Unit -3 Dynamic Programming

Dynamic programming is a computer programming technique where an algorithmic problem is first broken down into sub-problems, the results are saved, and then the sub-problems are optimized to find the overall solution — which usually has to do with finding the maximum and minimum range of the algorithmic query.

Dynamic Programming is also used in optimization problems. Like divide-and-conquer method, Dynamic Programming solves problems by combining the solutions of sub problems. Moreover, Dynamic Programming algorithm solves each sub-problem just once and then saves its answer in a table, thereby avoiding the work of re-computing the answer every time.

Richard Bellman was the one who came up with the idea for dynamic programming in the 1950s. It is a method of mathematical optimization as well as a methodology for computer programming. It applies to issues one can break down into either overlapping subproblems or optimum substructures.

Characteristics of Dynamic Programming

- The problem can be broken down into subproblems which are themselves smaller instances of the same type of problem.
- The same subproblems recur multiple times in the process of solving the problem.
- Dynamic programming algorithms store the solutions to these subproblems in a table (usually an array or a hash table) so that each subproblem is only solved once, thereby saving computation time.
- There are typically two approaches in dynamic programming which are bottom-up (iterative) and top-down (recursive with memoization).
- When correctly formulated, a dynamic programming solution will always yield an optimal solution to the problem.
- It is used in various problems, including, but not limited to, shortest path problems like Bellman-Ford, computing the nth Fibonacci number, the knapsack problem, and many others.

Dynamic programming is an efficient way to solve problems that might otherwise be very time-consuming or infeasible to solve through direct methods or naive recursion.

Two important Characteristics are as follows:

overlapping sub-problems and optimal substructure.

Overlapping Sub-Problems

Overlapping Sub-Problems Similar to Divide-and-Conquer approach, Dynamic Programming also combines solutions to sub-problems. It is mainly used where the solution of one sub-problem is needed repeatedly. The computed solutions are stored in a table, so that these don't have to be re-computed. Hence, this technique is needed where overlapping sub-problem exists. For example, Binary Search does not have overlapping sub-problem. Whereas recursive program of Fibonacci numbers have many overlapping sub-problems.

Optimal Sub-Structure

Optimal Sub-Structure A given problem has Optimal Substructure Property, if the optimal solution of the given problem can be obtained using optimal solutions of its sub-problems.

For example, the Shortest Path problem has the following optimal substructure property − If a node x lies in the shortest path from a source node u to destination node v, then the shortest path from u to v is the combination of the shortest path from u to x, and the shortest path from x to v. The standard All Pair Shortest Path algorithms like Floyd-Warshall and Bellman-Ford are typical examples of Dynamic Programming.

Steps of Dynamic Programming Approach

Dynamic Programming algorithm is designed using the following four steps –

1. Characterize the structure of an optimal solution.

2. Recursively define the value of an optimal solution.

3. Compute the value of an optimal solution, typically in a bottom-up fashion.

4. Construct an optimal solution from the computed information.

Difference between Dynamic programming and Divide and Conquer Technique

- Divide & Conquer algorithm partition the problem into disjoint sub problems solve the sub problems recursively and then combine their solution to solve the original problems.
- Divide & Conquer algorithm can be a Bottom-up approach and Top down approach.
- Dynamic Programming is used when the sub problems are not independent, e.g. when they share the same sub problems.
- In this case, divide and conquer may do more work than necessary, because it solves the same sub problem multiple times.
- Dynamic Programming solves each sub problems just once and stores the result in a table so that it can be repeatedly retrieved if needed again.
- Dynamic Programming is a Bottom-up approach- we solve all possible small problems and then combine to obtain solutions for bigger problems.
- Dynamic Programming is a paradigm of algorithm design in which an optimization problem is solved by a combination of achieving sub-problem solutions and appearing to the "principle of optimality".

Divide & Conquer Method	Dynamic Programming
It deals (involves) three steps at each level of recursion: **Divide** the problem into a number of sub problems. **Conquer** the sub problems by solving them recursively. **Combine** the solution to the sub problems into the solution for original sub problems.	It involves the sequence of four steps: ○ Characterize the structure of optimal solutions. ○ Recursively defines the values of optimal solutions. ○ Compute the value of optimal solutions in a Bottom-up minimum. ○ Construct an Optimal Solution from computed information.
For example: Merge Sort & Binary Search etc.	**For example:** Matrix Multiplication.

Greedy Algorithm/Method	**Dynamic Programming**
Makes the locally optimal choice at each step with the hope of finding a global optimum.	Breaks the problem down into smaller, simpler subproblems and solves each subproblem just once, storing the solution and is usually done bottom-up.
Does not guarantee a globally optimal solution for all problems. Often used for problems where local optimality leads to a global optimum.	Guarantees an optimal solution by considering all possible cases.
Typically, it does not address overlapping subproblems.	Explicitly solves and caches solutions to overlapping subproblems.
Avoids recomputing subproblems because it doesn't generally recognize subproblems.	Reuses subproblem solutions; thus, no need to recompute them.
Often used for optimization problems where a series of decisions leads to a solution.	Used for optimization problems that can be broken down into overlapping subproblems with the optimal substructure property.
Forward-looking: makes decisions based only on current information without regard to future consequences.	Backward-looking: decisions are made by considering future consequences due to optimal substructure.
Fractional Knapsack, Minimum Spanning Trees (Kruskal's, Prim's algorithms).	0/1 Knapsack, Shortest Path (Floyd-Warshall, Bellman-Ford algorithms), Fibonacci number series.
Does not require caching past decisions.	Uses memoization or tabulation to cache and retrieve results of subproblems.
Used when a problem has a “greedy choice property,” allowing for a local optimum to be chosen at each step.	Used when the problem can be divided into stages, with a decision at each stage affecting the outcome of the solution.
Often more efficient in terms of time complexity.	It can be less efficient than greedy if every possible solution is computed (though proper use of memoization mitigates this).

