CSE408-Lec#8

- Dynamic Programming
- Rod cutting Problem
- Coin Change problem
- Knapsack problem

Introduction

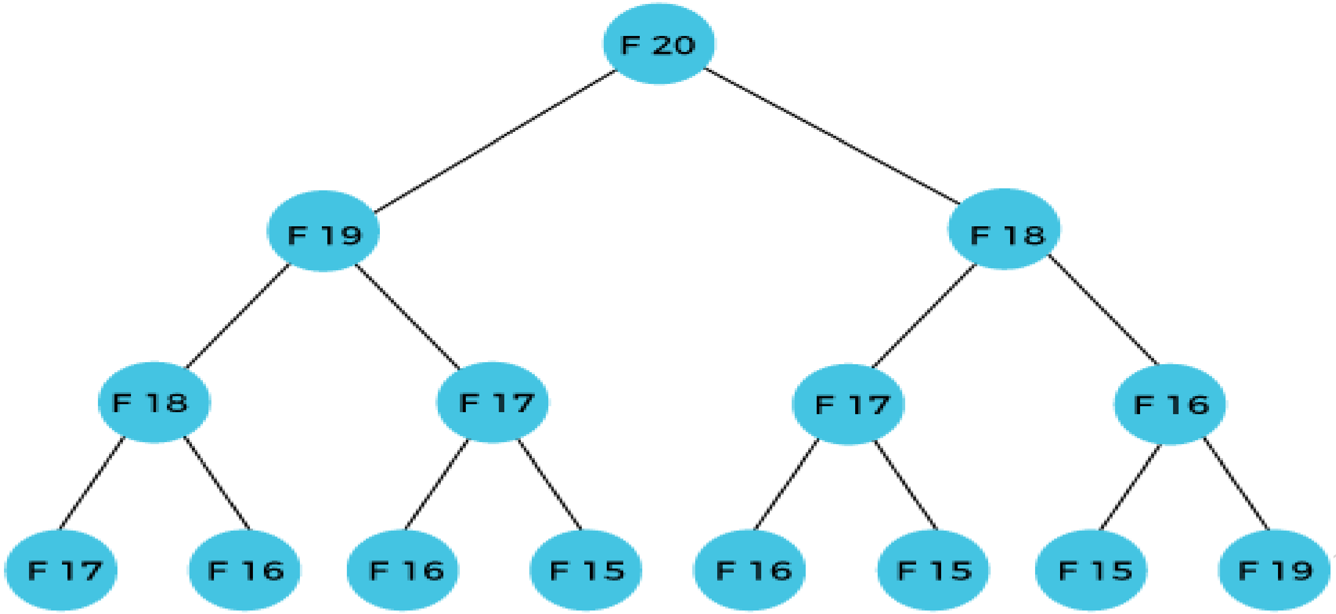
- Dynamic programming is a technique that breaks the problems into sub-problems, and saves the result for future purposes so that we do not need to compute the result again.
- The sub problems are optimized to optimize the overall solution is known as optimal substructure property.
- The main use of dynamic programming is to **solve optimization problems**.

Introduction

- **Let's understand this approach through an example.**
- Consider an example of the Fibonacci series. The following series is the Fibonacci series:
- **0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ,…**
  - Mathematically, we could write each of the terms using the below formula:
  - **F(n) = F(n-1) + F(n-2),**

Introduction

- How can we calculate F(20)?
- The F(20) term will be calculated using the nth formula of the Fibonacci series. The below figure shows that how F(20) is calculated.



