# CS 1332R WEEK 15

Dynamic Programming

Longest Common Subsequence

Final Exam Review

# ANNOUNCEMENTS

- ❑

# *Introduction to Dynamic Programming (DP)*

- A **strategy** for solving a class of problems that can be broken down into smaller, repetitive problems **that overlap** - different than just divide & conquer recursion
- Can reduce time complexity from exponential to polynomial
- Use Cases: KMP Failure Table, shortest path algorithms, scheduling algorithms, and so many more

## STEPS:

1. Break a problem down into smaller subproblems.
2. Solve these smaller problems and **save their solutions**.
3. If a subproblem is encountered again, just use the saved result (O(1)) instead of solving it again.

# *Motivation for Dynamic Programming*

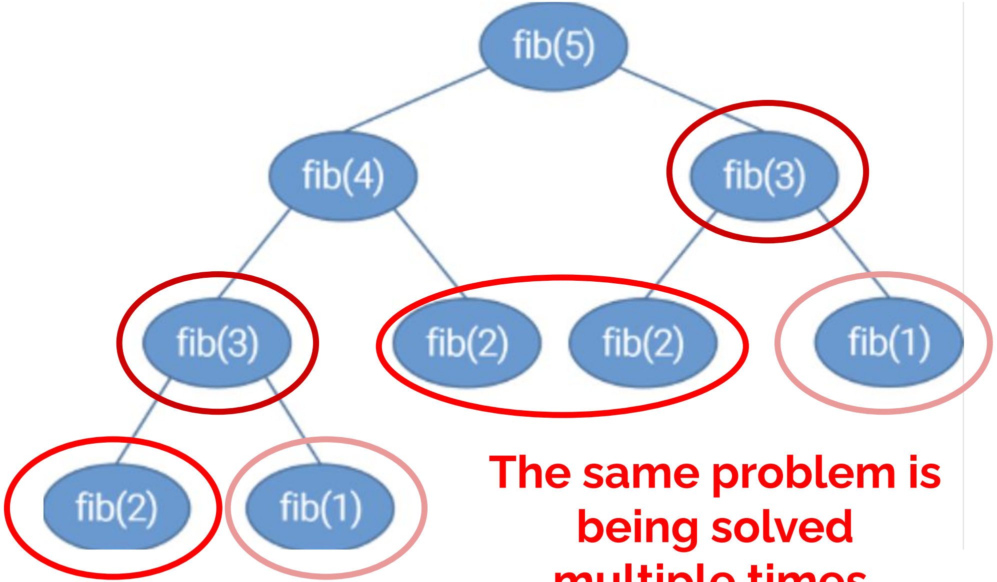
Let's explore the Fibonacci problem.

Fibonacci Sequence: 0, 1, 1, 2, 3, 5, 8, 13, …

```
fib(n) = fib(n - 1) + fib(n - 2)
```

## RECURSIVE, NON-DP SOLUTION

```
public int fib(int n) {
      if (n == 0) return 0;
      else if (n == 1) return 1;
          return fib(n - 1) + fib(n - 2);
}
```




**The same problem is being solved multiple times.**

# *Implementing Dynamic Programming*

## TWO APPROACHES

### MEMOIZATION

- ❏ Still utilizes recursion, “top-down” approach
- ❏ Saves previously solved solutions using some structure (e.g. map, array, etc.)

```
Map<Integer, Integer> memo;

public int fib(int n) {
        if (n == 0) return 0;
        else if (n == 1) return 1;
        if (!memo.containsKey(n)) {
            int result = fib(n - 1) + fib(n - 2);
            memo.put(n, result);
        }
        return memo.get(n);
}
```

### TABULATION

- ❏ Iterative, “bottom-up” approach
- ❏ Saves previously solved solutions in a table
- ❏ ***Faster than memoization***, no overhead space for recursion

****You've seen this before with KMP's failure table!!****

```
public int fib(int n) {
        int[] f = new int[n + 1];
        f[0] = 0;
        f[1] = 1;
        for (int k = 2; k <= n; k++) {
            f[k] = f[k - 1] + f[k - 2];
        }
        return f[n];
}
```

# *Longest Common Subsequence (LCS)*

- **SUBSEQUENCE**: A subset of a string where each character appears in the same order as in the strong. The characters are not necessarily contiguous in the string.

  **Ex: subsequences of BROWN**

  **BRO, BROW, BON, BOWN, BROWN**

- **PURPOSE**: Given two strings, find the longest subsequence that exists in both strings.

