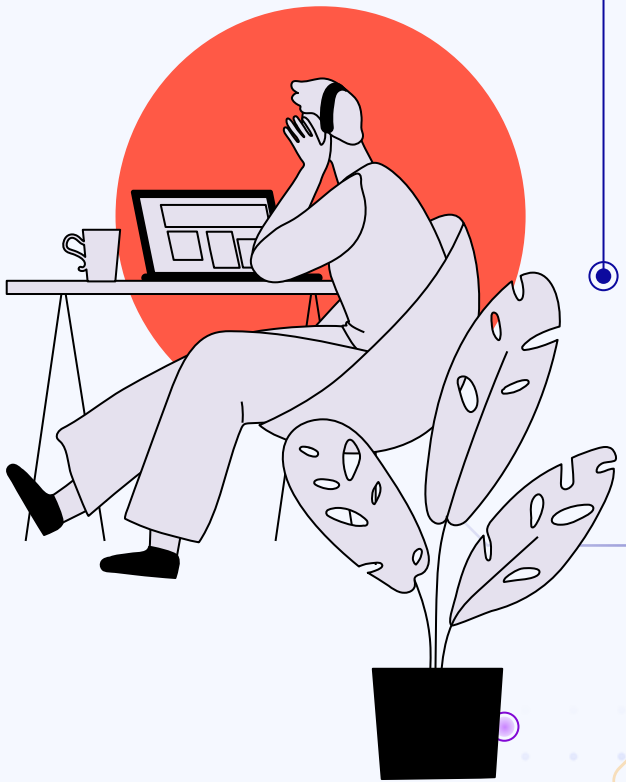
DYNAMIC PROGRAMMING



Nhóm 3
Đặng Hữu Phát
Lê Trọng Đại Trường

Table of contents

01

Basic concepts

1.1 What is DP

1.2 Main properties

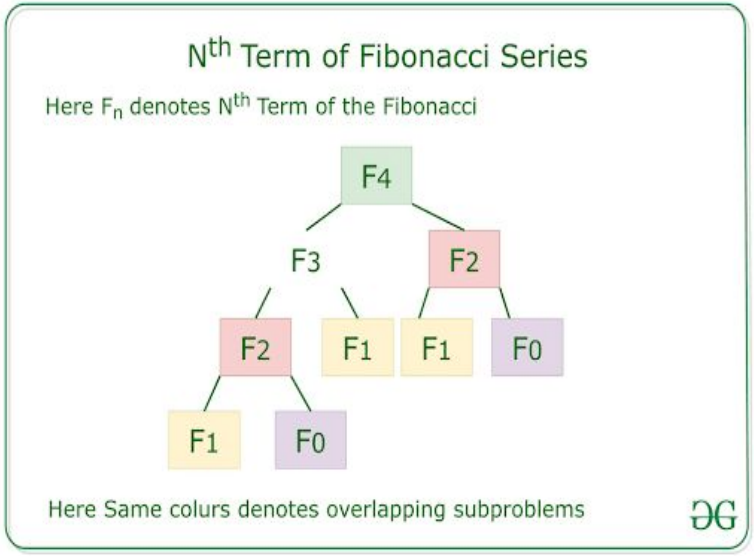
1.3 Approaching techniques

1.1 What is DP?

Example: Fibonacci Numbers

Problem:

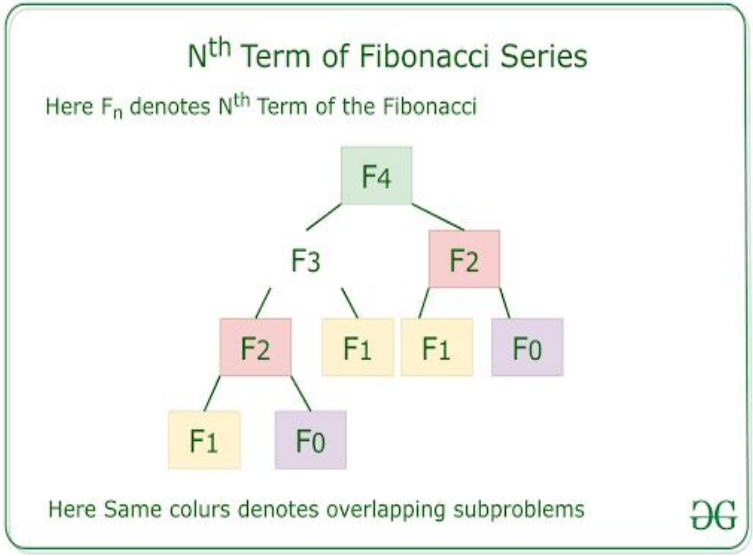
Solution:



1.1 What is DP?

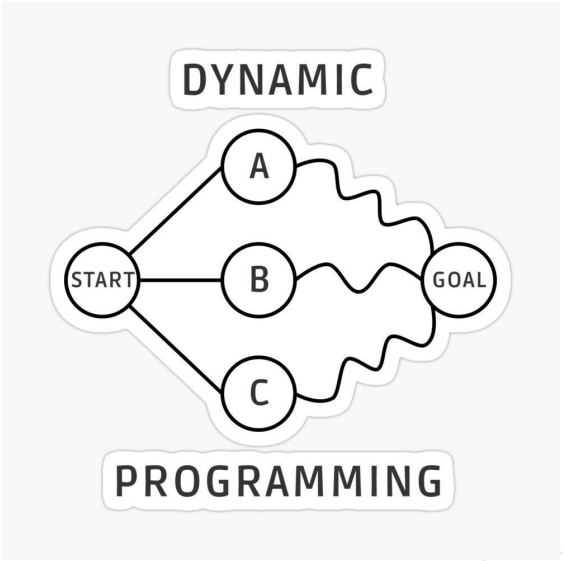
Example: Fibonacci Numbers
Problem: Recursive solution that has repeated calls for same inputs.
Solution: Store the results of subproblems.

➡ Dynamic Programming



1.1 What is DP?

Dynamic programming is a method of solving problems **by breaking them down into smaller sub-problems** and **reusing the solutions** to these sub-problems to build up to the solution for the original problem.



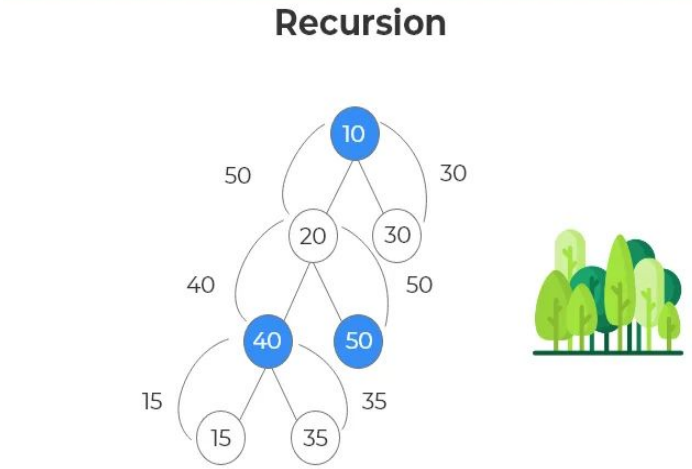
1.2 Main properties

Optimal Substructure

Overlapping Subproblem

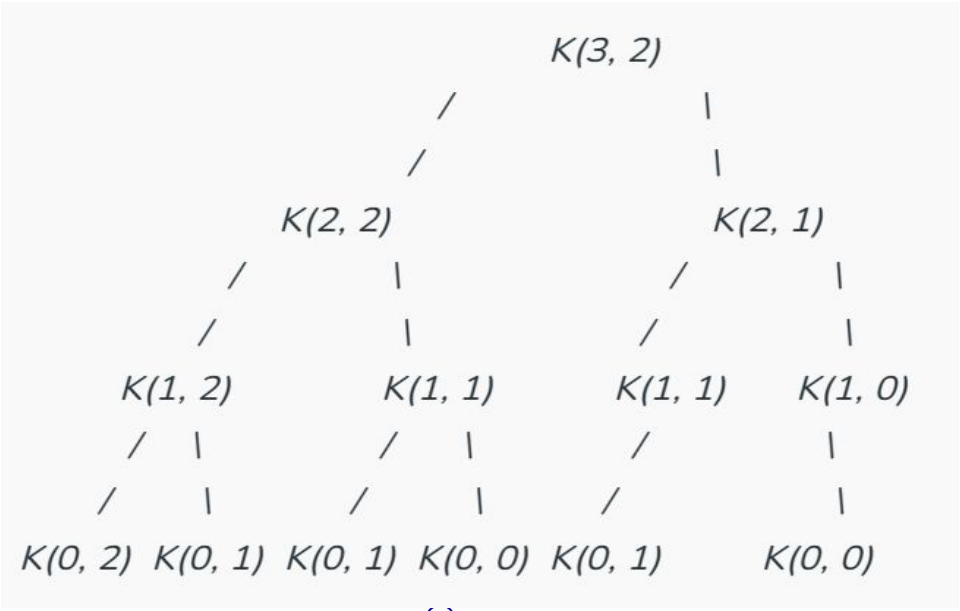
Optimal Substructure

```
int maxVal(TreeNode root) {
    if (root == null)
        return -1;
    int left = maxVal(root.left);
    int right = maxVal(root.right);
    return max(root.val, left, right);
}
```



