Course Name: M.Sc. in Computer Science
Semester – I, Session: 2019-2021
Department of Computer Science
Name of Faculty: Gautam Mahapatra, Associate Professor
Subject: Advanced Data Structure – Dynamic Programming (DP)

Class Taken:
Date: 7th April 2020, Time: 4.30PM – 6.00PM

Number of Students Attended: 24 / 25

Software used: Zoom
Internet Service: Jio-Fi

Details of the subject taught:

Dynamic Programming (DP)

Dynamic Programming or Dynamic Optimization is problem solving technique where any complex problem is broken down into a collection of sub-problems (may not be similar type) and these are solved just once and stored in memory for reuse, and to save computation next time when the same sub-problem appears in the complex problem solution instead of re-computing this stored solution is used. To identify stored solution proper indexing is used for each sub-problem. For storing the solution of sub-problems storage requirement increases

Example
Recursive: Top-Down solution for Fibonacci Number

```
var m := map(0 → 0, 1 → 1)
function fib(n)
    if key n is not in map m
        m[n] := fib(n − 1) + fib(n − 2)
    return m[n]
```

Where 'm' is a mapping for storage of the results of sub-problems. It is memoization. Here we are using top-down approach.

Memoization:

In computing, **memoization** or **memoisation** is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again

In dynamic programming using proper indexing solutions of the sub-problems are stored in direct-accessible memory to avoid re-computation and this memorizing technique is called as 'Memoization'.

In DP Memoization is an important issue and efficiency of DP depends on this.

Steps to Solve Problem using DP

Step 1: Define sub-problems
Step 2: Finding recurrences
Step 3: Solving the base cases

Non-recursive: Bottom – up solution for Fibonacci number

```
function fib(n)
 if n = 0
        return 0
else
        var previousFib := 0, currentFib := 1
        repeat n − 1 times // loop is skipped if n = 1
                var newFib := previousFib + currentFib
                previousFib := currentFib
                currentFib := newFib
        return currentFib
```

Longest Common Substring (LCS):

The Longest Common Substring problem is, given two strings *s* and *t*, find the longest substring (contiguous characters) that is in both *s* and *t*. This has been used to find the similarity between two different genes. Note that this is not the same as the Longest Common Subsequence problem, in which characters are not necessarily contiguous.

According to the standard framed by World Health Organization (WHO) and the China National Center for Disease Control (CNCDC) for the confirmation of the presence of 2019-nCoV virus in the human body the Single Strand Positive RNA of the said virus should have the following gene sequence when it is tested using the Real-time reverse Transcription Polymerase Chain Reaction (RT-PCR) or Next Generation Sequencing (NGS) techniques:
Target - 1 (ORFlab-Open Reading Framelab)
Forward Primer: CCCTGTGGGTTTTACACTTAA
Reverse Primer: ACGATTGTGCATCAGCTGA
Probe Reading: 5'-FAM-CCGTCTGCGGTATGTGGAAAGGTTATGG-BHQ1-3'
Target-2 (Nucleocapsid protein-N)
Forward Primer: GGGGAACTTCTCCTGCTAGAAT
Reverse Primer: CAGACATTTTGCTCTCAAGCTG
Probe Reading: 5'-FAM- TTGCTGCTGCTTGACAGATT-TAMRA-3'

As an example, say that s = "tofoodie" and t = "toody". The longest substring in each is "ood" at three characters. There are several substrings of two characters, including "to" and "oo" and "od".

Solving LCS using Dynamic Programming:

Define sub-problems:
Let D_{ij} be the length of the LCS of $x_1, x_2, \dots, x_j$ and $y_1, y_2, \dots, y_i$.
Find the recurrence
If $x_i = y_j$, they both contribute to the LCS
$D_{ij} = D_{i-1,j-1} + 1$

Otherwise, either x_i or y_j does not contribute to the LCS, so one can be dropped

$D_{ij} = \max\{D_{i-1,j}, D_{i,j-1}\}$

Find and solve the base cases: $D_{i0} = D_{0j} = 0$

Let D[i,j] be the length of the longest matching string suffix between s1..si and a segment of t between t1..tj. If the ith character in *s* doesn't match the jth character in *t*, then D[i,j] is zero to indicate that there is no matching suffix.

More formally:

D[i,j] = 0 if s[i] not equal to t[j] Chars don't match, so no suffix matches

D[i,j] =D[i-1,j-1] +1 if s[i] = t[j] Next chars match, so previous matches+1

If the characters at position i and j do match, then use the number of matches for the strings that don't include characters i and j, then add one.