Typically…

```
X = first string                 Y = second string
n = length of X                  m = length of Y
i = column, index of X           j = row, index of Y
```

# *Longest Common Subsequence (LCS)*

## STEPS

1. Create an `m+1 x n+1` 2D array. The value at arr[i][j] is the length of the longest common subsequence of the strings X.substring(0, i) and J.substring(0, j).
2. Fill the first row and column with zeros.
3. Begin filling in the rows left to right, top to bottom using the following rule:

```
if X[i] == Y[j]:
     arr[i][j] = arr[i-1][j-1] + 1
else:
     arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

***choose cell to the left if equal***

## DIAGRAMMING SETUP

| | X: | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| Y: | | | s | t | r | i | n | g |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | s | 0 | | | | | | |
| 2 | t | 0 | | | | | | |
| 3 | r | 0 | | | | | | |
| 4 | i | 0 | | | | | | |
| 5 | n | 0 | | | | | | |

*location of final answer: arr[m][n]*

```
if X[i] == Y[j]:
        arr[i][j] = arr[i-1][j-1] + 1
else:
        arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| X: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y: | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 H | 0 | | | | | | | | | | | |
| 2 E | 0 | | | | | | | | | | | |
| 3 R | 0 | | | | | | | | | | | |
| 4 I | 0 | | | | | | | | | | | |
| 5 T | 0 | | | | | | | | | | | |
| 6 A | 0 | | | | | | | | | | | |
| 7 G | 0 | | | | | | | | | | | |
| 8 E | 0 | | | | | | | | | | | |

```
if X[i] == Y[j]:
        arr[i][j] = arr[i-1][j-1] + 1
else:
        arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| | X: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y: | | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | E | 0 | | | | | | | | | | | |
| 3 | R | 0 | | | | | | | | | | | |
| 4 | I | 0 | | | | | | | | | | | |
| 5 | T | 0 | | | | | | | | | | | |
| 6 | A | 0 | | | | | | | | | | | |
| 7 | G | 0 | | | | | | | | | | | |
| 8 | E | 0 | | | | | | | | | | | |

```
if X[i] == Y[j]:
        arr[i][j] = arr[i-1][j-1] + 1
else:
        arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| Y \ X: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 E | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 R | 0 | | | | | | | | | | | |
| 4 I | 0 | | | | | | | | | | | |
| 5 T | 0 | | | | | | | | | | | |
| 6 A | 0 | | | | | | | | | | | |
| 7 G | 0 | | | | | | | | | | | |
| 8 E | 0 | | | | | | | | | | | |

```
if X[i] == Y[j]:
        arr[i][j] = arr[i-1][j-1] + 1
else:
        arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| | X: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y: | | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | E | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | R | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 4 | I | 0 | | | | | | | | | | | |
| 5 | T | 0 | | | | | | | | | | | |
| 6 | A | 0 | | | | | | | | | | | |
| 7 | G | 0 | | | | | | | | | | | |
| 8 | E | 0 | | | | | | | | | | | |

```
if X[i] == Y[j]:
        arr[i][j] = arr[i-1][j-1] + 1
else:
        arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| Y: \ X: | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | E | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | R | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 4 | I | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 5 | T | 0 | | | | | | | | | | | |
| 6 | A | 0 | | | | | | | | | | | |
| 7 | G | 0 | | | | | | | | | | | |
| 8 | E | 0 | | | | | | | | | | | |

```
if X[i] == Y[j]:
	arr[i][j] = arr[i-1][j-1] + 1
else:
	arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| X: | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y: | | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | E | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | R | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 4 | I | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 5 | T | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 |
| 6 | A | 0 | | | | | | | | | | | |
| 7 | G | 0 | | | | | | | | | | | |
| 8 | E | 0 | | | | | | | | | | | |

```
if X[i] == Y[j]:
        arr[i][j] = arr[i-1][j-1] + 1
else:
        arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| | X: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y: | | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | E | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | R | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 4 | I | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 5 | T | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 |
| 6 | A | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 7 | G | 0 | | | | | | | | | | | |
| 8 | E | 0 | | | | | | | | | | | |

```
if X[i] == Y[j]:
        arr[i][j] = arr[i-1][j-1] + 1
else:
        arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| | X: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y: | | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | E | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | R | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 4 | I | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 5 | T | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 |
| 6 | A | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 7 | G | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 4 |
| 8 | E | 0 | | | | | | | | | | | |

```
if X[i] == Y[j]:
        arr[i][j] = arr[i-1][j-1] + 1
else:
        arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| | X: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y: | | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | E | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | R | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 4 | I | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 5 | T | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 |
| 6 | A | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 7 | G | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 4 |
| 8 | E | 0 | 1 | 2 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 5 |

```
if X[i] == Y[j]:
        arr[i][j] = arr[i-1][j-1] + 1
else:
        arr[i][j] = max(arr[i-1][j], arr[i][j-1])
```

# *LCS:* Practice

| | X: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y: | | | G | E | O | R | G | I | A | T | E | C | H |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | E | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | R | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 4 | I | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 5 | T | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 |
| 6 | A | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 7 | G | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 4 |
| 8 | E | 0 | 1 | 2 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 5 |

Length of LCS: 5

To find the LCS:

1. Start at bottom right corner.
2. Go left until the value at a cell came from the upper row.
3. Go up to that cell, save the common character if it came from the upper <u>left</u>.
4. Repeat step 2.

Can there be multiple longest common subsequences?

**YES.**

## *LEETCODE PROBLEMS*

[1143. Longest Common Subsequence]

[70. Climbing Stairs]

[322. Coin Change]

# FINAL EXAM REVIEW

## Jeopardy

## Socrative: CS1332

Practice exams in Canvas: Files -> Resources -> Recitation Materials -> Recitation Practice Exams

# Any questions?

**Name**
**Office Hours**
**Contact**

**Name**
**Office Hours**
**Contact**

*Let us know if there is anything specific you want out of recitation!*