Delloson Seatwork_Module 1 - Lab Exercise.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Rename notebook

+ Code + Text

Dynamic Programming

Dynamic Programming(DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to the subproblems.

Top-Down Approach with Memoization

Whenever we solve a subproblem, we cache its result so that we don't end up solving it repeatedly if it's called multiple times. This technique of storing the results of the previously solved subproblems is called Memoization.

In memoization, we solve the bigger problem by recursively finding the solution to the subproblems. It is a top-down approach.

```
#Implementation of Fibonacci Series using Memoization
""" The Fibonacci Series is a series of numbers in which each number
    is the sum of the preceding two numbers.
    By definition, the first two numbers are 0 and 1.

    Implement with the following steps:
    - Declare function with parameters: Number N and Dictionary Memo.
    - If n equals 1, return 0
    - If n equals 2, return 1
    - If current element is not in memo, add to memoy by recursive call for previous function and add. """

def fibMemo(n, memo= {}):
  if n == 1:
    return 0
  if n == 2:
    return 1
  if not n in memo:
    memo [n] = fibMemo(n-1, memo) + fibMemo(n-2, memo) + fibMemo(n-2, memo)
  return memo[n]

print("Fibonacci sequence using memoization: ", fibMemo(3))

#Implementation of Factorial of a number N using Memoization
def facMemo(n, memo= {}):
  if n == 0 or n ==1:
    return 1
  if n in memo:
    return memo[n]
  memo [n] = n * facMemo(n-1, memo)
  return memo[n]

print("Factorial using memoization:", facMemo(5))
```

```
Fibonacci sequence using memoization:  1
Factorial using memoization: 120
```

Delloson Seatwork_Module 1 - Lab Exercise.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Bottom-Up Approach with Tabulation

Tabulation is the opposite of the top-down approach and does not involve recursion. In this approach, we solve the problem "bottom-up". This means that the subproblems are solved first and are then combined to form the solution to the original problem.

This is achieved by filling up a table. Based on the results of the table, the solution to the original problem is computed.

[2]
```
#Implementation of Fibonacci Series using Tabulation
""" Fibonacci Series can be implemented using Tabulation using the following steps:
    - Declare the function and take the number whose Fibonacci Series is to be printed.
    - Initialize the list and input the values 0 and 1 in it.
    - Iterate over the range of 2 to n+1.
    - Append the list with the sum of the previous two values of the list.
    - Return the list as output. """

#Implementation of Factorial of a number N using Tabulation

def fibonacci_bottomup(n):
  table = [0,1]
  while n >= len(table):
    table.append(table[-1] + table[-2])
  return table

#factorial using tabulation

def factorial_bottomup(n):
  table = [1,1]
  for i in range(2, n+1):
    table.append(i * table [-1])
  return table[n]

print(fibonacci_bottomup(5))
print(factorial_bottomup(6))
```

```
[0, 1, 1, 2, 3, 5]
720
```

Delloson Seatwork_Module 1 - Lab Exercise.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[1]  #Student Enrollment Problem
     """ You are given a set of N courses.
         Courses will be defined as (code, units).
         Select the maximum set of courses a student can hve.
         Maximum units allowed is based on input with a max of 24
         default of 18. """
     def student_enrollment(programs, max_units=23):
         def helper(i, u):
             if i == 0 or u == 0:
                 return 0, []
             code, units = programs[i - 1]
             if units <= u:
                 with_current = helper(i - 1, u - units)
                 without_current = helper(i - 1, u)
                 if with_current[0] + 1 > without_current[0]:
                     return with_current[0] + 1, with_current[1] + [code]
                 else:
                     return without_current
             else:
                 return helper(i - 1, u)

         max_programs, selected_programs = helper(len(programs), max_units)
         return selected_programs

     programs = [('CPE011', 3), ('CPE012', 3), ('CPE013', 6), ('CPE311', 6)]
     max_units = 23

     print("Programs Enrolled:", student_enrollment(programs, max_units,),"\nCurrent Units: ", max_units)
```

```
Programs Enrolled: ['CPE011', 'CPE012', 'CPE013', 'CPE311']
Current Units:  23
```