Dynamic Programming Lecture 1

Sunday, 25 August 2024 2:04 PM

What is Dynamic Programming?

> DP is an algoritmic technique to solve optimization problems where the solution can be constructed from solutions of subproblems

> The key idea is to avoid recomputing solutions to subproblems that have already been solved.

> Commonly used in the problems that exhibit Overlapping subproblems and optimal substructure

> Real-life applications of DP include shortest paths, knapsack, sequence alignment in bioinformatics.

eg. Find the n^{th} Fibonacci number.

$F_0 = 0, \quad F_1 = 1 \quad F_i = \underline{F_{i-1}} + \underline{F_{i-2}}$

$$F_i = \begin{cases} F_{i-1} + F_{i-2}, & i > 1 \\ 1, & i = 1 \\ 0, & i = 0 \end{cases}$$

Overlapping Subproblems :- Problems where smaller problems can be solved multiple times

eg.
$$\begin{tikzcd}[row sep=small]
 & F_{i-1} \arrow[r] \arrow[dr] & F_{i-2} \;\ldots \\
F_i \arrow[ur] \arrow[dr] & & F_{i-3} \;\ldots \\
 & F_{i-2} \arrow[r] \arrow[dr] & F_{i-3} \;\ldots \\
 & & F_{i-4} \;\ldots
\end{tikzcd}$$

Optimal substructure:- An optimal solution to the original problem can be constructed using the optimal solution of the subproblems.

eg. Rod cutting problem

Given a rod of length n and a list of prices for each length, determine the maximum revenue that can be obtained by cutting the rod and selling the parts.

Rod, length = 8

price = [1, 5, 8, 9, 10, 17, 17, 20]

len 1 2 3 4 5 6 7 8

Rod
↓

opt[8]
- 1 + opt[7]
  - 1 + opt[6]
  - 2 + opt[5]
  - 3 + opt[4]
  - 4 + opt[3]
  - 5 + opt[2]
  - 6 + opt[1]
  - 7 + opt[0]
- 2 + opt[6]
- 3 + opt[5]
- 4 + opt[4]
- 5 + opt[3]
- 6 + opt[2]
- 7 + opt[1]
- 8 + opt[0]

0

cutting

$$\text{maxProfit}(n) = \begin{cases} \max\limits_{1 \le i \le n} \left[\underline{\text{profit}\{i\}} + \text{maxProfit}\,\underline{\{n-i\}} \right] \\ 0, \quad n = 0 \end{cases}$$

$$\text{maxProfit}(3) = \max \left[(\text{profit}(1) + \text{maxProfit}(2)), \; (\text{profit}(2) + \text{maxProfit}(1)), \; (\text{profit}(3) + \text{maxProfit}(0)) \right]$$

=x=

Memoization (Top-Down approach)

↳ We store the results of subproblems to avoid redundant calculations. (implementing a cache for subproblem solutions)

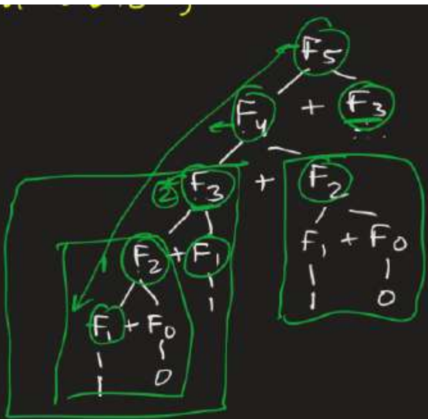
Calculations: (Implementing ...

