DESIGN AND ANALYSIS OF ALGORITHMS

CS 4120/5120

ELEMENTS OF DYNAMIC PROGRAMMING

AGENDA

- DP VS Divide and conquer
- Elements of DP

DYNAMIC PROGRAMMING

- The process of solving the rod-cutting problem is a technique that is called ***dynamic programming*** (DP).
  - "Programming" in this context refers to a tabular method, not to writing computer code.

- The key technique is to store the solution to each such problem in case it should reappear.

DP VS. DIVIDE AND CONQUER

- Divide and conquer
  - Constantly cuts a problem into equal parts to make subproblems
  - Solves the original problem by solving closely related subproblems
  - Combine solutions to the subproblems.
  - No overlapping subproblems
    - Same subproblem do not arise repeatedly.

- Dynamic programming
  - Subproblems often arise from making a choice.
  - Solves original problem by solving subproblem of the same form.
  - Combine solutions to the subproblems.
  - Overlapping subproblems
    - Same subproblem may rise from different decisions

WHEN TO APPLY DP?

- Applies to optimization problems in which we make a set of choices **in order to arrive at an optimal solution**.
  - Many possible solutions with each having a ***value***.
  - We wish to find a solution with the optimal ***value***.
    - Maximum or minimum
  - We call such a solution **an** optimal solution.
  - Rod-cutting problem
    - 2^n solutions, each solution generates a revenue
    - Goal: to find a solution generates maximum revenue

HOW TO APPLY DP ?

- **Generally**, there are four steps to take for developing a DP solution.
  - **Step 1**: Characterize the structure of an optimal solution
  - **Step 2**: Recursively define the ***value*** of an optimization.
  - **Step 3**: Compute the ***value*** of an optimal solution.
  - **Step 4**: Construct the optimal solution from the computed information.

APPLYING DP
STEP 1

- **Step 1**: Characterize the structure of an optimal solution
  - Discover the ***optimal substructure*** of the problem.

- Rod-cutting problem with input being an n-inch rod
  - Solution
    - Cut the rod at i-inch
    - Then find an optimal way to cut the remaining $(n - i)$-inch rod
  - The optimal substructure is characterized by optimally cutting up an n-inch rod incorporates optimally cutting up one or more shorter rod.

APPLYING DP
STEP 2

- **Step 2**: Recursively define the ***value*** of an optimization.
  - Take advantage of the optimal substructure to recursively compute the optimal ***value***.

- Rod-cutting problem with input being an n-inch rod
  - Previous characterization
    - The optimal substructure is characterized by optimally cutting up an n-inch rod incorporates optimally cutting up one or more shorter rod.
  - $r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$

APPLYING DP
STEP 3

- **Step 3**: Compute the ***value*** of an optimal solution.
  - Design an algorithm to compute the value.

- Rod-cutting problem with input being an n-inch rod
  - Intuitive algorithms. CUT-ROD (p, n) takes $O(2^n)$
  - Top-down solution with memoization costs $O(n^2)$
    - MEMOIZED-CUT-ROD (p, n) and MEMOIZED-CUT-ROD-AUX (p, n, r)
      - The r is a tabular record of solutions to subproblems.
  - Bottom-up solution BOTTOM-UP-CUT-ROD (p, n) costs $O(n^2)$

APPLYING DP STEP 4

- **Step 4**: Construct the optimal solution from the computed information.
  - Extended algorithm that not only computes and stores the value, but also records the choices the algorithm makes to obtain the optimal value.

