Course: Advanced Algorithm and Problem Solving

WEEK 4 Dynamic Programming

Lemlem Kassa (Ph.D.)
Addis Ababa Science and Technology University, Ethiopia

April ,2025

Week 4: Dynamic programming

Content

- Introduction to dynamic programming
- Approaches of dynamic programming
- Examples of dynamic programming
- Greedy algorithms vs dynamic Programming
- Applications of dynamic programming
- Advantages and disadvantages of dynamic Programming

Lecture Learning Outcome

- Understand Dynamic Programming and the key concepts: overlapping and optimal substructure
- Understand the approaches of dynamic programming with different examples
- Understand the difference b/n Greedy Algorithms and Dynamic Programming
- Identify the applications of dynamic programming
- Understand the advantages and disadvantages of dynamic programming

Introduction to Dynamic Programming

- Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems.
- But unlike divide and conquer, these sub-problems are not solved independently.
- Rather, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems.
- Mostly, dynamic programming algorithms are used for solving optimization problems. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems.
- The solutions of sub-problems are combined in order to achieve the best optimal final solution.

[1]. TutorialsPoint, Dynamic Programming concepts,
https://www.tutorialspoint.com/data_structures_algorithms/dynamic_programming.htm

Introduction to Dynamic Programming …(cont'd)

Key Concepts of Dynamic Programming Algorithm:

a) Overlapping Subproblems

- The problem can be broken down into smaller subproblems, where the solutions to the subproblems are overlapping.
- Having subproblems that are overlapping means that the solution to one subproblem is part of the solution to another subproblem.

Example:

- In the Fibonacci sequence, calculating Fib(5) involves calculating Fib(4) and Fib(3), and calculating Fib(4) again involves Fib(3) and Fib(2). Without dynamic programming, these calculations would be repeated many times.
- Thus, by storing the results of Fib(2), Fib(3), and so on, we avoid these repeated calculations, making the algorithm much more efficient.

[2]. W3Schools, DSA Dynamic Programming, https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

Introduction to Dynamic Programming …(cont'd)

Key Concepts of Dynamic Programming Algorithm:

b) Optimal Substructure:

- If the optimal solution of the given problem can be obtained using optimal solutions of its sub-problems.
- The substructure must also be optimal so that there is a way to piece the solutions to the subproblems together to form the complete solution.

Example,

- In the shortest path problem, if we know the shortest path from A to B and the shortest path from B to C, then the shortest path from A to C will be the combination of these two shortest paths.
- Dynamic programming uses this concept to build up the solution to a larger problem by solving and combining the solutions to smaller subproblems.

[1]. TutorialsPoint, Dynamic Programming concepts, https://www.tutorialspoint.com/data_structures_algorithms/dynamic_programming.htm

Approaches of Dynamic Programming

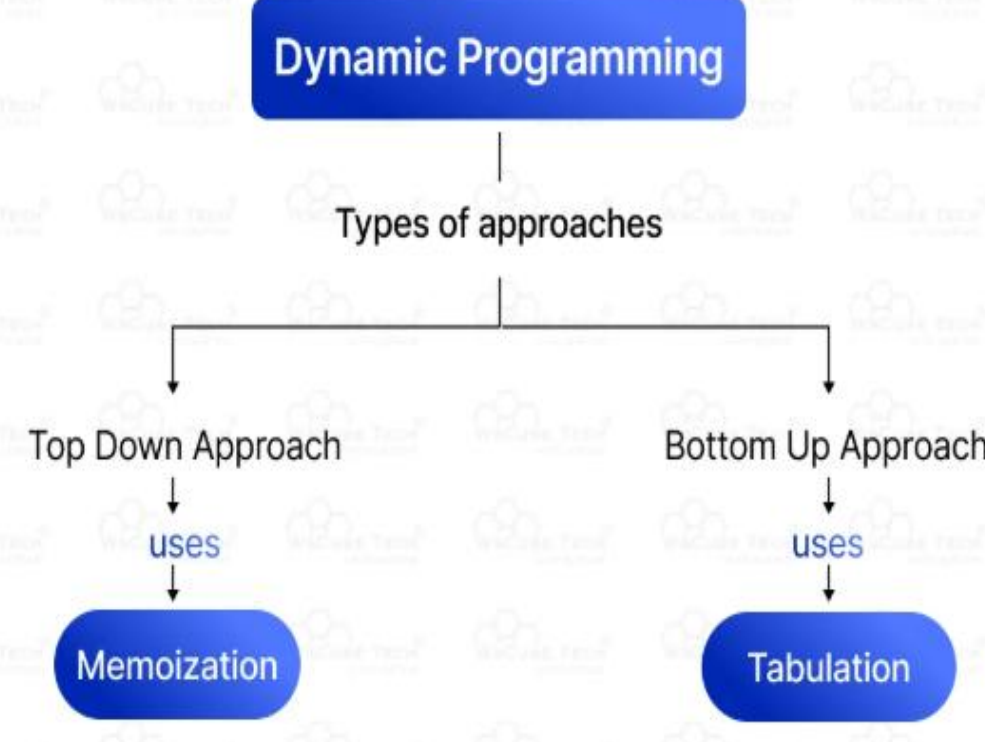
The two main approaches to Dynamic Programming.

