CS207 Design and Analysis of Algorithms

Sajith Gopalan

Indian Institute of Technology Guwahati

sajith@iitg.ac.in

February 14, 2022

Dynamic Programming

Dynamic Programming

- Particularly useful for optimization problems. Solution space. Constraints. Many feasible solutions, each of which satisfies the constraints, and has a value. We wish to find one feasible solution that minimises (maximizes) value.
- Steps:
  - Formulate the structure of an optimal solution
  - Recursively define the value of an optimal solution
  - Compute the value of an optimal solution, typically in a bottom-up fashion
  - Construct an optimal solution from computed information
- Works particularly when the following properties hold:
  - Optimal Substructure: an optimal solution to the problem contains within it optimal solutions to subproblems
  - Overlapping subproblems: recursive algorithm for the problem solves the same subproblems repeatedly

Rod Cutting Problem

- Imagine rods that are of a fixed diameter are made of a particular material
- Given is a sequence $p = \langle p_1, \ldots, p_n \rangle$, where p_i the market price of a rod of length i inches
- Required to find is r_n, the maximum money that can be made from a rod of length n inches
- Example: $p = \langle 1, 5, 5 \rangle$; required to find is r_3
- A rod of 3 inches has two cut points: at 1 inch and 2 inches
- At each each cut point, you may choose to cut (1) or not (0)
- 00: No cuts; a single piece of length 3; revenue: 5
- 01, 10: a single cut; two pieces of lengths 1 and 2 respectively; revenue: 6
- 11: two cuts; three pieces of unit length; revenue: 3
- $r_3 = 6$

Rod Cutting Problem

- Every cut solution can be decomposed into: $\langle$ the left most piece, the rest $\rangle$
- $p_i + r_{n-i}$ is the revenue from "cutting off i inches from the left end and then maximizing the revenue from the rest"
- Then r_n can be calculated using the following recurrence relation:

$$r_n = \left\{ \begin{array}{ll} 0 & \text{if } n = 0 \\ \max_{1 \leq i \leq n}\{p_i + r_{n-i}\} & \text{if } n > 1 \end{array} \right\}$$

Rod Cutting Problem

- If $T(n)$ is the time complexity of computing r_n using this recursive formulation,
- then
$$T(n) = \left\{ \begin{array}{ll} 1 & \text{if } n = 0 \\ 1 + \sum_{k=0}^{n-1} T(k) & \text{if } n > 1 \end{array} \right\}$$
- Claim: $T(n) = 2^n$
- Basis: $T(0) = 1 = 2^0$
- Hypothesis: $T(k) = 2^k$ for $k < n$
- Induction Step: $T(n) = 1 + \sum_{k=0}^{n-1} T(k) = 1 + \sum_{k=0}^{n-1} 2^k = 2^n$

Rod Cutting Problem: Iterative Algorithm

$r[0 \ldots n]$ and $s[0 \ldots n]$ are global arrays

Algorithm 1 RodCut(p, n)

```
r[0] = 0
for i = 1 to n do
    a = -∞
    for j = 1 to i do
        if (a < p[j] + r[i - j]) then
            a = p[j] + r[i - j]; s[i] = j;
        end if
    end for
    r[i] = a
end for
```

Rod Cutting Problem: Print Solution

Algorithm 2 PrintSolutionRodCut(p, n)

```
RodCut(p, n);
while n > 0 do
    Print s[n];
    n = n − s[n];
end while
```

$s[n]$ is the length of the leftmost piece of the best solution for length n

Print it, and recurse with the remaining length