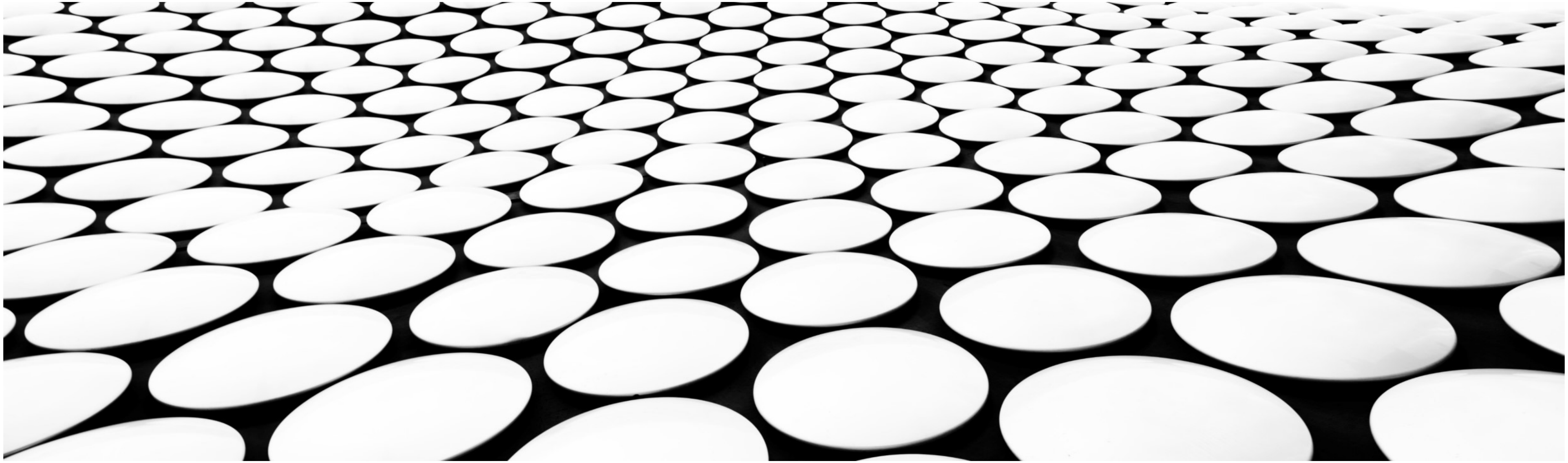
DYNAMIC PROGRAMMING



DEFINITION

- An optimization technique used to solve problems by breaking them down into simpler, overlapping subproblems and solving each subproblem only once, storing its solution to avoid redundant computations.

- If the same subproblem arises, its solution can be looked up rather than recomputed.

THE BASIC STEPS

- Understand the problem and determine the structure of an optimal solution. Identify the key subproblems that need to be solved.
- Express the value of an optimal solution in terms of the values of smaller subproblems. This is typically done through a recurrence relation.
- Solve the subproblems in a specific order, either starting from the smallest subproblems and working towards the larger problem (bottom-up approach) or starting with the original problem and recursively solving smaller subproblems (top-down approach).
- Once the values of the subproblems are computed, construct an optimal solution by combining these solutions.

APPLICATION OF DYNAMIC PROGRAMMING

- Dynamic programming can be applied to a wide range of problems, both in discrete and continuous optimization.
- It is often used to solve optimization problems where the goal is to find the best solution among a set of feasible solutions.
- Some classic examples of problems that can be solved using dynamic programming include:
  - Fibonacci Sequence: Computing Fibonacci numbers efficiently.
  - Shortest Path Problems: Finding the shortest path between two points in a graph (e.g., Dijkstra's algorithm, Bellman-Ford algorithm).
  - Longest Common Subsequence: Finding the longest subsequence common to two sequences.
  - Knapsack Problem: Maximizing the value of items in a knapsack without exceeding its capacity.
  - Matrix Chain Multiplication: Finding the most efficient way to multiply a chain of matrices.
  - Edit Distance: Measuring the similarity between two strings by counting the minimum number of operations (insertions, deletions, substitutions) required to transform one into the other.

DYNAMIC PROGRAMMING CATEGORIES

- Top-Down (Memoization, not "Memorization) involves solving the problem recursively and storing the solutions to subproblems in a table to avoid redundant computations.

MEMOIZATION

- Create a data structure (e.g., a dictionary or an array) to serve as a memoization table.
- Write a recursive function to solve the problem, but with the added step of checking the memoization table before solving a subproblem. If the solution is found in the table, return it; otherwise, compute and store the result.
- Apply the top-down approach, breaking down the original problem into subproblems and solving them recursively.
- Memoization ensures that the same subproblem is not solved multiple times, avoiding redundant computations.

