing the case of each decision. Recursively go through each node on the solution set tree to find the 0-1 knapsack problem's proper answer.

In reality, there are many repeated answers, therefore we came up with the idea in order to avoid having to repeat each recursive solution in the future, we created a two-dimensional array to store the results. The memory recursion approach looks like this.

Actually, dynamic programming and the memory recursion method are pretty similar. As was already mentioned, the two vary in that one is bottom-up and the other is top-down. In fact, they may also be thought of as the way recursion and looping interact. Theoretically, loop and recursion are interchangeable. Consequently, the dynamic programming method's solution can be reached by turning with a two-dimensional array and the dynamic transfer equation that was previously memorized, turning the recursive process into a loop. Each item's choice can be viewed as a stage in the dynamic programming problem.

CONCLUSIONS

Using the recursive solution approach, finding the transfer equation, which is the same as the recursive formula, is the main objective of the dynamic programming approach. This article presents the fundamental concepts, steps for solving problems, and examples of applications of the dynamic programming method, and specifically explains how the dynamic programming approach differs from other approaches in terms of conversion relations is discussed. The essence of dynamic programming is evaluated and taken into consideration through comparison with other algorithms: apply the solutions to old issues to solve new ones.

REFERENCES

1. Levitin. A. (2019). Rethinking algorithm design and analysis. *32*(1):14-20.
2. Pferschy U, Scatamacchia R.(2017). Results of improved dynamic programming and approximation for the setups knapsack problem. *25*(2): 677-662.
3. Dereventsov D. B., Temlyakov V. F.(2022). A methodical approach to studying several greedy algorithm. *227*(12): 69-54.
4. Teaching Algorithms. SIGACT News, 36 (December 2015), 58–56. Baeza-Yates, R.
5. Bentley, J. (2016). *Programming pearls*. Addison-Wesley Professional.. Brassard, G., & Bratley, P.

(1996). *Fundamentals of algorithmics*. Prentice-Hall, Inc...
6. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms*. MIT press. Computer Algorithms, 2019, Computer Science Press, Horowitz et al
7. Kernighan, B. W., & Pike, R. (1999). *The practice of programming*. Addison-Wesley Professional..
8. Levitin, A (2013)Should we teach the correct algorithm design techniques?. 179–183 in Proc. SIGCSE '99 . Neapolitan, R., & Naimipour, K. (2010). *Foundations of algorithms*. Jones & Bartlett Publishers.
9. Polya, G. (2004). *How to solve it: A new aspect of mathematical method* (No. 246). Princeton university press. Rawlins. J. (2019). Compared To What? : an Introduction to the Study of AlgorithmsComp. Sc. Press.
10. Algorithms, R. Sedgewick, 2020, Addison-Wesley.
11. Skiena, S. (1997). Minkowski Sum. The Algorithm Design Manual.. Vandevoorde, D. (1998). The maximal rectangle problem. *Dr. Dobb's journal*, *23*(4).