Introduction

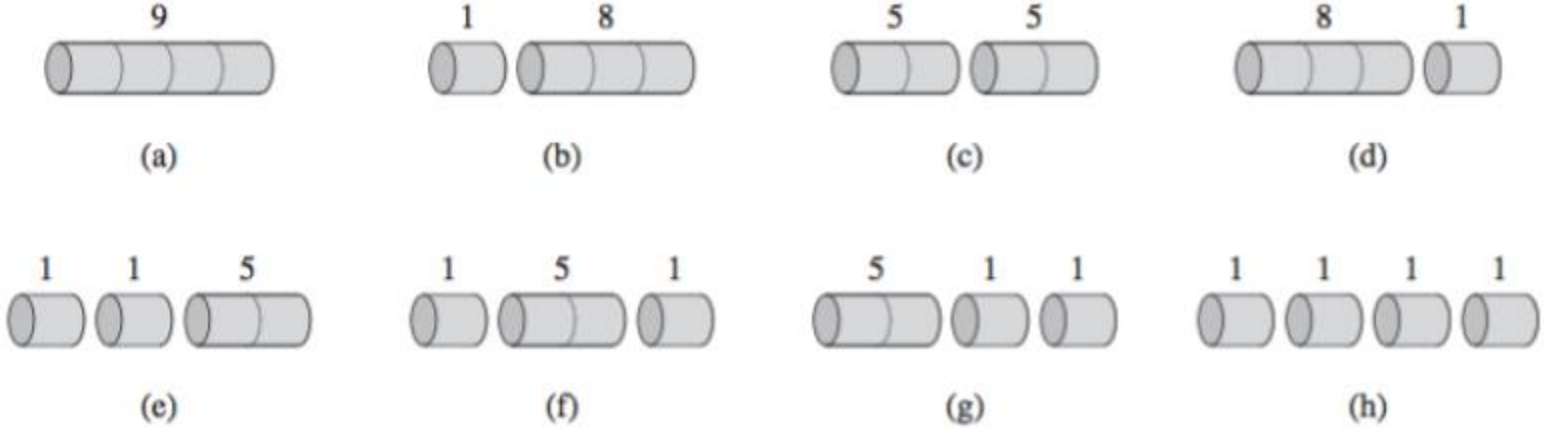
- How does the dynamic programming approach work?
- It breaks down the complex problem into simpler sub problems.
- It finds the optimal solution to these sub-problems.
- It stores the results of sub problems (memorization). The process of storing the results of sub problems is known as memorization.
- It reuses them so that same sub-problem is calculated more than once.
- Finally, calculate the result of the complex problem

Rod cutting

- Suppose you have a rod of length n, and you want to cut up the rod and sell the pieces in a way that maximizes the total amount of money you get. A piece of length i is worth pi dollars.

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- 8 possible ways of cutting the rod of length 4 are

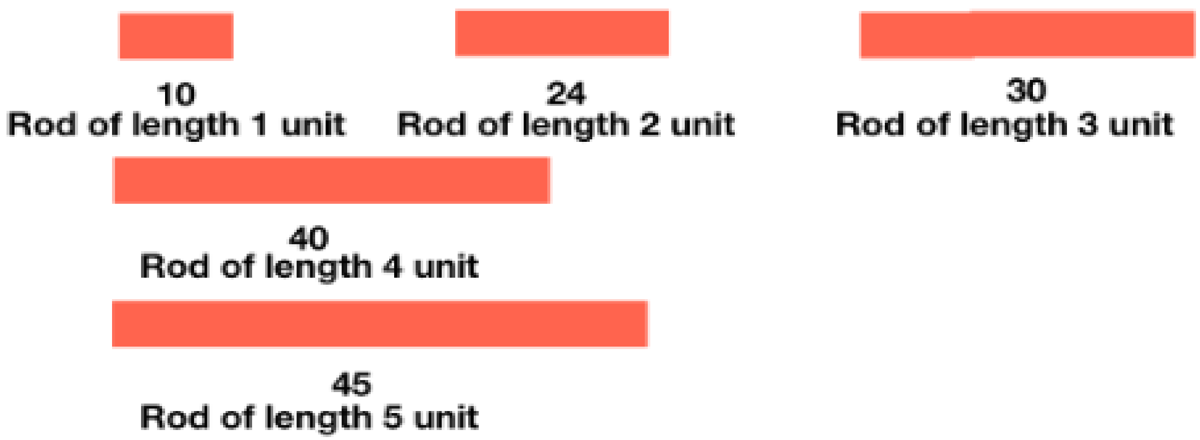


Rod cutting

- According to the problem, we are provided with a long rod of length n units.
- We can cut the rod in different sizes and each size has a different cost associated with it i.e., a rod of i units length will have a cost of ci$.