Fibonacci problem

$$F_i = \begin{cases} F_{i-1} + F_{i-2}, & i > 1 \\ 1, & i = 1 \\ 0, & i = 0 \end{cases}$$

Simple Recursion:-

```
function fib(n):
    if n==0:
        return 0
    if n==1:
        return 1
    return fib(n-1)+fib(n-2)
```



$T = O(2^n)$

$S = O(2^n)$

$F_5 = F_4 + F_3$

$= F_3 + F_2 + F_2 + F_1$

$= F_2 + F_1 + F_1 + F_0 + F_1 + F_0 + F_1$

$= F_1 + F_0 + F_1 + F_1 + F_0 + F_1 + F_0 + F_1$

$F_5 \rightarrow 2$

$F_4 \rightarrow 2$

$\vdots$

$\underbrace{2 \cdot 2 \cdot 2 \cdots 2}_{n}$

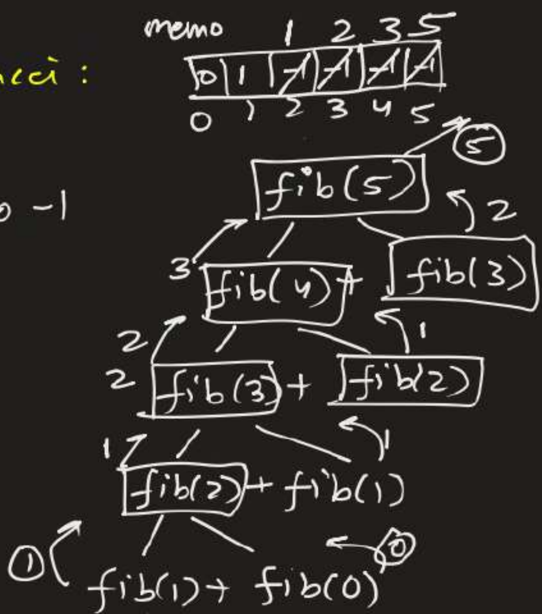
$O(2^n)$

Pseudo code for memoization based fibonacci :

memo = array of size n+1 , initialized to -1

memo[0] = 0, memo[1] = 1

```
function fibo (n):
    if memo[n] != -1 :
        return memo[n]
    memo[n] = fib(n-1) + fib(n-2)
    return memo[n]
```



$$\boxed{\begin{array}{l} T = O(n) \\ S = O(n) \end{array}}$$

Rod-Cutting Problem

$$\text{maxProfit}(n) = \begin{cases} \max\limits_{1 \le i \le n} \left[\underbrace{\text{profit}\{i\}}_{\text{cutting}} + \text{maxProfit}\{n-i\} \right] \\ 0, \quad n = 0 \end{cases}$$

Rod, length = 8

price = [1, 5, 8, 9, 10, 17, 17, 20]

len	1	2	3	4	5	6	7	8
price	1	5	8	9	10	17	17	20

Simple Recursion:-

```
maxProfit (n, profit []):
    if n==0:
        return 0
    mp = 0
    for (i: 1 -> n)
        mp = max(mp, profit[i] + maxProfit[n-i])
    return mp
```

maxprofit(4)

max(p(1) + $\boxed{\text{maxprofit(3)}}$ ...)

max(p[1] + mp(2), p[2] + mp[1], p[3] + mp(0))

max(p(1) + mp(1), p(2) + mp(0))

1[(p(1) + mp(0))

mp(n) → n, n, n, ..., n $\quad O(n^n)$

Pseudo code with memoization:

memo = array of size n+1, initialized to -1

memo[0] = 0

$\boxed{\begin{array}{l} T = O(n^2) \\ S = O(n) \end{array}}$

```
maxProfit(n, profit[]):
    if memo[n] != -1:
        return memo[n]

    for (i: 1 -> n)
        memo[n] = max(memo[n], profit[i] + maxProfit(n-i))

    return memo[n]
```

Rod, length = 8

price = [1, 5, 8, 9, 10, 17, 17, 20]

len 1 2 3 4 5 6 7 8

	0	1	5	8	10	13	17	18	22
memo	0	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~
	0	1	2	3	4	5	6	7	8

mp(8)

p[1]+mp(7) p[2]+mp(6) . . . . p[8]+mp(0)

0

p[1]+mp(6) p[2]+mp(5) . . . p[7]+mp(0)

0

p[1]+mp(5) . . . p[6]+mp(0)

0

p[1]+mp(4) . . . p[5]+mp(0)

mp(n)

mp(n-1) mp(n-2) . . . mp(1)

mp(n-2) mp(n-3) . . . mp(1)

$n-1$

$+$

$n-2$

$+$

$n-3$

$+$

$\vdots$

1

$T =$

$1 + 2 + 3 + \dots n-1$

$= \boxed{O(n^2)}$

$S = O(n)$

Tabulation Method (Bottom-up Approach)

↳ Solve the problem by iteratively solving subproblems & building up the solution from the base cases.