- Rod-cutting problem with input being an n-inch rod
  - The EXTENDED-BOTTOM-UP-CUT-ROD (p, n)
    - The computes the optimal value r while maintaining the solution s.
  - PRINT-CUT-ROD-SOLUTION (p, n)

ELEMENTS OF DP

- The **two key ingredients** that an optimization problem must have in order for dynamic programming to apply:
  - Optimal substructure
  - Overlapping subproblems
- **Reconstructing an optimal solution**
- **Memoization**

OPTIMAL SUBSTRUCTURE KEY INGREDIENTS #1

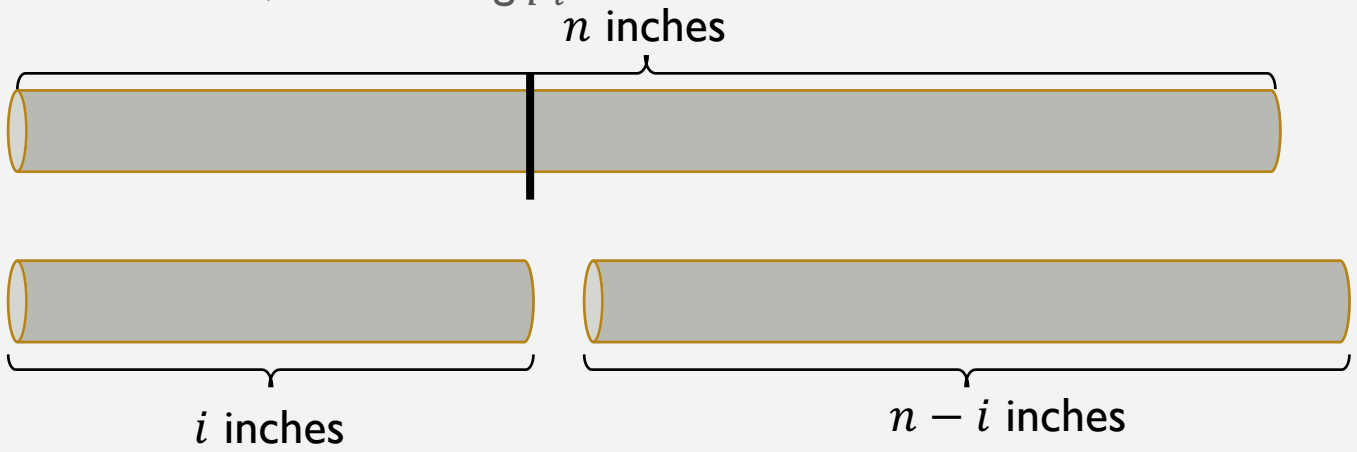
- A problem exhibits ***optimal substructure*** if an optimal solution to the problem contains within it optimal solutions to subproblems.
  - This is a sign that dynamic programming **might** be applied.
    - Later in the semester, we will see greedy strategy also utilizes optimal substructure.

- Finding the optimal substructure is the pathway to building an optimal solution to the problem.
  - Helps us recursively compute the optimal value (normally a formula is derived)
  - We can design an algorithm to compute the formula.
- How to find the optimal substructure?

FINDING THE OPTIMAL SUBSTRUCTURE STEP 1

- **Step 1**: A solution to the problem consists of making a choice.

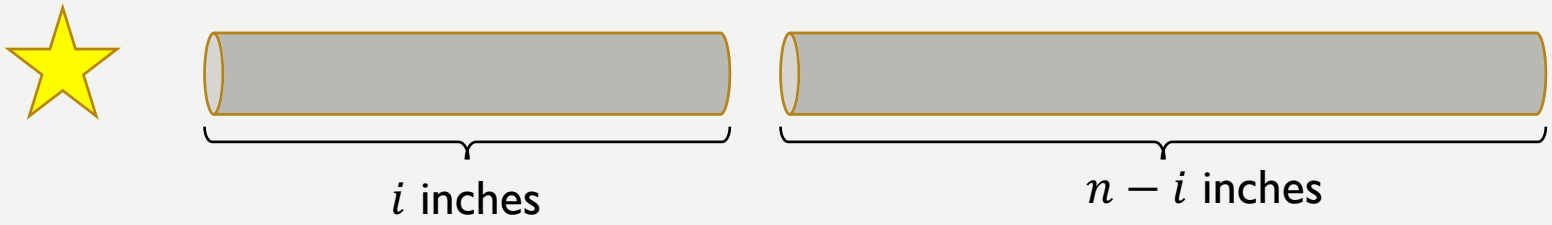
- Rod-cutting problem
  - Choosing an initial cut at i , or choosing p_i.