Here is the initial table that shows D[0,0]. The columns and rows are zero because there can be no match if one of the strings is empty.

D[0,0]		T (i=1)	O	F	O	O	D	I	E (i=8)
	0	0	0	0	0	0	0	0	0
T (j=1)	0								
O	0								
O	0								
D	0								
Y(j=5)	0								

Next we fill out D[1,1], D[2,1], D[3,1], etc. to D[8,1]. There is only match at D[1,1] so we fill out its cell with the value of D[0,0]+1:

D[0,0]		T (i=1)	O	F	O	O	D	I	E (i=8)
	0	0	0	0	0	0	0	0	0
T (j=1)	0	1	0	0	0	0	0	0	0
O	0								
O	0								
D	0								
Y(j=5)	0								

Next we fill out the second row, where j = 2. At the first "O" we add one to D[1,1] and then we also have matches at the other O's which would be matches for an individual character.

D[0,0]		T (i=1)	O	F	O	O	D	I	E (i=8)
	0	0	0	0	0	0	0	0	0
T (j=1)	0	1	0	0	0	0	0	0	0
O	0	0	2	0	1	1	0	0	0
O	0								
D	0								
Y(j=5)	0								

Next we fill out the third row, where j = 3. This row shows several sub-matches, with "O" by itself, and also "OO":

D[0,0]		T (i=1)	O	F	O	O	D	I	E (i=8)

	0	0	0	0	0	0	0	0	0
T (j=1)	0	1	0	0	0	0	0	0	0
O	0	0	2	0	1	1	0	0	0
O	0	0	1	0	1	2	0	0	0
D	0								
Y(j=5)	0								

Continuing for the rest of the table yields:

D[0,0]		T (i=1)	O	F	O	O	D	I	E (i=8)
	0	0	0	0	0	0	0	0	0
T (j=1)	0	1	0	0	0	0	0	0	0
O	0	0	2	0	1	1	0	0	0
O	0	0	1	0	1	2	0	0	0
D	0	0	0	0	0	0	3	0	0
Y(j=5)	0								

To find the longest common substrings we just scan through the table and pick out the entry with the largest value. In this case, it is the value 3 and tells us that the previous three characters are a match (OOD). It also gives us the matches of length 2, if we're interested in those, and the matches of length 1.
The runtime is Θ(mn) to fill out the table, where m is the length of s, and n is the length of t.

D[0,0]		T (i=1)	O	F	O	O	D	I	E (i=8)
	0	0	0	0	0	0	0	0	0
T (j=1)	0	1	0	0	0	0	0	0	0
O	0	0	2	0	1	1	0	0	0
O	0	0	1	0	1	2	0	0	0
D	0	0	0	0	0	0	3	0	0
Y(j=5)	0	0	0	0	0	0	0	0	0

```
Function LCS(X, Y)
for(i = 0; i <= n; i++) D[i][0] = 0;
for(j = 0; j <= m; j++) D[0][j] = 0;
for(i = 1; i <= n; i++) {
        for(j = 1; j <= m; j++) {
                if(x[i] == y[j])
                        D[i][j] = D[i-1][j-1] + 1;
                else
                        D[i][j] = max(D[i-1][j], D[i][j-1]);
        }
}
```

Example: TSP
Given a weighted graph with n nodes, find the shortest path that visits every node exactly once.

Brute force algorithm takes: $O(n!)$

Using DP takes:$O(n^2 2^n)$

Define sub-problems:
Let $f(i,S)$ is the length of the optimal path that visits every node in the set S exactly once and ends at i
There are approximately $n2^n$ sub-problems

Return $min_{i\in V} f(i,V)$ where V is the set of nodes of input graph $G=(V,E)$

Find the recurrence:
Consider a path that visits all nodes in S exactly once and ends at i
Right before arriving j, the path comes from some i in $S-\{j\}$
And that sub-path has to be the optimal one that covers $S-\{j\}$, ending at i
We just try all possible candidates for j

$$f(i,S) = min_{j\in S}\{c_{i,j} + f(j, S-\{j\})\}$$

Solving Base Cases:
For each node i, $f(j,\emptyset) = c_{j,i}$ for all non starting nodes j and particular starting node i.
Let following C be the cost matrix for a directed graph for we need a minimum cost tour starting from vertex 1:

$$C = \begin{bmatrix} 0 & 12 & 11 & 16 \\ 15 & 0 & 15 & 10 \\ 8 & 14 & 0 & 18 \\ 9 & 11 & 17 & 0 \end{bmatrix}$$

