CSE221 ALGORITHMS

Topic: Dynamic Programming (LCS + Binary Knapsack)

Prepared by:
Saad Bin Sohan
BRAC University

Email: sohan.academics@gmail.com

GitHub: https://github.com/saad-bin-sohan

Dynamic Programming

any problem with the following two properties is a Dynamic Programming problem.

properties:

(i) Overlapping subproblems: by solving the subproblems we can solve the actual problem.

(ii) Optimal substructure: all

the subproblem must be solved by following the same methods.

All the problems having these two properties fall under Dynamic programming.

We will study two problem types that can be solved using dynamic programming -

a) Longest Common Subsequence (LCS)

b) Binary Knapsack

Longest Common Subsequence

Subsequence:

From a given data, all the possible sequential combinations (where the sequence of elements is same as original) are its subsequence.

e.g: ABCDE থেকে

A, AB, AC, BDE, ABCD, ABCDE ....

এগুলো সব valid subsequence.

কিন্তু ACB নেওয়া যাবে since B is

not after C.

e.g: A CEBD থেকে পাই,

A, CE, ACD etc....

"Common" subsequence: multiple data (e.g: string type data) এর মধ্যে common subsequence গুলো।

e.g:

CE

ABCDE → A, AB, AC, ACD, BDE .... ABCD

ACEBD → A, CE, ACD, ACED

common subsequence: A, CE, ACD....

"Longest" Common Subsequence:

→ common subsequence এর মধ্যে longest length এর টা.

e.g.:

CE

ABCDE → A, AB, AC, ACD, BDE .... ABCD

ACEBD → A CE ACD ACED

length: 1 length: 2 length: 3

"ABCDE" এবং "ACEBD" এর মধ্যে 3 length এর চেয়ে longer common subsequence পাওয়া যাবে না যে maximum number of data যারা serial এ আছে is 3.

∴ longest common subsequence: "ACD"

This approach is called "Brute force" because it follows the rawest approach.

n number data gives us $2^n - 1$ subsequence. so this approach is very inefficient and we need better approach (using dynamic programming)

Dynamic Programming

→ Table / 2d array use করে all sorts of DP problem solve করা হয়। it's called iterative DP.

Approach :

two datasets S, T with indices like:

$S = [1, 2, 3 \dots, i-1, i]$ → index এর value

$t = [1, 2, 3 \dots, j-1, j]$ → index এর মান

we start with comparing elements at the end

pseudocode:

if $S[i] == T[j]$:

$$LCS[i, j] = 1 + LCS[i-1, j-1]$$

else:

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

e.g.: $S =$ Panama

$T =$ Banana

LCS value কত?

soln:

	empty string	p	a	n	a	m	a
empty string							
b							
a							
n							
a							
n							
a							

now table টার (i, j) তম ঘরে থাকবে ঐ সংখ্যা that represents-

s এর 0th থেকে (i-1) তম index পর্যন্ত যাওয়া sequenগুলো

t এর 0th থেকে (j-1) তম index পর্যন্ত যাওয়া sequence গুলো

এই sequence গুলোর মধ্যে longest comman টার length

and table টার last row, last Column এর box টাতে final LCS এর answer থাকবে

if S[i]==T[j]:

LCS[i, j] = 1+ LCS[i-1, j-1] (diagonal)

else:

$LCS[i,j] = \max(LCS\underbrace{[i-1, j]}^{\text{left}}, LCS\underbrace{[i, j-1]}^{\text{top}})$

	empty string	p	a	n	a	m	a
empty string	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0
a	0	0	1	1	1	1	1
n	0	0	1	2	2	2	2
a	0	0	1	2	3	3	3
n	0	0	0	2	3	3	3
a	0	0	1	2	3	3	4

final answer

0 টার reason empty string এর সাথে অন্য কারো common element নাই। so no longest common sequence

final answer 4 use করে LCS এর string টাও পেতে পারি। we have to start bottom to up using 4 and যে যার থেকে আসছে ওভাবে trace back করতে করতে matching letter গুলো নিবো।

LCS: anaa

[answer]

Practice problem

$\boxed{1}$ S = human

→ answer should be 5

T = chimpanzee

$\boxed{2}$ S = algorithm

T = lithium

Binary Knapsack

→ only difference with fractional knapsack is that binary knapsack doesn't allow choosing a partial amount of any item. You either take the entirely available amount of that item or you don't take that item at all.

NB: Fractional knapsack → Greedy method এ solveable

Binary knapsack → Greedy দিয়ে solveable না। we need DP.

Que solve using binary knapsack

Knapsack = 8 kg

item	price	weight
A	100	5
B	60	4
C	60	4
D	20	4

if we solved the problem of binary knapsack using greedy approach

item	price	weight	price per weight
A	100	5	20
B	60	4	15
C	60	4	15
D	20	4	5

according to greedy approach since A has max value per unit we choose A item first and since its binary knapsack we have to take whole amount of A.

$\therefore$ A $\rightarrow$ 5 kg $\rightarrow$ 100 TK but since limit is 8 kg we can't take any other item since if we did, we had to choose whole of it and no item is within 3 kg (8-5).

Que solve using binary knapsack (DP approach)

item	weight	price
A	2	3
B	3	4
C	4	5
D	3	6

soln: নতুন table বানাবো

column heading → 0 থেকে knapsack limit অবধি

row heading → item names and their weight

	0	1	2	3	4	5
0	0	0	0	0	0	0
A(2)	0					
B (3)	0					
C (4)	0					
D (3)	0					

If item_weight > weight_limit:

copy top [can't take]

else:

Max (top, item_price + dp[prev_row][capacity - item_weight])