LEC16 LCS & LIS

4.5 https://github.com/GUMI-21/MIT6.006_note

Dynamic Programming Review

- Recursion where subproblem dependencies overlap, forming DAG
- Recurse but re-use (Top down: record and lookup subproblem solutions)
- Carefull brute force (Bottom up: do each subproblem in order)

Dynamic Programming Steps(SRT BOT)

Subproblem
Relate
Topological order DAG
Base cases
Original problem
Time analysis
=>single sequence, then {prefixed S[:i], suffixes S[:i], substrings S[i:j]}

Longest Common Subsequences(LCS)

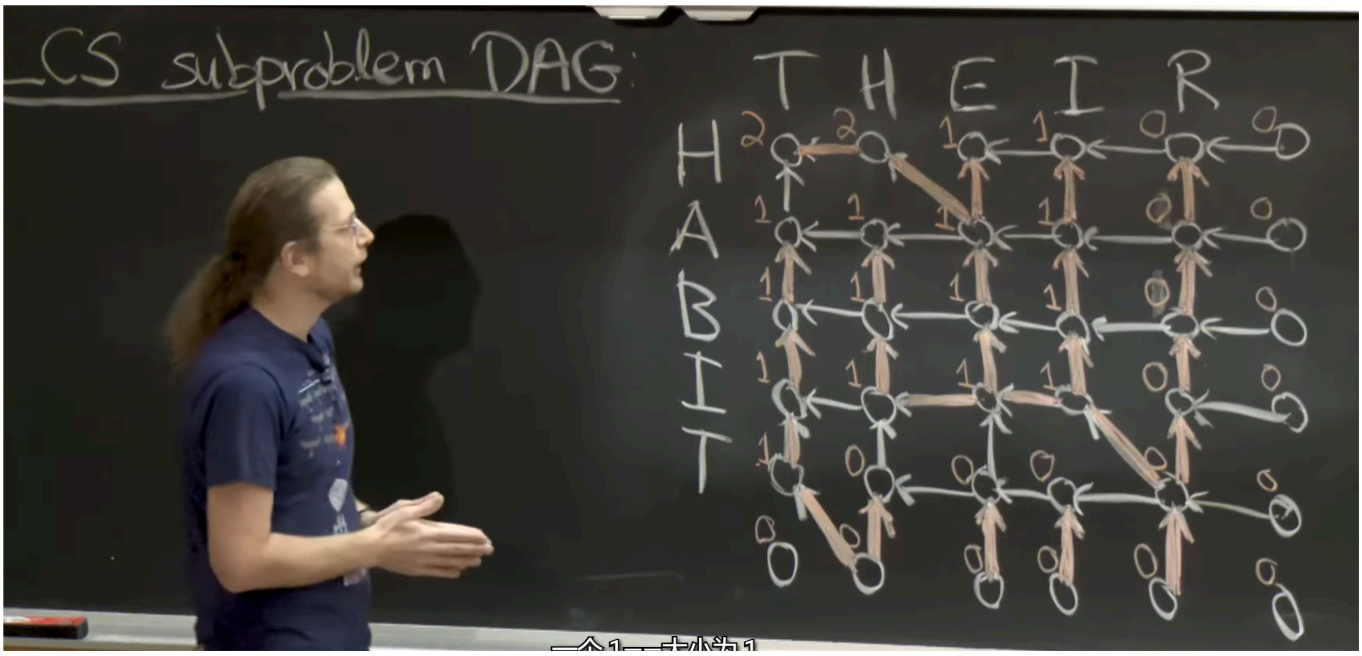
EX:
HIEROGLYPHOLOGY
MICHAELANGELO
=> HELLO
given two sequences A & B, find longest sequence L that's subsequence of both A & B

- **Trick**
  Subproblems for mutiple inputs: multiply subproblem spaces (cross product)
  ->every subproblem in LCS is a pair of suffixes
  LCS:
  -Subproblems: L(i,j)=LCS(A[i:],B[j:]), for 0<=i<=|A|, 0<=j<=|B|
  -Relate: L(i, j) =
  if A[i]=B[\j] : 1+L(i+1, j+1)
  else: max{L(i+1,j), L(i, j+1)}} >= one of A[i] & B[j] not in LCS (Think about first letter of subsequence)
  -Topological order: for i = |A|,...,0: for j=|B|,...,0
  -BaseCase: L(|A|,j) = 0 = L(i,|B|)

-Original: L(0,0)
-Time: $\theta(A * B)$ subproblems. * *$\theta(1)$

A EXAMPLE

LCS is Hi, Parent pointers to find LCS



diagonal edge: A[i]=B[j]*
else: compare i+1,j and i,j+1 take the max value to node

Longest Increasing Subsequence(LIS)

CARBOHYDRATE
find LIS of alphabetically increasing
=> A B O R T
given Seq. A, find LIS(A) Strictly increasing.
Attempt:
-Subproblems: L(i) = LIS(A[i:])
-Relate: L(i) = max{L(i+1), 1+L(i+1)} there is something error here, 1+L(i+1) is always bigger than L(i+1)
-Original: L(0)

So we need to add a constraint to Subproblem
=>
-Subproblems: L(i) = LIS(A[i:]) that starts with A[i] (includes)
-Relate: L(i) = 1 + max{L(j) | i<j<|A|, where A[i] < A[j]} or {0}
-Original: max{L(i) | for all i <= |A|}
If DP array has mutiply dimension, we need to initialize All boundary base case!!
-Topo order: for i = |A|,...,0

-Base case: L(|A|) = 0 (*no item*, A[|A|])
-Time anaylsis: $\theta(|A|)$ subproblems work * $\theta(|A|)$ non-recursive work(choices) / per subproblem so running time is $O(|A|^2)$ and add a max operation of original problem takes $O(|A|)$, so the running time is $O(|A|^2)$

LIS Subproblem DAG

A EXAMPLE
EMPATHY => EMPTY
-> SSSP from base to every vertex, and find the longest path!
the right path is the shortest path tree
Lot of DP problems can written as Shortest paths problem

Alternating Coin Game

a sequence of coins. 5 10 100 25
-given sequence conis of value V_0,...,V_n-1
-every turn on player choose the first or the last coin.
-Subproblems: (Substrings)
X(i,j, p) = max. total value I can get from coins of values V_i,...,V_j
there is no DP need suffix and prefix, if you need to do, meeans use substrings
p = {me, you}
if player p goes first
-relate: 2cases
X(i,j, me) = max{ $X(i+1,j,you)+V_i, X(i,j-1,you)+V_j$}
X(i,j, you) = min{$X(i+1,j,me), X(i,j-1,me)$}
Topological: increasing j-i
Basecase: X(i,j,me) = V_i, X(i,j,you) = 0
Original: O(0,n,me)
T: $\theta(n^2)$

Subproblem Constraints and Expansion