Calculation Steps for DP based solution of TSP

Step 1: Calculate base cases $\boldsymbol{f(i,\emptyset) = c_{i,1},\ 2 \le i \le n}$
We get

$f(2,\emptyset) = c_{2,1} = 15$
$f(3,\emptyset) = c_{3,1} = 08$
$f(4,\emptyset) = c_{4,1} = 09$

Step 2: C$\boldsymbol{alculate\ f(i,S) = min_{j\in S}\{C_{ij} + f(j,S-\{j\})\}}$
For $|S| = 1$
We get

- $f(2,\{3\}) = min_{j\in\{3\}}\{c_{2,j} + f(j,S-\{j\})\} = min_{j\in\{3\}}\{c_{2,3} + f(3,\{3\}-\{3\})\} = min_{j\in\{3\}}\{c_{2,3} + f(3,\emptyset)\} = min_{j\in\{3\}}\{15+8\} = 23$
- $f(3,\{2\}) = min_{j\in\{2\}}\{c_{3,j} + f(j,S-\{j\})\} = min_{j\in\{2\}}\{c_{3,2} + f(2,\{2\}-\{2\})\} = min_{j\in\{2\}}\{c_{3,2} + f(2,\emptyset)\} = min_{j\in\{2\}}\{14+15\} = 29$
- $f(4,\{2\}) = min_{j\in\{2\}}\{c_{4,j} + f(j,S-\{j\})\} = min_{j\in\{2\}}\{c_{4,2} + f(2,\{2\}-\{2\})\} = min_{j\in\{2\}}\{c_{4,2} + f(2,\emptyset)\} = min_{j\in\{2\}}\{11+15\} = 26$
- $f(2,\{4\}) = min_{j\in\{4\}}\{c_{2,j} + f(j,S-\{j\})\} = min_{j\in\{4\}}\{c_{2,4} + f(4,\{4\}-\{4\})\} = min_{j\in\{4\}}\{c_{2,4} + f(4,\emptyset)\} = min_{j\in\{4\}}\{10+9\} = 19$
- $f(3,\{4\}) = min_{j\in\{4\}}\{c_{3,j} + f(j,S-\{j\})\} = min_{j\in\{4\}}\{c_{2,4} + f(4,\{4\}-\{4\})\} = min_{j\in\{4\}}\{c_{3,4} + f(4,\emptyset)\} = min_{j\in\{4\}}\{18+9\} = 27$
- $f(4,\{3\}) = min_{j\in\{3\}}\{c_{4,j} + f(j,S-\{j\})\} = min_{j\in\{3\}}\{c_{4,3} + f(3,\{3\}-\{3\})\} = min_{j\in\{3\}}\{c_{4,3} + f(3,\emptyset)\} = min_{j\in\{3\}}\{17+8\} = 25$

C$\boldsymbol{alculate\ f(i,S) = min_{j\in S}\{C_{ij} + f(j,S-\{j\})\}}$
For $|S| = 2$

- $f(2,\{3,4\}) = min_{j\in\{3,4\}}\{c_{2,j} + f(j,S-\{j\})\} = min_{j\in\{3,4\}}\{c_{2,3} + f(3,\{3,4\}-\{3\}), c_{2,4} + f(4,\{3,4\}-\{4\})\} = min_{j\in\{3,4\}}\{c_{2,3} + f(3,\{4\}), c_{2,4} + f(4,\{3\})\}$
  $= min_{j\in\{3,4\}}\{15+27, 10+25\} = min_{j\in\{3,4\}}\{42,35\} = 35$
- $f(3,\{2,4\}) = min_{j\in\{2,4\}}\{c_{3,j} + f(j,S-\{j\})\} = min_{j\in\{2,4\}}\{c_{3,2} + f(2,\{2,4\}-\{2\}), c_{3,4} + f(4,\{2,4\}-\{4\})\} = min_{j\in\{2,4\}}\{c_{3,2} + f(2,\{4\}), c_{3,4} + f(4,\{2\})\}$
  $= min_{j\in\{2,4\}}\{14+19, 18+26\} = min_{j\in\{2,4\}}\{33,44\} = 33$
- $f(4,\{2,3\}) = min_{j\in\{2,3\}}\{c_{4,j} + f(j,S-\{j\})\} = min_{j\in\{2,3\}}\{c_{4,2} + f(2,\{2,3\}-\{2\}), c_{4,3} + f(3,\{2,3\}-\{3\})\} = min_{j\in\{2,3\}}\{c_{4,2} + f(2,\{3\}), c_{4,3} + f(3,\{2\})\}$
  $= min_{j\in\{2,3\}}\{11+23, 17+29\} = min_{j\in\{3,4\}}\{34,46\} = 34$

