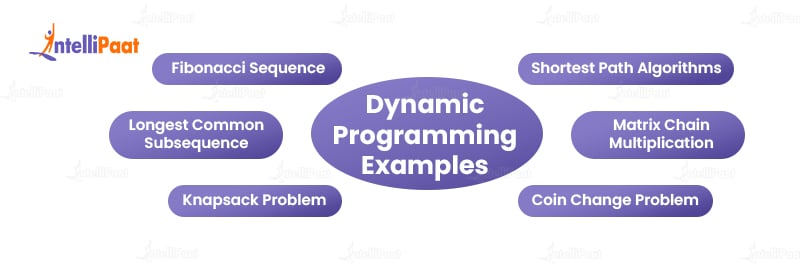
**Project Development Phase**

### Utilization Of Algorithms, Dynamic Programming, Optimal Memory Utilization

|  |  |
| --- | --- |
| **NM ID** | **8073B6FD2C609D2A026419D64F1FD32** |
| **PROJECT TITTLE** | **BUILD IN EVENT MANAGEMENT USING SALESFORCE** |

## Dynamic Programming Examples

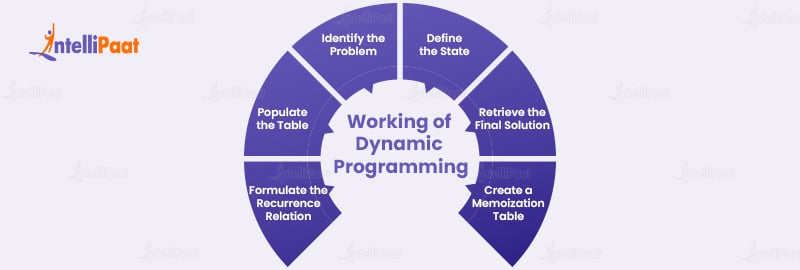


Dynamic programming is a very versatile approach to solving problems. Below are a few examples of how you can utilize dynamic programming algorithms.

* [**Fibonacci Sequence**](https://intellipaat.com/blog/what-is-fibonacci-series-in-c/)**:**One classic example is calculating the nth Fibonacci number using dynamic programming. By storing previously computed values, dynamic programming algorithm can avoid redundant calculations, resulting in significant performance improvements.
* **Shortest Path Algorithms:** Dynamic programming is instrumental in solving shortest path problems, such as Dijkstra’s or Bellman-Ford’s algorithms. It finds the shortest path efficiently by incrementally building optimal paths from a source node to other nodes.
* **Longest Common Subsequence:**Given two sequences, dynamic programming can efficiently find the longest common subsequence (LCS) between them. It avoids redundant computations by breaking the problem into smaller subproblems and storing intermediate results.
* **Matrix Chain Multiplication:** Dynamic programming in Java is commonly used to optimize matrix chain multiplication. It can minimize the number of scalar multiplications required by finding the optimal parenthesization of matrix multiplications.
* **Knapsack Problem:** The problem involves selecting a combination of items with maximum value while considering a weight constraint. Dynamic programming can be used to find the optimal solution by breaking the problem into smaller subproblems and utilizing the previously computed results.
* **Coin Change Problem:** Given a set of coin denominations and a target value, dynamic programming can determine the minimum number of coins required to reach the target value. This problem is often solved using bottom-up dynamic programming, starting with smaller values and gradually building up to the target value.

**Check out our**[**C Language Tutorial**](https://intellipaat.com/blog/tutorial/c-tutorial/)**to master the basics with our absolute beginner’s guide.**

## Working of Dynamic Programming



Dynamic programming avoids redundant calculations by storing the solutions to subproblems in a data structure, such as an array or a table. It allows for efficient retrieval of previously computed solutions when needed.

The general steps involved in implementing dynamic programming are as follows:

* **Identify the Problem:**Determine the optimization problem that can be divided into overlapping subproblems. This problem should exhibit both optimal substructure and overlapping subproblem properties.
* **Define the State:**Identify the variables or parameters that define the state of the problem. The state should concisely capture the essential information required to solve the problem.
* **Formulate the Recurrence Relation:** Express the solution to a larger problem in terms of the solutions to its subproblems. This recurrence relation provides the mathematical relationship between the current state and its substrates.
* **Create a Memoization Table:** Initialize a data structure, such as an [**array**](https://intellipaat.com/blog/tutorial/java-tutorial/java-array-and-string/) or a table, to store the solutions to subproblems. This table serves as a cache for storing and retrieving previously computed solutions.
* **Populate the Table:** Iterate through the subproblems in a bottom-up manner, filling the table with solutions based on the recurrence relation. Start with the simplest subproblems and gradually build up to the larger ones.
* **Retrieve the Final Solution:**Once the table is populated, the final solution can be obtained by accessing the value stored in the table corresponding to the original problem’s state.

**Pseudo Code Example (Fibonacci sequence):**  
**function fibonacci(n):**  
**table = new Array(n+1)**  
**table[0] = 0**  
**table[1] = 1**  
**for i from 2 to n:**  
**table[i] = table[i-1] + table[i-2]**  
**return table[n]**

For example, this given illustration demonstrates the use of dynamic programming to compute the Fibonacci sequence. The process involves initializing a table with base cases (0 and 1) and subsequently populating it with solutions to subproblems (which is the sum of the previous two numbers). And finally, the solution is obtained by retrieving the value at the given nth index from the table.