v/s length of

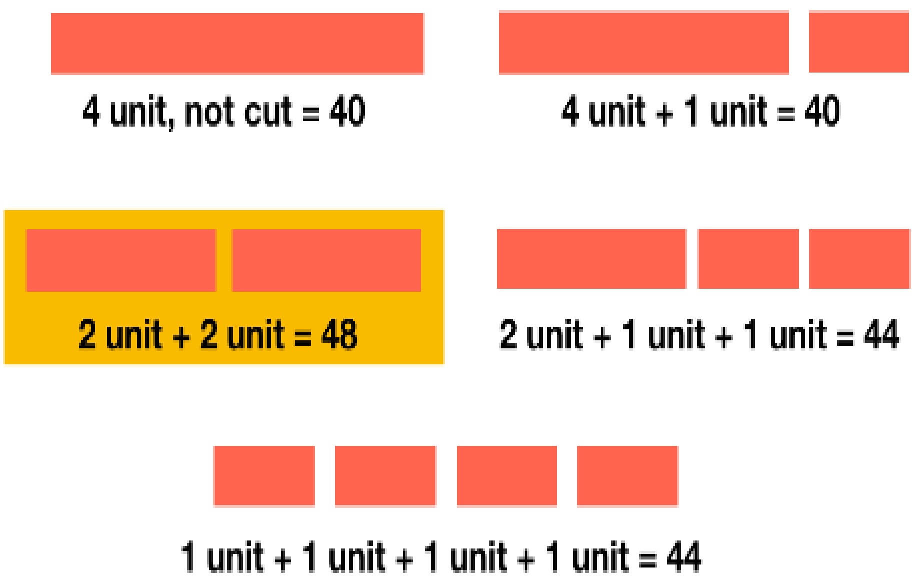
Length	1	2	3	4	5
Price	10	24	30	40	45



Rod Cutting

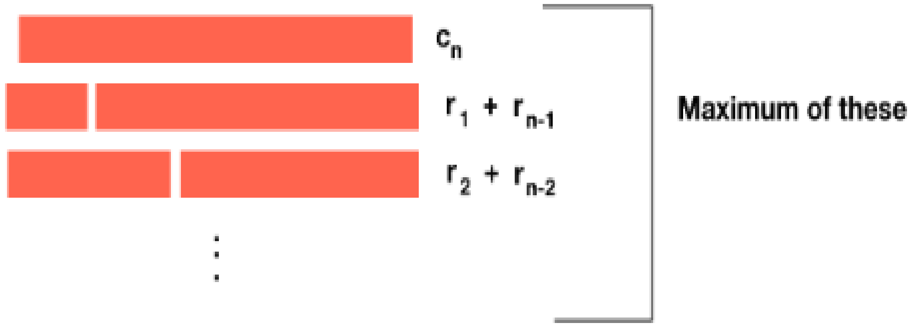
- Let's take a case when our rod is 4 units long, then we have the following different ways of cutting it.
- So, we have to choose between two option for a total of n-1 times and thus the total possible number of solutions are

$$\underbrace{2 * 2 * \ldots * 2}_{n-1 \text{ times}} = 2^{n-1}.$$



Rod Cutting

- For example, by selling the smaller pieces at the optimal price, we are generating maximum profit from the bigger piece. This property is called **optimal substructure**.
- We can say that when making a cut at i unit length, the maximum revenue can be generated by selling the first unit at ri$ and the second unit at rn−i$.
- The maximum revenue for a rod of length n (rn) will be the maximum of all these revenues.