Step 3: Calculate the optimal solution $\boldsymbol{f^* = f(1,V-\{1\}) = min_{2\le k\le n}\{c_{1,k} + f(k,V-\{1,k\})\}}$
We get

- $f^* = f(1,\{2,3,4\})$
  $= min_{2\le k\le 4}\{c_{1,2} + f(2,\{2,3,4\}-\{2\}), c_{1,3} + f(3,\{2,3,4\}-\{3\}), c_{1,4} + f(4,\{2,3,4\}-\{4\})\}$
  $= min_{2\le k\le 4}\{c_{1,2} + f(2,\{3,4\}), c_{1,3} + f(3,\{2,4\}), c_{1,4} + f(4,\{2,3\})\} = min_{2\le k\le 4}\{12+35, 11+33, 16+43\} = \min\{47,44,49\} = 44$

And path for optimum cost 44 unit is identified as follows:

- $f^* = f(1,\{2,3,4\}) = 44$
- $c_{1,3} + f(3,\{2,4\}) : 1 \to 3$
- $c_{3,2} + f(2,\{4\}) : 3 \to 2$
- $c_{2,4} + f(4,\emptyset) : 2 \to 4$
- $f(4,\emptyset) = c_{4,1} : 4 \to 1$

- So the path is $1 \to 3 \to 2 \to 4 \to 1$

Example: **Chain Matrix Multiplication**
Matrix Multiplication:
$C = A \times B = (c_{i,j})_{m \times p}$
where $A = (a_{i,j})_{m \times n}$ and $B = (b_{i,j})_{m \times n}$
Time complexity is $O(mnp)$
More than two matrix multiplication as in case of transformation operations on image in Graphics [such as image registration (rotate, translate and scaling) – given a particular image which and target image, by these operations iteratively to change input image to target form] called as chain matrix multiplication.

Example 1:
$A = (a_{i,j})_{100 \times 1}, B = (b_{i,j})_{1 \times 100}$ and $C = (c_{i,j})_{100 \times 1}$
$A \times B \times C = \{(A \times B) \times C, A \times (B \times C)\} = \{(100 * 100) * 100,\ 100 + (100 * 100)\} = \{1000000, 10100\}$
So, the second case is less time consuming.

Example 2:
$$A \times B \times C \times D = \{A \times ((B \times C) \times D), A \times (B \times (C \times D)), (A \times B) \times (C \times D), ((A \times B) \times C) \times D, (A \times (B \times C)) \times D\}$$
Different orderings are possible so we need to find the optimum solution.

- $M = M_1 \times M_2 \times \ldots \times M_n$

  $= \{(m^1_{i,j})_{p_1 \times p_2} \times (m^2_{i,j})_{p_2 \times p_3} \times (m^3_{i,j})_{p_3 \times p_4} \times \ldots \times (m^{n-1}_{i,j})_{p_{n-1} \times p_n} \times (m^n_{i,j})_{p_n \times p_{n+1}}$
  $= \prod_{k=1}^{n} (m^k_{i,j})_{p_k \times p_{k+1}}$

- Chain Matrix multiplication problem can be represented as $P = \{p_1, p_2, \ldots, p_n, p_{n+1}\}$

- $M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j, i \le j$

Using DP:

$f(i,i) = 0, 1 \le i \le n$
$f(i,i+1) = p_i p_{i+1} p_{i+2}, kay(i,i+1) = i, 1 \le i < q$
$f(i,i+s) = min_{i \le k < (i+s)}\{f(i,k) + f(k+1,i+s) + p_i p_{k+1} p_{i+s+1}\},\ for\ 1 \le i \le (n-s)\ and 1 < s < n$
$kay(i,i+s) = value\ of\ k\ that\ obtained\ for\ above\ minimum$

Optimal solution
$f^* = f(1,n) = f\big(1,1+(n-1)\big) = min_{1 \le k < n}\{f(1,k) + f(k+1,n) + p_1 p_{k+1} p_{n+1}\}$