FINDING THE OPTIMAL SUBSTRUCTURE STEP 2

- **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - At this point, you do not concern yourself with how to determine this choice.
  - Suppose that initially cutting at i-inch mark would lead to an optimal solution.

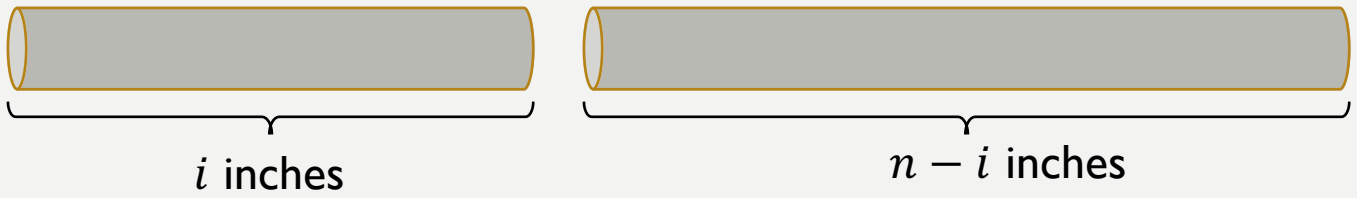
- Rod-cutting problem
  - Choosing an initial cut at i, or choosing p_i.



FINDING THE OPTIMAL SUBSTRUCTURE STEP 3

- **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.

- Rod-cutting problem
  - The space of the subproblem would be cutting from the far-left end then dealing with the remaining rod, i.e., **1 to $n - i$**.



FINDING THE OPTIMAL SUBSTRUCTURE STEP 4 (LAST STEP)

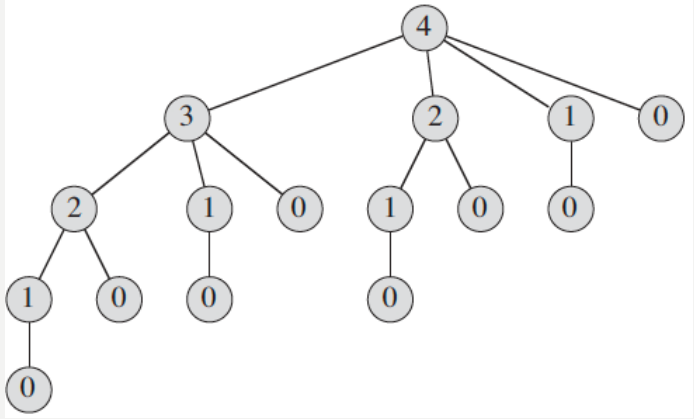
- **Step 4**: Show the solutions to the subproblem used within an optimal solution to the problem must themselves be optimal by using a "cut-and-paste" technique.
  - This step is referring to the proof of the optimal substructure.
  - A commonly used technique to prove (very often) the optimal substructure of a problem with a possible dynamic programming solution is **contradiction.**

FINDING THE OPTIMAL SUBSTRUCTURE GENERAL NOTES

- Optimal substructure varies across problem domains in two ways:
  - **how many subproblems** an optimal solution to the original problem uses, and
    - We shall see in the next example problem that more than one subproblem arises after we make a choice.
  - **how many choices** we have in determining which subproblem(s) to use in an optimal solution.
    - In the rod-cutting problem for an n-inch rod, obviously, we have n ($1 + \lfloor n/2 \rfloor$ to be more accurate) choices that determine which subproblem to use.