Difference Between Greedy and Dynamic Programming

Elements of Dynamic Programming

There are basically three elements that characterize a dynamic programming algorithm:-

1. Substructure: Decompose the given problem into smaller sub problems. Express the solution of the original problem in terms of the solution for smaller problems.

2. Table Structure: After solving the sub-problems, store the results to the sub-problems in a table. This is done because sub-problem solutions are reused many times, and we do not want to repeatedly solve the same problem over and over again.

3. Bottom-up Computation: Using table, combine the solution of smaller sub-problems to solve larger sub-problems and eventually arrives at a solution to complete problem.

Components of Dynamic programming

1. Stages: The problem can be divided into several sub problems, which are called stages. A stage is a small portion of a given problem. For example, in the shortest path problem, they were defined by the structure of the graph.

2. States: Each stage has several states associated with it. The states for the shortest path problem were the node reached.

3. Decision: At each stage, there can be multiple choices out of which one of the best decisions should be taken. The decision taken at every stage should be optimal; this is called a stage decision.

4. Optimal policy: It is a rule which determines the decision at each stage; a policy is called an optimal policy if it is globally optimal. This is known as Bellman principle of optimality.

5. Given the current state, the optimal choices for each of the remaining states do not depend on the previous states or decisions. In

the shortest path problem, it was not necessary to know how we got a node only that we did.

6. There exists a recursive relationship that identifies the optimal decisions for stage j, given that stage j+1, has already been solved

7. The final stage must be solved by itself.

Applications of dynamic programming

1. 0/1 knapsack problem
2. All pair Shortest path problem
3. Reliability design problem
4. Longest common subsequence (LCS)
5. Flight control and robotics control
6. Time-sharing: It schedules the job to maximize CPU usage

0/1 Knapsack Problem:

Dynamic Programming Approach:

Knapsack Problem: Knapsack is basically means bag.

A bag of given capacity.

- knapsack is like a container or a bag. Suppose we have given some items which have some weights or profits.
- We have to put some items in the knapsack in such a way total value produces a maximum profit..

1. W ≤ capacity

2. Value ← Max

There are two types of knapsack problems:

- 0/1 knapsack problem
- Fractional knapsack problem

0/1 knapsack problem

The 0/1 knapsack problem means that the items are either completely or no items are filled in a knapsack.

For example, we have two items having weights 5kg and 3kg, respectively and the capacity of the bag is 6kg ,If we pick the 5kg item then we cannot pick 1kg item from the 3kg item (item is not divisible); we have to pick the 2kg item completely. This is a 0/1 knapsack problem in which either we pick the item completely or we will pick that item. The 0/1 knapsack problem is solved by the dynamic programming.

The fractional knapsack problem

- The fractional knapsack problem means that we can divide the item.
- The fractional knapsack problem is solved by the Greedy approach.

Example of 0/1 knapsack problem.

Consider the problem having weights and profits are:

Weights: {3, 4, 6, 5}

Profits: {2, 3, 1, 4}

The weight of the knapsack is 8 kg

The number of items is 4

The above problem can be solved by using the following method:

x_i = {1, 0, 0, 1}

= {0, 0, 0, 1}

= {0, 1, 0, 1}

Another approach to solve the problem is dynamic programming approach. In dynamic programming approach, the complicated problem is divided into sub-problems, then we find the solution of a sub-problem and the solution of the sub-problem will be used to find the solution of a complex problem.

The above are the possible combinations. 1 denotes that the item is completely picked and 0 means that no item is picked. Since there are 4 items so possible combinations will be:

2^4 = 16; So. There are 16 possible combinations that can be made by using the above problem. Once all the combinations are made, we have to select the combination that provides the maximum profit.

Example Let us consider that the capacity of the knapsack is W = 25 and the items are as shown in the following table.

Item :	A	B	C	D
Profit :	24	18	18	10
Weight :	24	10	10	7

Without considering the profit per unit weight (pi/wi),

if we apply dynamic approach to solve this problem, first item A will be selected as it will contribute maximum profit among all the elements. After selecting item A, no more item will be selected. Hence, for this given set of items total profit is 24. Whereas, the optimal solution can be achieved by selecting items, B and C, where the total profit is 18 + 18 = 36.

Matrix Chain Multiplication

It is a Method under Dynamic Programming in which the previous output is taken as input for next.

Here, Chain means one matrix's column is equal to the second matrix's row [always].

NOTE: For dynamic programming approaches i.e. Top Down and Bottom –up , Refer codetantra ,

Practice numericals done in class.