EXAMPLE

```
function result = fibonacci(n, memo)
    if nargin < 2 %first call with one argument e.g. r=fibonacci(7)
        memo = containers.Map;  % Create a memoization table
(dictionary)
    end
    if isKey(memo, n)
        result = memo(n);
        return;
    elseif n <= 2
        result = 1;
    else
        result = fibonacci(n-1, memo) + fibonacci(n-2, memo);
    end
    memo(n) = result;  % Store the result in the memoization table
end
```

DYNAMIC PROGRAMMING CATEGORIES

- Bottom-Up (Tabulation) involves solving the problem iteratively, starting from the smallest subproblems and building up to the original problem.

TABULATION

- Create a table (usually an array or matrix) to store solutions to subproblems. Initialize entries that represent base cases.
- Iterate over the table in a bottom-up manner, starting from the base cases and progressing towards the solution of the original problem.
- For each entry in the table, compute its value based on the solutions to smaller subproblems. Use the optimal substructure property to iteratively build up the solution.
- The final result of the original problem is typically found in one or more entries of the table, depending on the nature of the problem.
- If the problem requires constructing an optimal solution, you can use the information stored in the table to trace back and reconstruct the solution.

EXAMPLE

```
function result = fibonacciTabulation(n)
    if n == 1
        result = 1;
        return;
    else
        table = zeros(1, n);
        table(1) = 1;  % Base case for Fibonacci(1)
        for i = 2:n
            table(i) = table(i-1) + table(max(i-2, 1));
        end
        result = table(n);
    end
end
```

SIGNS OF DYNAMIC PROGRAMMING SUITABILITY

Optimal Substructure

Sign: The problem can be broken down into smaller, independent subproblems.

Explanation: The optimal solution to the overall problem can be constructed from optimal solutions to its subproblems. In other words, the problem exhibits optimal substructure.

SIGNS OF DYNAMIC PROGRAMMING SUITABILITY

Overlapping Subproblems

Sign: The problem can be decomposed into subproblems that are reused or solved multiple times.

Explanation: Solutions to the subproblems are cached or memoized, and these stored solutions are reused to avoid redundant computations.

SIGNS OF DYNAMIC PROGRAMMING SUITABILITY

Memoization or Tabulation Benefit

Sign: There is potential for memoization (top-down) or tabulation (bottom-up) to improve efficiency.

Explanation: Memoization involves storing the results of expensive function calls and returning the cached result when the same inputs occur again. Tabulation involves building a table and filling it in a way that avoids redundant computations.

SIGNS OF DYNAMIC PROGRAMMING SUITABILITY

Recursive Nature

Sign: A recursive solution to the problem involves solving the same problem with different inputs.

Explanation: Dynamic programming often involves recursive decomposition of the problem into subproblems, leading to a natural formulation for memoization.

SIGNS OF DYNAMIC PROGRAMMING SUITABILITY

Sequential Dependence

Sign: The solution to a subproblem depends only on the solutions of some of its preceding subproblems.

Explanation: The problem has a sequential or overlapping structure where each subproblem relies on the solutions of previous subproblems.

SIGNS OF DYNAMIC PROGRAMMING SUITABILITY

Reducible to Smaller Instances

Sign: The problem can be divided into smaller instances of the same problem.

Explanation: Dynamic programming is effective when a large problem can be reduced into smaller instances of the same problem. The solutions to these smaller instances are then combined to solve the larger problem.

SIGNS OF DYNAMIC PROGRAMMING SUITABILITY

Combinatorial Optimization

Sign: The problem involves finding an optimal solution from a set of feasible solutions.

Explanation: Dynamic programming is commonly used in combinatorial optimization problems, where the goal is to find the best solution from a finite set of possibilities.

SIGNS OF DYNAMIC PROGRAMMING SUITABILITY

Not Too Many Subproblems

Sign: The number of distinct subproblems is reasonable given the available resources.

Explanation: While dynamic programming can provide optimal solutions, the number of distinct subproblems should not be excessively large, as this could lead to an impractical number of computations.

SIGNS OF DYNAMIC PROGRAMMING SUITABILITY

Polynomial Time Complexity

Sign: Dynamic programming solutions often have polynomial time complexity.

Explanation: For a problem to be suitable for dynamic programming, the time complexity of solving the problem and its subproblems should not grow too rapidly.