Objectives

- 0-1 knapsack problem
- Recursive formula for solving 0-1 knapsack using Dynamic programming approach
- Example of 0-1 knapsack problem
- Algorithm and time complexity

0/1 Knapsack problem:

For a given weights and values (profits) of n items, put these items in a knapsack of capacity W to get the maximum total profit in the knapsack.

- Items are indivisible that is fraction of item cannot be selected. Either we have to select the whole item or completely discard the item.

What is the correct way to find such items?

1. Naïve Solution takes exponential time complexity that is $O(2^n)$.
2. Greedy approach takes polynomial time complexity but does not guarantee that 100% you will get the optimal solution for 0-1 knapsack problem. Although in case of fractional knapsack Greedy approach gives the optimal solution.
3. Dynamic programming approach gives the optimal solution of 0-1 knapsack problem.

Example 1:

Objects	Ob1	Ob2	Ob3	Ob4
Wt.	1	3	4	5
Value(p)	1	4	5	7

Capacity of knapsack (M or W) = 7

The recursive formula is:

$$V[i,w] = \max\{V[i-1,w], V[i-1,w-w[i]] + val[i]\}$$

p	wt		0	1	2	3	4	5	6	7
		0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1	1	1
4	3	2	0	1	1	4	5	5	5	5
5	4	3	0	1	1	4	5	6	6	9
7	5	4	0	1	1	4	5	7	8	9

Items selected are:

X1	X2	X3	X4
0	1	1	0

Example 2:

Objects	Ob1	Ob2	Ob3	Ob4
Wt.	2	3	4	5
Value(p)	1	2	5	6

Capacity of knapsack = 8

The recursive formula is:

$$V[i,w] = \max\{V[i-1,w], V[i-1,w-w[i]] + val[i]\}$$

p	wt		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5	6	6	7	8

Items selected are:

X1	X2	X3	X4
0	1	0	1

$Algorithm:$

```
int Knapsack(int m, int wt[], int val[], int n)
{
int K[n+1][m+1], i, w;
for(i = 0; i <= n; i++)
{
for(w = 0; w <= m; w++)
{
if (i == 0 || w == 0)
K[i][w] = 0
elseif (wt[i] <= w)
K[i][w] = max(val[i] + K[i-1]*(w - wt[i]), K[i-1][w])
else
K[i][w] = K[i-1][w]
}
}
return K[n][m];
}
```

Time Complexity: $O(nM)$ where n is the no. of items and M is the capacity of knapsack.

Longest Common Subsequence Problem using DP

- Longest Common Subsequence (LCS)Problem
- Dynamic Programming Approach
- Example of LCS
- Algorithm using DP approach
- Time Complexity

Longest Common Subsequence (LCS)

In the longest-common-subsequence problem, given two sequences

$$X = \{x_1, x_2, \ldots, x_m\}\ and Y = \{y_1, y_2, \ldots, y_n\}$$

Objective is to find a maximum length common subsequence of X *and* Y.

A subsequence is a sequence that appears in same relative order, but not necessarily contiguous.

Example:

Str1="abcdefg"

Str2="abxdfg"

Common subsequences are:

"a", "b", "d", "f", "g", "ab", "df", "dfg", "abd", "abdfg"

LCS="abdfg"

Dynamic Programming Approach:

If the last characters match

- $LT[i][j] = LT[i-1][j-1] + 1$

If the last characters do not match:

- $LT[i][j] = \max(LT[i-1][j], LT[i][j-1])$

Example 1:

Str1: A G G T A B

Str2: G X T X A Y B

	0	A	G	G	T	A	B
0	0	0	0	0	0	0	0
G	0	0	1	1	1	1	1
X	0	0	1	1	1	1	1
T	0	0	1	1	2	2	2
X	0	0	1	1	2	2	2
A	0	1	1	1	2	3	3
Y	0	1	1	1	2	3	3
B	0	1	1	1	2	3	4

To find the LCS sequence from the table start from the bottom right corner

Case 1: if the value is not the maximum of top and left cell then it is the part of LCS and we select the corresponding character and move diagonally up.

Case 2: if the value came from top cell then move up. If top cell and left cell have same value we can move in either direction.

Cases 1 and 2 are repeated until we reach at the cell where stored value is 0.

Example 2:

Str1: S T O N E

Str2: L O N G E S T

	0	L	O	N	G	E	S	T
0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	1
T	0	0	0	0	0	0	1	2
O	0	0	1	1	1	1	1	2
N	0	0	1	2	2	2	2	2
E	0	0	1	2	2	3	3	3

```
int LCS(char* str1, Char * str2, int m, int n)
{
int LT[m+1][n+1], i, j;
for(i = 0; i ≤ m; i++)
{
for(j = 0; j ≤ n; j++)
{
if (i == 0 || j == 0)
LT[i][j] = 0
elseif (str1[i-1] == str2[j-1])
LT[i][j] = 1 + LT[i-1][j-1];
else
LT[i][j] = max(LT[i-1][j], LT[i][j-1]);
}
}
return LT[m][n];
}
```

Assignment: (Ref. Chapter 15, Page No. 378, CORMEN)

Exercises 15.2-1, 15.2-2 and 15.2-3 based on Matrix Chain Multiplication Problem.

Assignment: (Ref. Chapter 15, Page No. 396, CORMEN)

Exercises 15.4-1, 15.4-2 and 15.4-3 based on Longest Common Subsequence Problem.

THANKS !