Fibonacci Problem:

$$f_i = \begin{cases} f_{i-1} + f_{i-2}, & i > 1 \\ 1, & i = 1 \\ 0, & i = 0 \end{cases}$$

```
function fib(n):
    dp = array of length n+1, initialized to -1
    dp[0] = 0
    dp[1] = 1
    for (i : 2 -> n)
        dp[i] = dp[i-1] + dp[i-2]
    return dp[n]
```

0	1	1	2	3	5	8	13
0	1	2	3	4	5	6	7

$T = O(n)$

$S = O(n)$

```
function fib(n):
    if (n ≤ 1) return n;
    last = 1,   secondlast = 0 , temp

    for (i : 2 → n)
        temp = last
        last = last + secondlast
        secondlast = temp

    return last
```

$$\boxed{T = O(n),\ S = O(1)}$$

fib(5)

last = ~~1~~ ~~2~~ ~~3~~ 5   sl = ~~0~~ ~~1~~ ~~2~~ 3

i = 2 t = 1 , last = 1, sl = 1
i = 3 t = 1 , last = 2 , sl = 1
i = 4 t = 2 , last = 3, sl = 2
i = 5 t = 3, last = 5, sl = 3
return 5

Rod Cutting Problem

$$\text{maxProfit}(n) = \begin{cases} \max\limits_{1 \le i \le n} \left[\underline{\text{profit}\{i\}}^{\text{cutting}} + \text{maxProfit}\{\underline{n-i}\} \right] \\ 0, \quad n = 0 \end{cases}$$

```
function maxProfit(n, profit[]):
    dp = array of size n+1, initialized to -1
    dp[0] = 0
    for (k: 1 -> n):
        for (i: 1 -> k):
            dp[k] = max(dp[k], p[i] + dp[k-i])
    return dp[n]
```

$$\boxed{\begin{array}{l} T = O(n^2) \\ S = O(n) \end{array}}$$

What is the time complexity of calculating the nth Fibonacci number using a naive recursive approach?

A. O(n)
B. O(n^2)
C. O(2^n)
D. O(log n)

Naive Recursion:. $O(2^n)$, DP :- $O(n)$

—x—

Which of the following statements is/are true about dynamic programming?

A. Dynamic Programming is an optimization technique.
B. Memoization is a technique used in dynamic programming.
C. Dynamic programming always provides the optimal solution. ← optimal substructure must exist.
D. Every problem that can be solved using backtracking can also be solved using dynamic programming.

↳ optimal substructure + overlapping subproblems.

You are given a rod of length 5 units. The prices for lengths 1, 2, 3, 4, and 5 are as follows:

Length	Price
1	4
2	5
3	6
4	P4
5	P5

What are the possible values of P4 and P5 that would allow you to achieve the maximum profit that can be obtained by cutting the rod and selling the pieces as 22 units?

A. P4 = 18, P5 = 20
B. P4 = 18, P5 = 22
C. P4 = 20, P5 = 22
D. P4 = 8, P5 = 22

	A	B	C	D
$1+4 \to 4+P_4$	22	22	24	12
$5 \to P_5$	20	22	22	22

Sol.

len	1	2	3	4	5
Profit	4	5	6	P_4	P_5

5:
$1+4 \to 4+P_4$
$2+3 \to 11$ ×
$1+1+3 \to 4\times 2+6 = 14$ ×
$1+2+2 \to 4+5\times 2 = 14$ ×
$1+1+1+2 \to 4\times 3+5 = 17$ ×
$1+1+1+1+1 \to 4\times 5 = 20$ ×
$5 \to P_5$