CS 312: Algorithms

More Dynamic Programming: Rod Cutting

Dan Sheldon

Mount Holyoke College

Last Compiled: November 5, 2018

Dynamic Programming Recipe

- Step 1: Devise simple recursive algorithm
  - Make *one decision* by trying all possibilities
  - Use a recursive solver to evaluate the value of each
  - Problem: it does redundant work, often exponential time
- Step 2: Write recurrence for optimal value
- Step 3: Design iterative algorithm

- Weighted interval scheduling: first decision has two options
- Rod-cutting: first decision has n options

Rod Cutting

- Formulate problem on board
- **Problem Input**:
  - Steel rod of length n, can be cut into integer lengths
  - Price $p(i)$ for a rod of length i
- **Goal**
  - Cut rods into lengths $i_1, \ldots, i_k$ such that $i_1 + i_2 + \ldots i_k = n$.
  - Maximize value $p(i_1) + p(i_2) + \ldots + p(i_n)$

First decision?

Choose length i of first piece, then recurse on smaller rod

Steps 1 and 2

Step 1: Recursive Algorithm.

```
CutRod(j)
  if j = 0 then return 0
  best = 0
  for i = 1 to j do
     val = p[i] + CutRod(j − i)
     best = max(best, val)
  end for
  return best
```

- Running time for CutRod(n)? $\Theta(2^n)$

Step 2: Recurrence

$$\text{OPT}(j) = \max_{1 \le i \le j} \{p_i + \text{OPT}(j - i)\}$$

$$\text{OPT}(0) = 0$$

From Recurrence to Algorithm

$$\text{OPT}(j) = \max_{1 \le i \le j} \{p_i + \text{OPT}(j - i)\}$$

$$\text{OPT}(0) = 0$$

What size memoization array M? What order to fill?

- $M[\cdot]$ accepts same "arguments" as $\text{OPT}(j) \rightarrow$ indices of unique subproblems. Range of values of j determines size of M. $M[0..n]$
- Fill M so RHS values are computed before LHS. Fill from 0 to n

Step 3: Iterative Algorithm

- Array $M[0..n]$ where $M[j]$ holds value of $\mathrm{OPT}(j)$. Fill from 0 to n.

```
CutRod-Iterative
  Initialize array M[0..n]
  Set M[0] = 0
  for j = 1 to n do
      best = 0
      for i = 1 to j do
          val = p[i] + M[j − i]
          best = max(best, val)
      end for
      Set M[j] = best
  end for
```

- Running time? $\Theta(n^2)$ Note: body of for loop identical to recursive algorithm, directly implements recurrence

Epilogue: Recover Optimal Solution

Trace back from end and reconstruct choices that lead to optimal value

Run previous algorithm to fill in M array, but with the following modification: let first-cut[j] be the index i that leads to the largest value when computing $M[j]$.

```
cuts = {}
j = n                                  ▷ Remaining length
while j > 0 do
    j = j − first-cut[j]
    cuts = cuts ∪ {first-cut[j]}
end while
```