LEC18 DP4

Recall

SRTBOT paradigm for recursive alg. design & with memoization DP algorithm desion.
-Subproblem definition

- for sequence S try prefixes S[:i], suffixes S[i:] Substring S[i:j]
- for nonnegative integer K, try integers in [0,k]
- add Subproblems & constraints to "remember state"
  -Relate subproblem solutions recursively
- identify guestion about subproblem solution that if you knew answer, reduces to "smaller" subproblems
- locally brute-force all answers to question
- can think of correctly guessing answer, then loop
  -Topological order on subprobs => DAG
  -Base case of relation
  -original problem
  -Time analysis

$$\sum_{x \in X} \text{work}(x), \text{ or if work}(x) = O(W) \text{ for all } x \in X, \text{ then } |X| \cdot O(W)$$

- work(x) measures nonrecursive work in relation; treat recursions as taking O(1) time

Rod cutting:

given rod of lenght L & value V(l) of rod of length l for all $l \in \{1, 2, \ldots L\}$
what's max-value partition of length-l rod?
-example: L=7,
l: 1 2 3 4 5 6 7
v(l): 1 10 13 18 20 31 32
-> 6+1 -> 31+1=32
or 3 + 2 + 2 = 13 + 10 + 10 = 33 the best

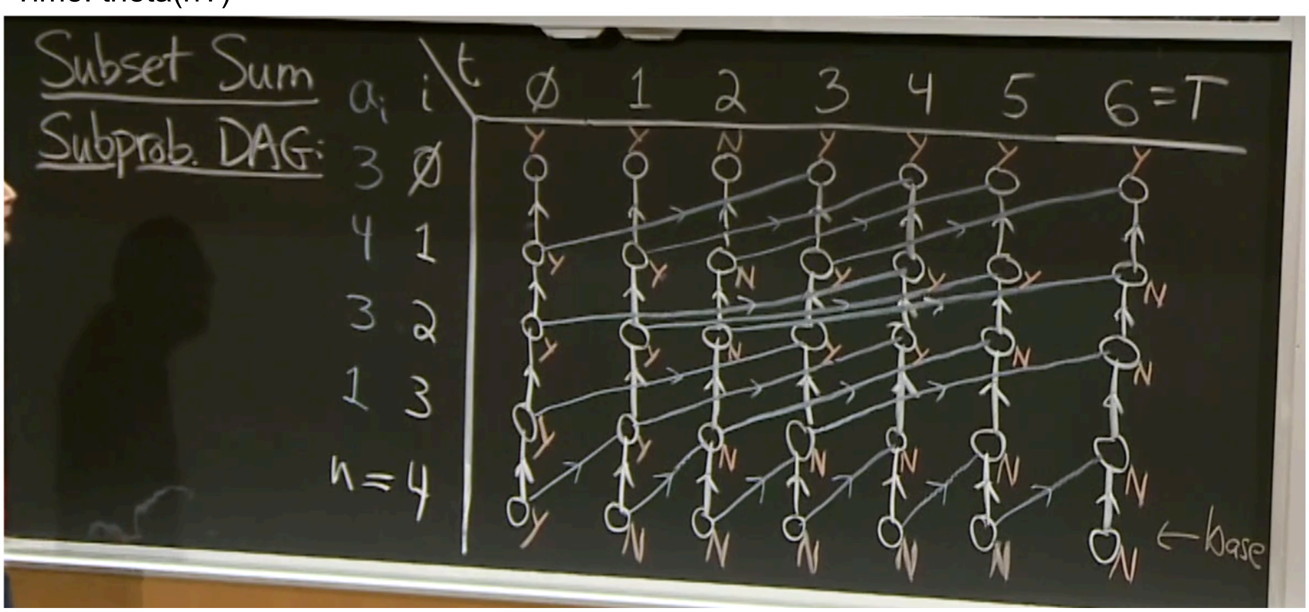
- SRTBOT
  -Subproblems:
  x(l) = max value partition of length l for l = 0,1,....,L
  -Relate:

X(l) = max {v(p) + X(l-p) | for p in 1 <= p <= l}
-Topo. order:
increasing l, for l = 0, 1, ..., L
-Base case: x(0) = 0
-Original: x(L)
-Time: $\theta(L)\ subprobs. * O(L) time = O(L^2) time$
Is $\theta(L^2)$ polynomial time? Yes
(Strongly) polynomial time = polynomial in input size (measured in words)