1. Top-Down Approach (Memoization)

- It start with the main problem and break it down into smaller subproblems recursively.
- As we solve each subproblem, we store (or "memoize") its result in a table or cache.
- If the same subproblem is encountered again, the stored result is used instead of recalculating it.
- This approach avoids redundant calculations and speeds up the solution.

Example:

- Calculating **Fibonacci numbers** using recursion and storing previously computed values to avoid recalculating them.



[2]. W3Schools, DSA Dynamic Programming, https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

Approaches of Dynamic Programming …(cont'd)

2. Bottom-Up Approach (Tabulation)

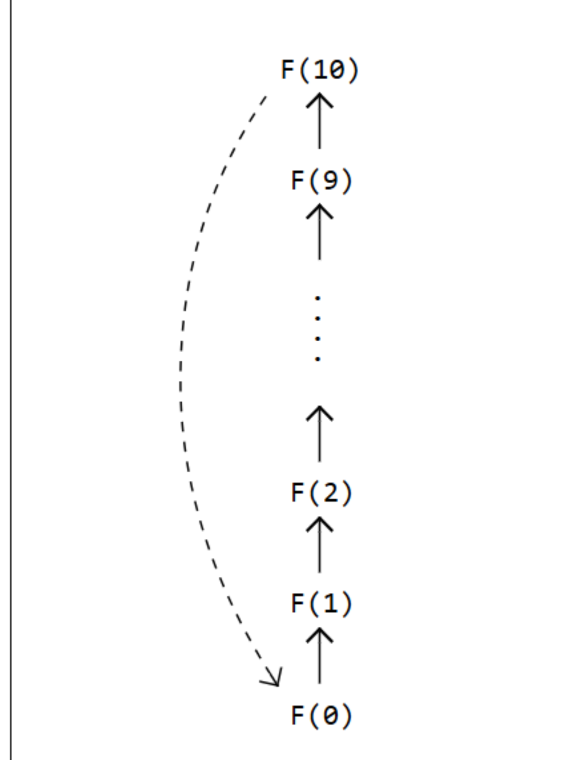
- Solve the smallest subproblems first and use their results to build up the solution to larger subproblems.
- This is done iteratively, filling up a table (or "tabulation") until the final solution to the main problem is reached.
- This approach is often more space-efficient and avoids the overhead of recursive calls.