Overlapping Subproblems

0-1 Knapsack problem



Optimal Substructure

The optimal solution to the entire problem can be built by combining optimal solutions of its subproblems

Overlapping Subproblem

The same subproblems are solved multiple times during the computation.

1.3 Approaching techniques

Memoization

Tabulation

1.3 Approaching techniques

Memoization

- Cache the results of function calls
- Return the cached result if the function is called again with the same inputs.

Tabulation

- Store the results of the subproblems in a table
- Use these results to solve larger subproblems until we solve the entire problem

1.3 Approaching techniques

	Tabulation	Memoization
State	State Transition relation is difficult to think	State transition relation is easy to think
Code	Code gets complicated when lot of conditions are required	Code is easy and less complicated
Speed	Fast, as we directly access previous states from the table	Slow due to lot of recursive calls and return statements
Subproblem solving	If all subproblems must be solved at least once, a bottom-up dynamic-programming algorithm usually outperforms a top-down memoized algorithm by a constant factor	If some subproblems in the subproblem space need not be solved at all, the memoized solution has the advantage of solving only those subproblems that are definitely required
Table Entries	In Tabulated version, starting from the first entry, all entries are filled one by one	Unlike the Tabulated version, all entries of the lookup table are not necessarily filled in Memoized version. The table is filled on demand.

Top-down

```
var m = map(0 → 0, 1 → 1)
function fib(n)
    if key n is not in map m
        m[n] = fib(n - 1) +
fib(n - 2)
    return m[n]
```

- Time Complexity: O(N)
  Computes each Fibonacci number only once and stores the result in an array

- Space Complexity: O(N)
  Lookup table has been created

Bottom-up

```
function fib(n)
    if n = 0
        return 0
    else
        var prevFib = 0, currFib = 1
        repeat n - 1 times
            var newFib = prevFib + currFib
            prevFib = currFib
            currFib  = newFib
    return currFib
```

- Time Complexity: O(N)
- Auxiliary Space: O(N)

02

Why ?

2.1 Advantages

2.2 Disadvantages

2.1 Advantages

Optimality

Dynamic programming can be used to find the optimal solution to problems with optimal substructure.

EFficiency

Dynamic programming can significantly improve the efficiency of algorithms for solving problems with overlapping subproblems.

Simplicity

Dynamic programming algorithms are often relatively simple to implement, even for complex problems.

2.2 Disadvantages

Identifying Subproblems:

Identifying optimal substructure and overlapping subproblems can be challenging.

High Memory Requirements

Dynamic programming often demands significant memory resources for storing subproblem solutions, potentially leading to impractical memory usage.

Dependency on Problem Characteristics

The effectiveness of DP relies on the presence of optimal substructure and overlapping subproblems.

03
DP vs Divide & Conquer vs Greedy

Example

DP

Greedy

Divide Conquer

Example

DP

Fibonacci series,
Longest common subsequence,
Knapsack problem,

Greedy

Huffman coding,
Kruskal's algorithm,
Dijkstra's algorithm.

Divide Conquer

Merge sort,
Quick sort,
Binary search, etc.

Approach and goal

DP

Greedy

Divide Conquer

Approach and goal

DP

Solves subproblems recursively and stores their solutions to avoid repeated calculations

Greedy

Makes locally optimal choices at each step with the hope of finding a global optimum

Divide Conquer

Breaks down a problem into smaller subproblems solves each subproblem recursively, and then combines the solutions

Time and space complexity

DP

Greedy

Divide Conquer

Time and space complexity

DP

More efficient but slower than greedy.