Example:
- $P = \{p_1, p_2, \ldots, p_5, p_{5+1}\} = \{10,5,1,10,2,10\}, n = 5$
- So the optimum value

$f^* = f(1,5) = min_{1 \le k < 5}\{f(1,k) + f(k+1,5) + p_1 p_{k+1} p_{5+1}\}$
$= \min\{f(1,1) + f(2,5) + p_1 p_{1+1} p_{5+1},\ f(1,2) + f(3,5) + p_1 p_{2+1} p_{5+1}, f(1,3) + f(4,5) + p_1 p_{3+1} p_{5+1}, f(1,4) + f(5,5) + p_1 p_{4+1} p_{5+1}\}$
$= min\{f(1,1) + f(2,5) + 10 \times 5 \times 10,\ f(1,2) + f(3,5) + 10 \times 1 \times 10, f(1,3) + f(4,5) + 10 \times 10 \times 10, f(1,4) + f(5,5) + 10 \times 2 \times 10\}$
$= \min\{0 + f(2,5) + 500, 50 + f(3,5) + 100, f(1,3) + 200 + 1000, f(1,4) + 0 + 200\}$

$f(2,5) = f(2,2+3) = min_{2 \le k < (2+3)}\{f(2,k) + f(k+1,5) + p_2 p_{k+1} p_{2+3+1}\}$
$= min\{f(2,2) + f(3,5) + p_2 p_{2+1} p_{2+3+1}, f(2,3) + f(4,5) + p_2 p_{3+1} p_{2+3+1}, f(2,4) + f(5,5) + p_2 p_{4+1} p_{2+3+1}\}$
$= \min\{f(2,2) + f(3,5) + 5 \times 1 \times 10, f(2,3) + f(4,5) + 5 \times 10 \times 10,\ f(2,4) + f(5,5) + 5 \times 2 \times 10\}$
$= \min\{0 + f(3,5) + 50, 50 + 200 + 500, f(2,4) + 0 + 100\}$
$f(3,5) = f(3,3+2) = min_{3 \le k < (3+2)}\{f(3,k) + f(k+1,5) + p_3 p_{k+1} p_{2+3+1}\}$
$= min\{f(3,3) + f(4,5) + p_3 p_{3+1} p_{2+3+1}, f(3,4) + f(5,5) + p_3 p_{4+1} p_{2+3+1}\}$
$= \min\{0 + 200 + 1 \times 10 \times 10, 20 + 0 + 1 \times 2 \times 10\} = 40$

$$f(2,4) = f(2,2+2) = min_{2\le k<(2+2)}\{f(2,k) + f(k+1,4) + p_2 p_{k+1} p_{2+2+1}\}$$
$$= min\{f(2,2) + f(3,4) + p_2 p_{2+1} p_{2+2+1}, f(2,3) + f(4,4) + p_2 p_{3+1} p_{2+2+1}\}$$
$$= \min\{0 + 20 + 5 \times 1 \times 2, 50 + 0 + 5 \times 10 \times 2\} = 30$$

$$f(2,5) = \min\{0 + f(3,5) + 50, 50 + 200 + 500, f(2,4) + 0 + 100\}$$
$$= \min\{0 + 40 + 50, 50 + 200 + 500, 30 + 0 + 100\} = 90$$

Using similar approaches we get:
$f(1,3) = 150\ and\ f(1,4) = 90$
So the optimum solution of this chain multiplication is:
$$f(1,5) = \min\{0 + f(2,5) + 500, 50 + f(3,5) + 100, f(1,3) + 200 + 1000, f(1,4) + 0 + 200\}$$
$$= \min\{0 + 90 + 500, 50 + 40 + 100, 150 + 200 + 1000, 90 + 0 + 200\} = 190$$

To know the sequence of multiplication following relations are used:
$M_{i,j} = M_i \times M_{i+1} \times \ldots \times M_j, i \le j$
$kay(1,5) =$ for which $f(1,5)\ is\ min = 2$
$M_{1,2} \times M_{2+1,5}$
$for\ M_{1,2},\ kay(1,2) = 1$
$for\ M_{2+1,5},\ kay(3,5) = 4$
$for\ M_{1,2} = M_{1,1} \times M_{2,2}$
$for\ M_{3,4} = M_{3,3} \times M_{4,4}$
$for\ M_{3,5} = M_{3,4} \times M_{5,5}$
$for\ M_{1,5} = M_{1,2} \times M_{3,5}$