Example:

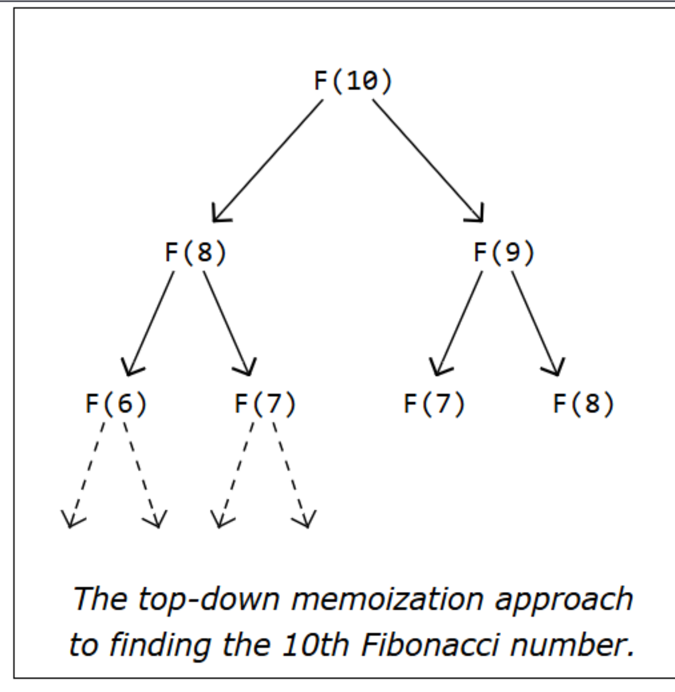
- Calculating **Fibonacci numbers** by iteratively filling an array from the smallest numbers (Fib(0), Fib(1)) to the desired number (Fib(n)).

[3]. WsCube Tech, Dynamic Programming (With Problems & Key Concepts), https://www.wscubetech.com/resources/dsa/dynamic-programming

Approaches of Dynamic Programming …(cont'd)



The bottom-up tabulation approach to finding the 10th Fibonacci number.



The top-down memoization approach to finding the 10th Fibonacci number.

- As you can see in the images above, the tabulation approach starts at the bottom by solving F(0) first, while the memoization approach start at the top with F(10) and breaking it into smaller and smaller subproblems from there

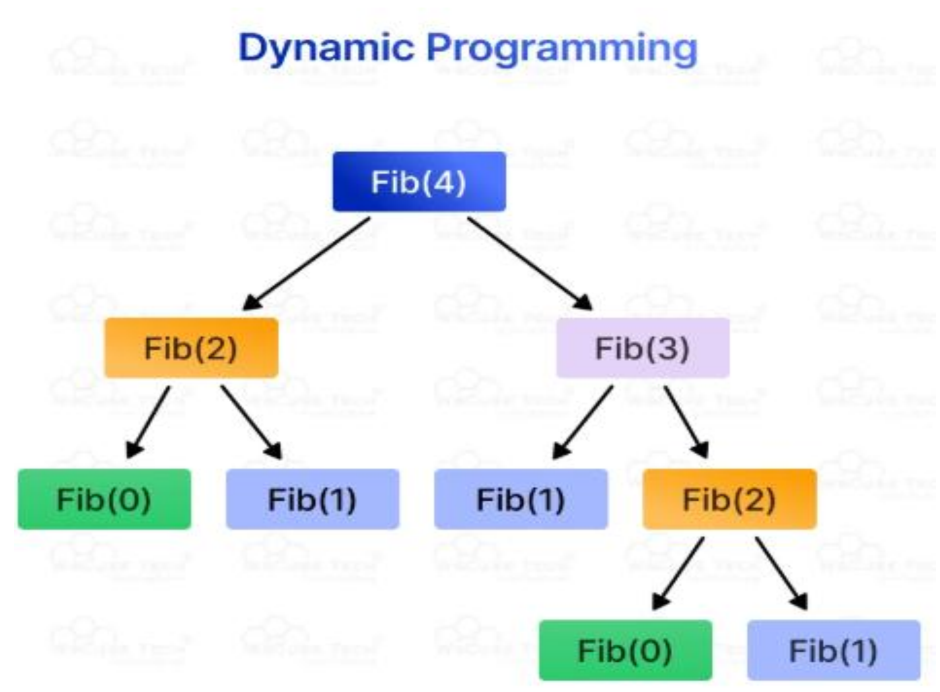
[2]. W3Schools, DSA Dynamic Programming, https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

Approaches of Dynamic Programming …(cont'd)

Find The nth Fibonacci Number using Dynamic Programming

- The Fibonacci numbers is a sequence of numbers starting with 0 and 1, and the next numbers are created by adding the two previous numbers.
- The 8 first Fibonacci numbers are: 0,1,1,2,3,5,8,13.
- And counting from 0, the 4th Fibonacci number F(4) is 3 .
- In general, this is how a Fibonacci number is created based on the two previous:

$$F(n)=F(n-1)+F(n-2)$$



Approaches of Dynamic Programming …(cont'd)

How to design an algorithm using Dynamic Programming?