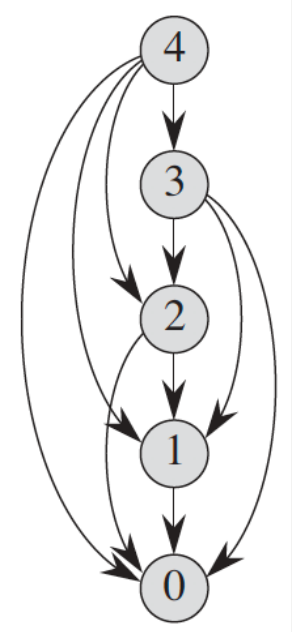
OVERLAPPING SUBPROBLEMS KEY INGREDIENTS #2

- When a recursive algorithm revisits the same problem repeatedly, we say that the optimization problem has **overlapping subproblems**.
  - DP takes advantage by **solving each subproblem ONCE** and then
  - **storing the solution in a table** where it can be looked up when needed, **using constant time per lookup**.
    - The way the recursion works in the CUT-ROD algorithm is very similar to a *depth-first-search* path.



SUBPROBLEM GRAPHS

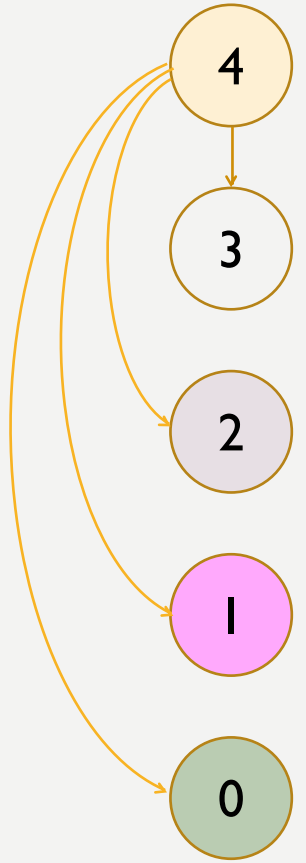
- A ***subproblem graph*** describes the set of subproblems involved in solving a problem and how the subproblems depend on one another.
  - It is a ***directed*** graph.
  - The vertices in a subproblem graph represent **distinct** subproblems.
  - A directed edge from the vertex for subproblem x to the vertex for subproblem y ***if*** determining an optimal solution for subproblem x involves directly considering an optimal solution for subproblem y.



READING/DRAWING A SUBPROBLEM GRAPH

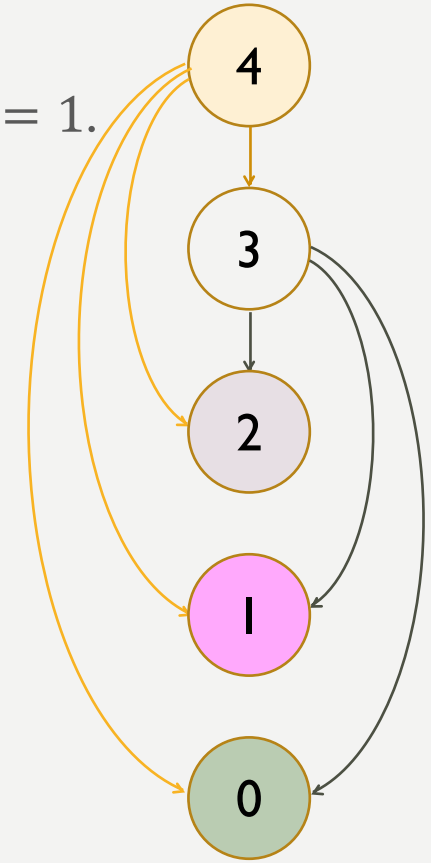
- A directed edge from the vertex for subproblem x to the vertex for subproblem y ***if*** determining an optimal solution for subproblem x involves directly considering an optimal solution for subproblem y.

- Consider the subproblem graph for the rod-cutting problem with $n = 4$.
  - The specific instance start with vertex 4.