$$r_n = max\{c_n, (r_1 + r_{n-1}), (r_2 + r_{n-2}), \ldots, (r_{n-1} + r_1)\}$$

Top Down Code for Rod Cutting

- **Example:**

length	1	2	3	4	5
weight	1	5	8	9	10

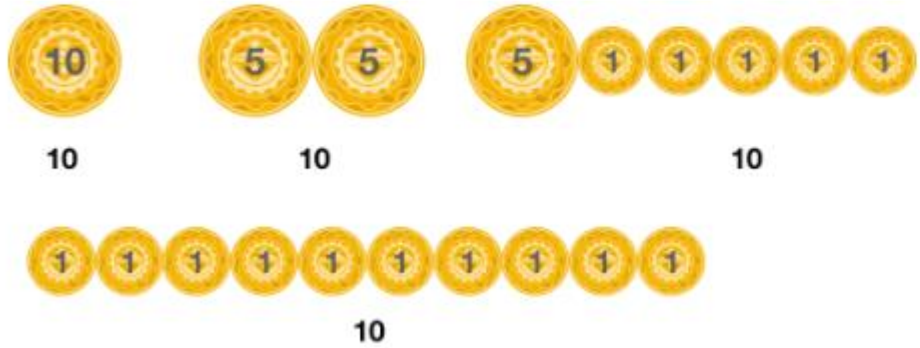
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1+1=2	1+2=3	1+3=4	1+4=5
2	0	1	5	1+5=6	6+4=10	10+1=11
3	0	1	5	8	8+1=9	8+5=13
4	0	1	5	8	9	9+1=10 13(max)
5	0	1	5	8	9	10+0=1

Top Down Code for Rod Cutting

- **Algorithm:**
- **For(I =1 to n)**
- **for(j=1 to n)**
- **if(i<j)**
- **rod_cut[i][j]=rod-cut[i-1][j]**
- **else**
- **rod_cut[i][j]=max(rod_cut[i-1][j],arr[i]+rod_cut[i][j-1])**

Coin Change Problem | Dynamic Programming

- In the coin change problem, we are basically provided with coins with different denominations like 1¢, 5¢ and 10¢.
- Now, we have to make an amount by using these coins such that a minimum number of coins are used.
- Let's take a case of making 10¢ using these coins, we can do it in the following ways:
- Using 1 coin of 10¢
- Using two coins of 5¢
- Using one coin of 5¢ and 5 coins of 1¢
- Using 10 c



Approach to Solve the Coin Change Problem

- Coin change problem also has the property of the optimal substructure i.e., the optimal solution of a problem incorporates the optimal solution to the subproblems.

```
def coinChange(coins, amount):
    dp = [0] * (amount + 1)
    dp[0] = 1

    for coin in coins:
        for j in range(coin, amount + 1):
            dp[j] += dp[j - coin]

    return dp[amount]

# Example usage:
coins = [1, 2, 3]
amount = 4
print("Number of ways to make change:", coinChange(coins, amount))
```

Approach to Solve the Coin Change Problem

- EXAMPLE 1:
- FIND THE TOTAL NUMBER OF DIFFERENT DENOMINATIONS (1,2,3,) CERTAIN AMONT W=5
- COINS={1,2,3}
- W={5}
- TOTAL NUMBER OF WAYS=
- (1,1,1,1,),
- (1,1,1,2),
- (1,2,2),
- (1,1,3),
- (2,3)
- EXAMPLE 2.FIDN THE NUMBER

Approach to Solve the Coin Change Problem

- EXAMPLE 2:
  DENOMINATIONS (2,3,5,10) CERTAIN AMONT W=15 .

SOLUTION:
1.IFCOIN_VALUE>W,THEN JUST COPY THE ABOVE CELLS
2.EXCLUDE THE COIN ,
3.INCLUDE THE COIN
4.ADD STEP 2 AND 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
3	1	0	1	0+1=1	1+0=1	0+1=1	1+1=2	0+1=1	1+1=2	0+2=2	1+1=2	0+2=2	1+2=3	0+2=2	1+2=3	0+3=3
5	1	1	1	1	1	1+1=2	2+1=3	1+1=2	3	3	4	5	5	5	6	7
10	1	1	1	1	1	2	3	2	3	3	4+	5	6	6	7	9

Approach to Solve the Coin Change Problem

ALGORITHM

```
1.a[i][0]=1
2 for(i=0,i<=coin.length,i++)
        for(j=0,j<=amount,j++)
                {
                    if(coins[i]>j)
                        a[i][j]=a[i-1][j];
                      else
                          a[i][j]=a[i-1] +a[][j-coins[i]];
                }
```

Knapsack Problem-

- You are given the following-
- A knapsack (kind of shoulder bag) with limited weight capacity.
- Few items each having some weight and value.
- **The problem states-**
- Which items should be placed into the knapsack such that-
- The value or profit obtained by putting the items into the knapsack is maximum.
- And the weight limit of the knapsack does not exceed.