1) Check if the problem has "overlapping subproblems" and an "optimal substructure".
2) Solve the most basic subproblems.
3) Find a way to put the subproblem solutions together to form solutions to new subproblems.
4) Write the algorithm (the step-by-step procedure).
5) Implement the algorithm (test if it works).

[2]. W3Schools, DSA Dynamic Programming, https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

Examples of Dynamic Programming

Example : Dynamic Programming to find the nth Fibonacci Number

Step 1: Check if the problem has "overlapping subproblems" and an "optimal substructure".

Does it have overlapping subproblems?

- Yes, the 6th Fibonacci number is a combination of the 5th and 4th. This shows that the problem of finding the nth Fibonacci number can be broken into subproblems.
- Also, the subproblems overlap because F(5) is based on F(4) and F(3), and F(6) is based on F(5) and F(4).
- Both solutions to subproblems F(5) and F(6) are created using the solution to F(4), and there are many cases like that, so the subproblems overlap as well.

Examples of Dynamic Programming …(cont'd)

Optimal substructure?

- The Fibonacci number sequence has a very clear structure, because the two previous numbers are added to create the next Fibonacci number, and this holds for all Fibonacci numbers except for the two first.
- This means we know *how* to put together a solution by combining the solutions to the subproblems.
- We can conclude that the problem of finding the nth Fibonacci number satisfies the two requirements, which means that we can use Dynamic Programming to find an algorithm that solves the problem.

[2]. W3Schools, DSA Dynamic Programming,
https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

Examples of Dynamic Programming …(cont'd)

Step 2: -

Solve the most basic subproblems.

- Solving the most basic subproblems first is a good place to start to get an idea of how the algorithm should run.
- In our problem of finding the nth Fibonacci number, finding the most basic subproblems is not that hard, because we already know that

$$F(0) = 0$$
$$F(1) = 1$$
$$F(2) = 1$$
$$F(3) = 2$$
$$F(4) = 3$$
$$F(5) = 5$$
$$F(6) = 8$$
$$\dots$$

Examples of Dynamic Programming …(cont'd)

Step 3:

Find a way to put the subproblem solutions together to form solutions to new subproblems.

- For our problem, how the subproblems are put together is quite straightforward, we just need to add the two previous Fibonacci numbers to find the next one.
- So, for example, the 2nd Fibonacci number is created by adding the two previous numbers:

$F(2) = F(1) + F(0)$, and that is the general rule as well, like mentioned earlier: $F(n) = F(n-1) + F(n-2)$

[2]. W3Schools, DSA Dynamic Programming,
https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

Examples of Dynamic Programming …(cont'd)

Step 4:

Write the algorithm (the step-by-step procedure).

- Instead of writing the text for the algorithm straight away, it might be wise to try to write a procedure to solve a specific problem first, like finding the 66th Fibonacci number.
- For reference, the 8 first Fibonacci numbers are: 0,1,1,2,3,5,8–,130,1,1,2,3,5,8_,13.

[2]. W3Schools, DSA Dynamic Programming, https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

Examples of Dynamic Programming …(cont'd)

Step 5:-

Implement the algorithm (test if it works).