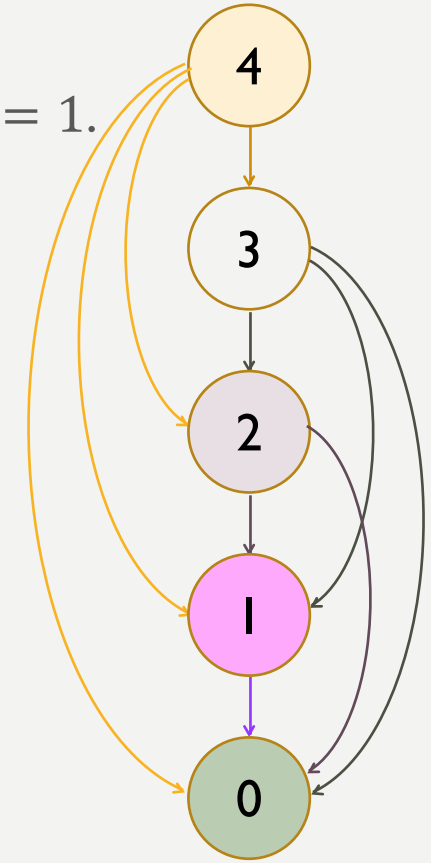
READING/DRAWING A SUBPROBLEM GRAPH

- Continue to solve the unresolved (sub)problems are when $n = 3, n = 2,$ and $n = 1$.
  - For $n = 3$, we have the following cases.
    - Cut at 1 inch. Subproblem $n = 2$ rises. (Overlap)
    - Cut at 2 inch. Subproblem $n = 1$ rises. (Overlap)
    - No cutting. Subproblem $n = 0$ rises. (Overlap)



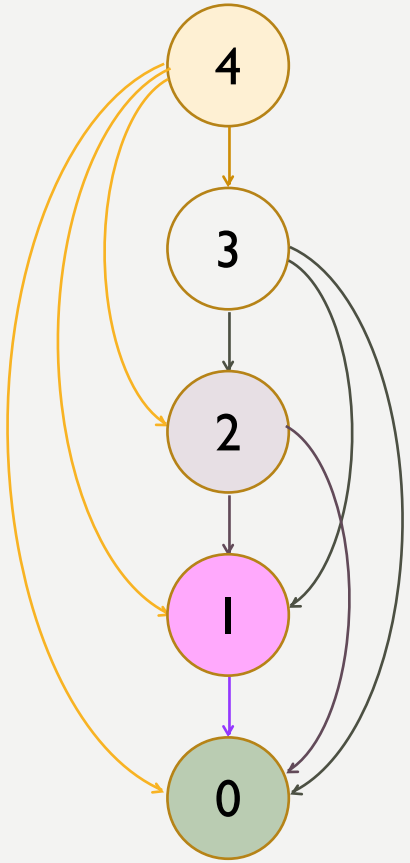
READING/DRAWING A SUBPROBLEM GRAPH

- Continue to solve the unresolved (sub)problems are when $n = 3, n = 2,$ and $n = 1$.
  - For $n = 2$, we have the following cases.
    - Cut at 1 inch, creating subproblem $n = 1$. (Overlap)
    - No cutting. Subproblem $n = 0$ rises. (Overlap)
  - For $n = 1$, we have the following case.
    - No cutting. Subproblem $n = 0$ rises. (Overlap)



USING THE SUBPROBLEM GRAPH

- The ***subproblem graph*** can be used to discover overlapping subproblems in a problem.
- As we draw the subproblem graph, we can
  - ***collect*** our thoughts
  - realize that we need a ***memo*** to record the subproblems that have been solved
  - come closer to ***deriving the recurrence relation*** between the original problem and the subproblem



RECONSTRUCTING AN OPTIMAL SOLUTION

- Keep in mind ***the principle of algorithm design***
  - The algorithms are **meant to be implemented by computer programs** to solve a real-world problem.
    - The optimal value might not be the only thing we are looking for.
    - We also need the choices that lead up to the optimal value.

MEMOIZAITON

- Using memoization can **achieve the efficiency** of the bottom-up dynamic programming approach **while maintaining a top-down strategy**.
  - ***Memoize*** the natural, but inefficient, recursive algorithm.

NEXT UP
MATRIX-CHAIN PARENTHESIZATION

REFERENCE

- Screenshots are taken from the textbook.