Knapsack Problem-

- **Knapsack Problem Variants-**
- Knapsack problem has the following two variants-
- Fractional Knapsack Problem
- 0/1 Knapsack Problem

Knapsack Problem-

- **Fractional Knapsack Problem Using Greedy Method-**
  - For each item, compute its value / weight ratio.
    - Arrange all the items in decreasing order of their value / weight ratio.
- Start putting the items into the knapsack beginning from the item with the highest ratio.
- Put as many items as you can into the knapsack

Knapsack Problem-

- **Problem-**
  For the given set of items and knapsack capacity = 60 kg, find the optimal solution for the fractional knapsack problem making use of greedy approach
- **Step-01:**
- Compute the value / weight ratio for each item-

Item	Weight	Value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
3	15	45	3
4	22	77	3.5
5	25	90	3.6

Knapsack Problem-

- **Step-02:**
- Sort all the items in decreasing order of their value / weight ratio-

I1	I2	I5	I4	I3
(6)	(4)	(3.6)	(3.5)	(3)

- **Step-03:**
- Start filling the knapsack by putting the items into it one by one.

Knapsack Weight	Items in Knapsack	Cost
60	Ø	0
55	I1	30
45	I1, I2	70
20	I1, I2, I5	160

Knapsack Problem-

Now,

- Knapsack weight left to be filled is 20 kg but item-4 has a weight of 22 kg.
- Since in fractional knapsack problem, even the fraction of any item can be taken.
- So, knapsack will contain the following items-
- **< I1 , I2 , I5 , (20/22) I4 >**

- Total cost of the knapsack
- = 160 + (20/27) x 77
- = 160 + 70
- = 230 units

Q1

1. Which of the following is/are property/properties of a dynamic programming problem?

a) Optimal substructure

b) Overlapping sub problems

c) Greedy approach

d) Both optimal substructure and overlapping sub problems

Q2

2. If an optimal solution can be created for a problem by constructing optimal solutions for its subproblems, the problem possesses ______________ property.

a) Overlapping subproblems
b) Optimal substructure
c) Memoization
d) Greedy

Q3

3. If a problem can be broken into subproblems which are reused several times, the problem possesses _______________ property.

a) Overlapping subproblems
b) Optimal substructure
c) Memoization
d) Greedy

Q4

4. If a problem can be solved by combining optimal solutions to non-overlapping problems, the strategy is called _______________

a) Dynamic programming

b) Greedy

c) Divide and conquer

d) Recursion

Q5

5. When dynamic programming is applied to a problem, it takes far less time as compared to other methods that don’t take advantage of overlapping subproblems.
a) True
b) False

Q6

6. In dynamic programming, the technique of storing the previously calculated values is called

____

a) Saving value property
b) Storing value property
c) Memorization
d) Mapping

7. Which of the following problems is NOT solved using dynamic programming?

a) 0/1 knapsack problem

b) Matrix chain multiplication problem

c) Edit distance problem

d) Fractional knapsack problem

8. Which of the following techniques can be used to solve the Rod Cutting Problem efficiently?

a) Greedy algorithm

b) Depth-first search

c) Breadth-first search

d) Memoization

9. What is memorization in the context of dynamic programming?

A technique for storing the results of expensive function calls and reusing them later.

b) A technique for solving problems by breaking them down into smaller subproblems and solving each subproblem only once.

c) A technique for optimizing recursive algorithms by storing intermediate results in a table.

d) A technique for minimizing memory usage in algorithms by storing only essential information.

10. What is the time complexity of solving the Rod Cutting Problem using memoization?

a) O(n)

b) O(n log n)

c) O(n^2)
d) O(2^n)

11. Which data structure is commonly used in memoization for the Rod Cutting Problem?

a) An array or a hash table

b) A linked list

c) A stack

d) A queue

12. What is the space complexity of the memoized solution to the Rod Cutting Problem?

a) O(n)

b) O(1)

c) O(log n)

d) O(n^2)