Greedy

Efficient and fast than divide and conquer.

Divide Conquer

Less efficient and slower.

Optimal solution

DP

Guarantees the optimal solution.

Greedy

May or may not provide the optimal solution

Divide Conquer

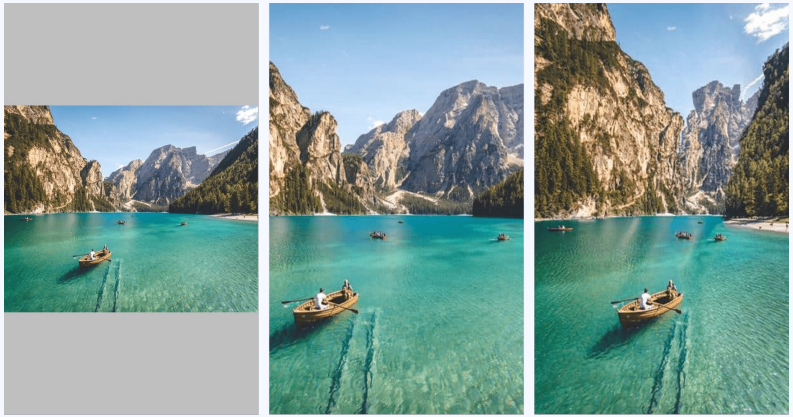
It is used to obtain a solution to the given problem, it does not aim for the optimal solution

04

Application

Application

Seam Carving



(a) Scaled Image (b) Cropped Image (c) Seam Carved Image

- Resize an image without distorting its important features, such as faces, objects, or text.
- Using dynamic programming to compute the cumulative energy of each pixel in the image, based on its gradient and its neighbors.

Application

Cryptography

- Involving techniques such as encryption, decryption, digital signatures, authentication, etc.
- Using dynamic programming algorithms to encrypt and decrypt messages.

Application

Google Maps

- Providing the most efficient and realistic route recommendations for each mode of transportation.
- Using dynamic programming to find the shortest path between a starting point and a destination.

Question 1

Which of the following standard algorithms is not Dynamic Programming based?

Choices

A Bellman–Ford Algorithm for single source shortest path

B Floyd Warshall Algorithm for all pairs shortest paths

C 0-1 Knapsack problem

D Prim's Minimum Spanning Tree

Question 1

Which of the following standard algorithms is not Dynamic Programming based?

Choices

A Bellman–Ford Algorithm for single source shortest path

B Floyd Warshall Algorithm for all pairs shortest paths

C 0-1 Knapsack problem

D Prim's Minimum Spanning Tree

Question 2

We use dynamic programming approach when

Choices

A We need an optimal solution

B The solution has optimal substructure

C The given problem can be reduced to the 3-SAT problem

D It's faster than Greedy

Question 2

We use dynamic programming approach when

Choices

A We need an optimal solution

B The solution has optimal substructure

C The given problem can be reduced to the 3-SAT problem

D It's faster than Greedy

Question 3

Which of the following is NOT a characteristic of dynamic programming?

Choices

A Memoization, which involves storing the results of expensive function calls and reusing them.

B Breaking a problem into smaller overlapping subproblems.

C Solving problems in a sequential manner.

D Dynamic programming can be used for problems where the solution has an optimal substructure.

Question 3

Which of the following is NOT a characteristic of dynamic programming?

Choices

A Memoization, which involves storing the results of expensive function calls and reusing them.

B Breaking a problem into smaller overlapping subproblems.

C Solving problems in a sequential manner.

D Dynamic programming can be used for problems where the solution has an optimal substructure.

Thanks!

Do you have any questions?

Coin counting

Problem:

Let's assume in our currency system; we have coins worth 1, 7, 10. Our objective is to count a specific value, 15, using the least possible coins

Optimal solution:

An optimal solution will pick 2 coins worth 7 and 1 coin worth 1

Greedy:

a greedy algorithm will first pick the coin with maximum value from the available coins. So it'll pick 1 coin worth 10 and 5 coins worth 1.

https://www.baeldung.com/cs/greedy-vs-heuristic-algorithm