- To implement the algorithm above, we assume that the argument n to the function is a positive number (the nth Fibonacci number), we use a for loop to create new Fibonacci numbers, and we return the base cases F[0] and F[1] straight away if the function is called with 0 or 1 as an argument.

```
def nth_fibo(n):
    if n==0: return 0
    if n==1: return 1

    F = [None] * (n + 1)
    F[0] = 0
    F[1] = 1

    for i in range(2, n + 1):
        F[i] = F[i - 1] + F[i - 2]

    return F[n]

n = 6
result = nth_fibo(n)
print(f"The {n}th Fibonacci number is {result}")
```

Examples of Dynamic Programming …(cont’d)

0/1 Knapsack Problem

- In the 0/1 Knapsack problem, a subproblem could be defined as finding the maximum value that can be achieved with a given capacity and a subset of the items

Rules:

- Every item has a weight and value.
- Our knapsack has a weight limit.
- Choose which items you want to bring with you in the knapsack.
- We can either take an item or not, you cannot take half of an item for example.

[2]. W3Schools, DSA Dynamic Programming, https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

Examples of Dynamic Programming …(cont'd)

- There are two versions of the problem:
  1. 0-1 knapsack problem
  2. Fractional knapsack problem

1. Items are indivisible; we either take an item or not.
   - Solved with **dynamic programming**
2. Items are divisible: we can take any fraction of an item.
   - Solved with a **greedy algorithm**.

[2]. W3Schools, DSA Dynamic Programming, https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

Examples of Dynamic Programming …(cont'd)

- Suppose the sets of profit and weight are respectively specified as follows: P={1,2,5,6} and W= {2,3,4,5}.
- Number of objects are 4 and the capacity of the knapsack is 8. n=4; wt=8

- Use tabulate procedure to understand the problem
- We add zeroes to the 0th row and 0th column because if the weight of item is 0, then it weighs nothing; if the maximum weight of knapsack is 0, then no item can be added into the knapsack.

- Recognize the maximum profit from the table and identify the items that make up the profit, in this example, its {3, 5}, with the maximum profit is 8.

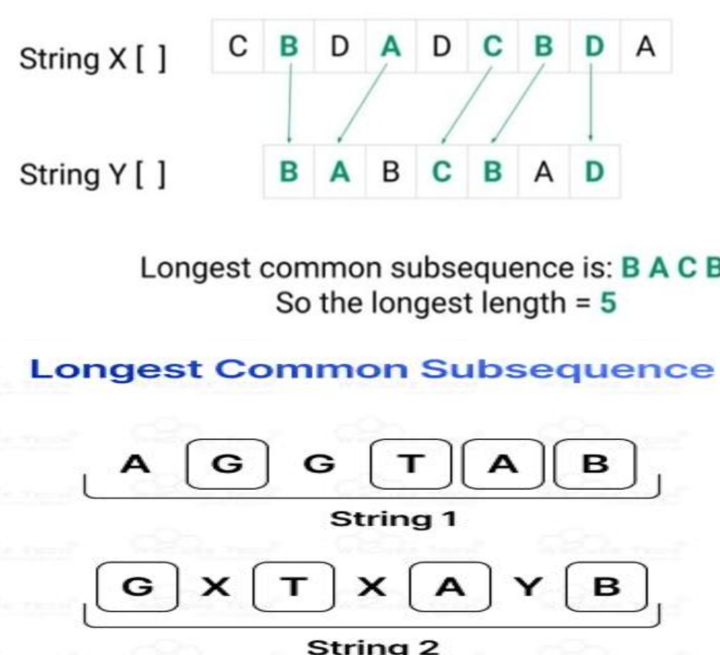
Increasing weights (→)		0(w→)	0	1	2	3	4	5	6	7	8
P_i	w_i	0(i↓)	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5	6	6	7	8

Table: 0/1 Knapsack Problem.

[4]. Techprodezza,0/1 Knapsack Problem, Tharun Buddigina, October 27, 2020, https://www.programiz.com/dsa/dynamic-programming

Examples of Dynamic Programming …(cont'd)

- The **Longest Common Subsequence (LCS)** is a way to find the longest sequence that is common to two strings, but the characters in this sequence don't need to be next to each other.
- They just need to appear in the same order.



Example:

- If you have two strings "ABCBDAB" and "BDCAB," the LCS is "BCAB" because "BCAB" is the longest sequence that appears in both strings in the same order.