Subset Sum

given multiset A=$\{a_0, a_1, a_2, ..., a_{\{}n - 1\}\}$ of n integers & target sum T, does any subset S <= A sum to T ?
-example: A = {2, 5, 7, 8, 9} T = 21, 25
for 21 => YES, S={5,7,8}
for 25 => NO
-decision problem: YES/NO answer

- SRTBOT for (SS)
  -Subproblems:
  X(i, t) = does any subset S in A[i:] sum to t, for i = 0,1,...,n, t = 0,1,...,T
  -Relate
  X(i, t) = OR(any) {x(i+1, t), <- a_i not in S,
  x(i+1, t-a_i) if a_i <= t}
  -Topological order: decreasing i
  -Base case: x(n,t) = { if t=0, yes; otherwise no.}
  -Original problem: X(0,T)
  -Time: theta(nT)



->
Is $\theta(nT)$ is polynomial time? NO: not polynomial input size of n+1
->Beacuse input is size n+1
-know T <= 2^w , w >= lgn, but w could be >> lgn, e.g. w=n, T<=2^n
means nT could be exponential in n+1
->this is a *pseudo-polynomial*

Pseudopolynomial

polynomial input size & input integers (T)
=>polynomial if input intergers <= polynomial input size
others
-counting sort, DAA, Fibonacci, Rdix sort pseudopoly

🌟 以 Subset Sum 为例：

🔷 Original Problem:

> 给定一个数组 A[0..n-1]，是否存在某个子集 S，使得它的和是 T？

我们用 `x(0, T)` 表示这个问题：

> 用 A[0:] 是否能组成 T？

✅ 接下来，用逆向归纳思考：

- 想组成 `T`，有两个选择：
  1. 不用 A[0]，变成：`x(1, T)`
  2. 用 A[0]，变成：`x(1, T - A[0])`

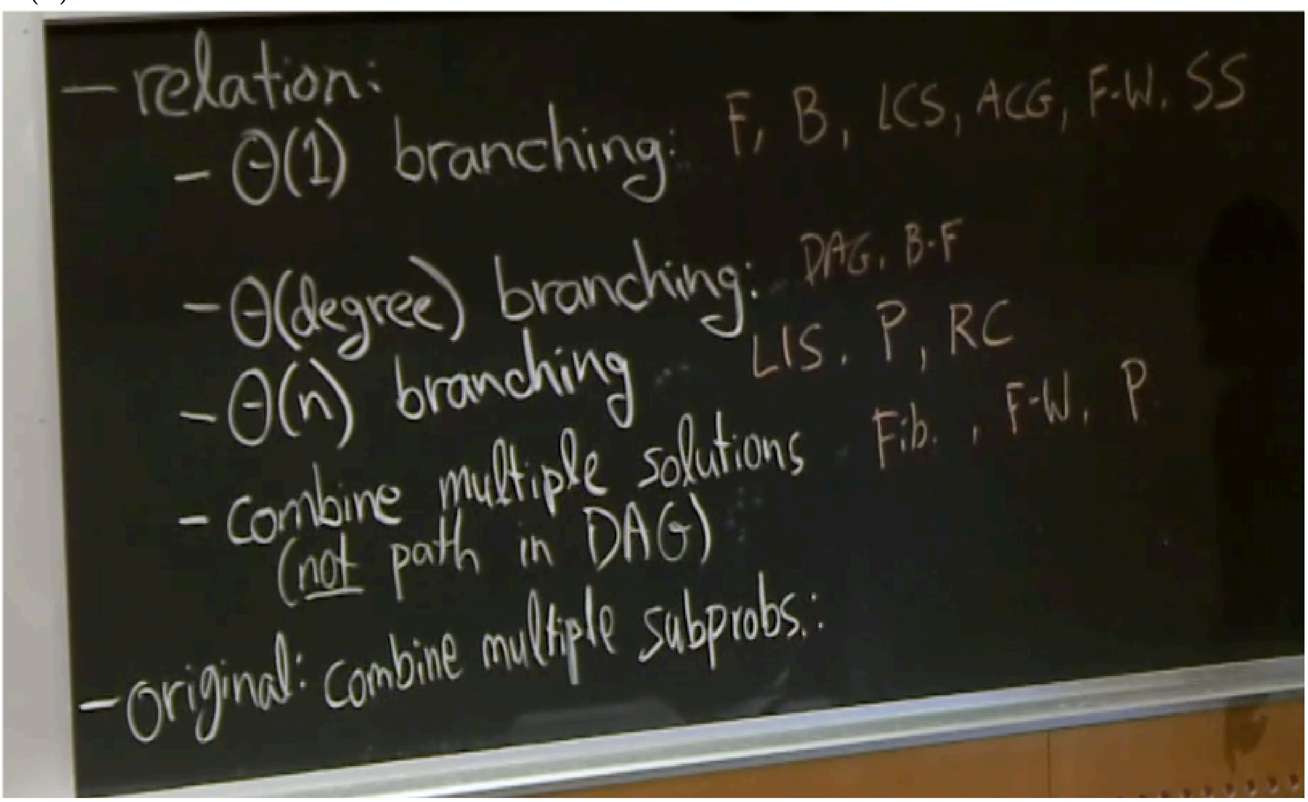
也就是说：

$$x(0,T) = x(1,T) \text{ or } x(1,T - A[0])$$

这时候你就会发现：

> ❗ 我们不是只关心 `x(i, T)`，我们会访问到所有 `x(i, t)`，只要 `0 <= t <= T`

Main features of DP

- Subpreoblems
  -prefixes/suffixes: Bowling game,LCS, LIS, Floyd-warshall, subset sum
  -substrings: ACG, Paren
  -multiple sequences: LCS
  -integers: Rod Cutting, Subset sum, Fibonacci
  -pseudopolynomial
  -vertices
- Subproblem constraints / expansion
  -nonexpansive constraint: LIS
  -2x: ACG, P
  -$\theta(1)x$: Piano
  -$\theta(n)x : Bellman - Ford$
  

Supplement

- Subproblem Expension
  在动态规划的子问题中，在尝试解决一个特定的子问题时，我们发现为了得到最优解，需要更多的信息或者状态，这通常意味着子问题需要进一步的进行分解，或者考虑额外的约束条件。
  Introduce these missing information or constraint.!
- Guess and Brute Force
  In subproblem, we Brute Force All possiable situation.

🧠 常见动态规划问题：子问题定义套路表

问题类型	原问题描述	子问题定义	状态参数	状态转移思路	常见思考角度
🧮 Fibonacci	F(n) = F(n-1)+F(n-2)	F(i): 前 i 项的结果	i	F(i) = F(i-1) + F(i-2)	顺推 or 递归
🎒 0/1 背包	选若干物品，总重不超过 W，最大价值	dp(i, w): 前 i 件物品，总重 w 时最大值	i, w	dp(i, w) = max(不选, 选)	**原问题角度推状态转移**
🧩 Subset Sum	A 中是否存在子集和为 T?	x(i, t): A[i:] 是否能组成 t	i, t	x(i, t) = x(i+1, t) or x(i+1, t-A[i])	**从原问题反推，目标是 t**
📚 LCS (最长公共子序列)	找两个串的最长公共子序列	dp(i, j): A[0:i], B[0:j] 的LCS长度	i, j	若匹配则 +1，否则 max(左，上)	子问题必须保留两个索引信息
✂️ Edit Distance	从 A 转换到 B 的最小操作数	dp(i, j): A[0:i], B[0:j] 的编辑距离	i, j	插入、删除、替换三种操作	**从尾部逐步考虑操作**
📈 LIS (最长上升子序列)	找 A 的最长上升子序列	dp(i): 以 A[i] 结尾的 LIS 长度	i	dp(i) = max(dp(j)+1), j<i 且 A[j]<A[i]	**从结尾“反推”最优子结构**
🌐 Floyd-Warshall	所有点对最短路径	d(u,v,k): 只允许中间节点 ≤k 的最短路	u,v,k	d(u,v,k)=min(d(u,v,k−1), d(u,k,k−1)+d(k,v,k−1))	增量构建可用点集合
⛓️ Matrix Chain Multiplication	最优矩阵相乘顺序	dp(i,j): A[i]~A[j] 的最小乘法次数	i, j	枚举中间断点 k	**子结构包含子区间**
🧗 Climbing Stairs	每次上 1 或 2 级，多少种走法?	dp(i): 走到 i 有几种方式	i	dp(i) = dp(i-1) + dp(i-2)	跟 Fibonacci 类似
🧮 Coin Change	用最少的硬币组成金额	dp(i): 组成金额 i 所需	i	dp(i) = min(dp(i - coin) + 1)	**从目标金额 i 出发**

最少硬币数