Examples of Dynamic Programming …(cont'd)

Why is LCS is important?

- LCS is important because it helps us compare sequences, which is useful in many areas of computer science:
- **Text Comparison:** The longest common subsequence algorithm is used in tools that compare documents or files to find differences or similarities.
  - **Example,** it can help detect plagiarism by comparing two pieces of text to see how much they match.
- **DNA Sequence Analysis:** In bioinformatics, the LCS algorithm is used to compare DNA sequences to find similarities, which can help scientists understand genetic relationships between different organisms.
- **Version Control:** LCS is used in software development to help manage changes to code. It helps in merging different versions of files by finding common sequences in them.

Examples of Dynamic Programming …(cont’d)

Example of Longest Common Subsequence (LCS)

- Let's find the LCS of the strings "ABCBDAB" and "BDCAB."

Step 1: Write Down the Strings

- We have two strings:
  - String 1: "ABCBDAB“ ---- 2 power of 7 possible subsequence is possible
  - String 2: "BDCAB"

Step 2: Identify Common Subsequences

- A subsequence is a sequence that appears in the same order in both strings but not necessarily consecutively.
- Find the longest such sequence that both strings share.

Examples of Dynamic Programming …(cont’d)

Step 3: Find the LCS

Find the LCS of the strings **"ABCBDAB**" and **"BDCAB”**

- Let's look for the common subsequences:
  - The character "B" appears in both strings.
  - "C" also appears in both strings after "B."
  - After "C," "A" appears in both strings.
- Finally, "B" appears again in both strings after "A."
- So, one possible common subsequence is "BCAB".

Step 4: Verify the Length

- The length of "BCAB" is 4, which is the longest sequence that both strings have in common.

Step 5: Conclusion

- So, the LCS of "ABCBDAB" and "BDCAB" is "BCAB".

Examples of Dynamic Programming …(cont’d)

LCS using dynamic programming.

- For example take two strings “LONGEST” and “STONE”.
- First construct LCS dynamic table using algorithm specified above

			L	O	N	G	E	S	T
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
S	1	0	0	0	0	0	0	1	1
T	2	0	0	0	0	0	0	1	2
O	3	0	0	1	1	1	1	1	2
N	4	0	0	1	2	2	2	2	2
E	5	0	0	1	2	2	3	3	3
				O	N		E		

Algorithm :

```
if(i == 0 || j == 0)
      LCS[i][j]=0;
else if(x[i] == y[j])
      LCS[i][j] = 1 + LCS[i-1][j-1];
else
      LCS[i][j] = max(LCS[i-1][j] , LCS[i][j-1]);
```

- **So, the LCS of ‘’LONGEST” and “STONE”. is “ONE".**

[5]. Medium, Longest Common Subsequence Using Dynamic Programming, https://medium.com/kevinmavani/longest-common-subsequence-using-dynamic-programming

Applications of Longest Common Subsequence

- The basis of data comparison programs such as the diff-utility, and has applications in bioinformatics.
- It is also widely used by revision control systems, such as SVN and Git, for reconciling multiple changes made to a revision-controlled collection of files.

Time and Space Complexity

- *Time Complexity: $O(2^N)$,* i.e., exponential as we generate and compare all the subsequences of both the strings.

Greedy Algorithms vs Dynamic Programming

Greedy Algorithm

- Both greedy and dynamic programming are tools for optimization.
- However, greedy algorithms look for locally optimum solutions or in other words, a greedy choice, in the hopes of finding a global optimum.
- Hence greedy algorithms can make a guess that looks optimum at the time but becomes costly down the line and do not guarantee a globally optimum.

Dynamic programming,

- On the other hand, dynamic programming finds the optimal solution to subproblems and then makes an informed choice to combine the results of those subproblems to find the most optimum solution.

[6]. Programiz, Dynamic Programming. https://www.programiz.com/dsa/dynamic-programming

Applications of Dynamic Programming

- **Optimization Problems:** Used to solve various optimization problems like the 0/1 Knapsack, Longest Increasing Subsequence (LIS), and Maximum Subarray problems.
- **String Matching**: Helps in problems like Edit Distance, Longest Common Subsequence(LCS), and String Segmentation.
- **Graph Algorithms:** Applied in finding shortest paths (e.g., Bellman-Ford, Floyd-Warshall), and in network flow problems.
- **Resource Allocation:** Used in scenarios where resources need to be allocated optimally, such as in production planning and scheduling.

Applications of Dynamic Programming …(cont’d)

- **Bioinformatics:** Employed in DNA sequence alignment, protein structure prediction, and evolutionary tree construction.
- **Game Theory:** Helps in solving combinatorial game problems where the goal is to find optimal strategies.
- **Financial Modeling:** Used for risk management, portfolio optimization, and options pricing in finance.
- **Machine Learning:** Dynamic programming algorithms like Viterbi are used in hidden Markov models and other probabilistic models.

Applications of Dynamic Programming …(cont'd)

- **Control Systems:** Applied in dynamic optimization problems like the optimal control of systems over time.
- **Computer Vision:** Used in image processing tasks such as seam carving for content-aware image resizing.
- **Operations Research:** Helps in solving linear programming problems and other resource management challenges.
- **Artificial Intelligence:** Applied in pathfinding algorithms and decision-making processes in AI systems, such as in reinforcement learning

[3]. WsCube Tech, Dynamic Programming (With Problems & Key Concepts)https://www.wscubetech.com/resources/dsa/dynamic-programming

Advantages and Disadvantages of Dynamic Programming

Advantages

- **Efficiency:** Reduces the time complexity by avoiding redundant calculations through the reuse of already computed solutions.
- **Optimal Solutions:** Ensures that the solutions to problems are optimal by solving and combining the results of subproblems.
- **Versatility:** Applicable to a wide range of problems, from combinatorial optimization to sequence alignment in bioinformatics.
- **Handles Complex Problems:** Can solve problems that are difficult or impossible to solve using other techniques, especially those with overlapping subproblems.
- **Improves Recursive Algorithms:** Converts exponential-time recursive algorithms into polynomial-time solutions.

Advantages and Disadvantages of Dynamic Programming

Disadvantages

- **High Space Complexity:**
  - Can require significant memory to store solutions for all subproblems, especially in multi-dimensional DP problems.
- **Complexity in Formulation:**
  - Identifying subproblems, defining recurrence relations, and setting up base cases can be challenging and may require deep insight into the problem.
- **Not Always Necessary:**
  - Dynamic Programming might be overkill for simpler problems where a greedy approach or divide-and-conquer technique would suffice.

[3]. WsCube Tech, Dynamic Programming, https://www.wscubetech.com/resources/dsa/dynamic-programming

Summary

- Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems. And the results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems.
- In dynamic programming the problem should be able to be divided into smaller overlapping sub-problem. Final optimum solution can be achieved by using an optimum solution of smaller sub-problems.
- Both greedy and dynamic programming are tools for optimization. However, greedy algorithms look for locally optimum solutions or in other words, a greedy choice, in the hopes of finding a global optimum.
- Dynamic programming has the following advantages: - efficiency, optimal solutions, versatility, handles complex problems, improves recursive algorithms

References

1. TutorialsPoint, Dynamic Programming concepts. https://www.tutorialspoint.com/data_structures_algorithms/dynamic_programming.htm

2. W3Schools, DSA Dynamic Programming, https://www.w3schools.com/dsa/dsa_ref_dynamic_programming.php

3. WsCube Tech, Dynamic Programming. https://www.wscubetech.com/resources/dsa/dynamic-programming

4. Techprodezza,0/1 Knapsack Problem, Tharun Buddigina, October 27, 2020, https://www.programiz.com/dsa/dynamic-programming

5. Medium, Longest Common Subsequence Using Dynamic Programming. https://medium.com/kevinmavani/longest-common-subsequence-using-dynamic-programming

6. Programiz, Dynamic Programming. https://www.programiz.com/dsa/dynamic-programming

Thank You!

For your attention