Dynamic Programming

Chapter 15
Dictionary Definition

- **Program (noun)** \ˈprō-ˌgram, -grəm\ 
  - a sequence of coded instructions that can be inserted into a mechanism (such as a computer)

- **Programming (noun)** \ˈprō-ˌgra-miŋ, -grə-\  
  - a plan of action to accomplish a specified end

- **Synonyms**
  - progressing, plan out, arrange
Rod Cutting Problem
Rod Cutting Problem

- Given a rod of length $n$ inches and a table of prices $p_i$ for $i = 1, 2, \ldots, n$, determine the maximum revenue $r_n$ obtainable by cutting up the rod and selling the pieces.

- Naïve solution?
  - Try all possibilities. Exponential!
Recursive Solution

- Define $r_i$ as the maximum possible revenue you can get for a rod of length $i$
- We can express $r_i$ as follows

\[
 r_i = \begin{cases} 
 0 & i = 0 \\
 \max \left\{ p_i \max_{1 \leq j < i} \left( r_j + r_{i-j} \right) \right\} & i > 0 
\end{cases}
\]

- Final answer is $r_n$
def rodcut(i, p)
    return 0 if i == 0
    best_cut = p[i]
    for j in (1..i-1)
        value = rodcut(j, p) + rodcut(i-j, p)
        best_cut = [best_cut, value].max
    end
    return best_cut
end
Execution Recurrence Tree

$n_5$

$n_4$

$n_3$

$n_2$

$n_1$

$n_3$

$n_2$

$n_1$

$n_2$

$n_1$

$n_1$

$n_1$
Memoized Recursive Solution

```python
@best_cuts = []
def rodcut_memoized(i,p):
    return 0 if i == 0
    return @best_cuts[i] if @best_cuts[i]
    best_cut = p[i]
    for j in range(1,i):
        value = rodcut_memoized(j,p) +
                 rodcut_memoized(i-j,p)
        best_cut = [best_cut, value].max
    return @best_cuts[i] = best_cut
```
def rodcut_bottomup(n, p):
    best_cuts = []
    for i in (1..n):
        best_cuts[i] = p[i]
        for j in (1..i-1):
            value = best_cuts[j] + best_cuts[i-j]
            best_cuts[i] = [best_cuts[i], value].max
    end
    return best_cuts[n]
end
Optimal Substructure

- The problem is to find the optimal cut for a rod of length $n$
- Assuming that we cut $n$ at position $i$
- This leads to two subproblems, cutting two rods of lengths $i$ and $n - i$
- The optimal solution when $n$ is cut at position $i$ must include the optimal cuts of rods of lengths $i$ and $n - i$
- Proof by contradiction
Dynamic Programming

- You have a big problem
- You can break it down into smaller subproblems
- You don’t know the best way to split it
  - So, you try all possibilities
- Identify similar subproblems that are solved many times
- Devise a better (polynomial) algorithm
Matrix Chain Multiplication
Example

\[ \text{Cost}((A_1 A_2)A_3) = 10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 7,500 \]

\[ \text{Cost}(A_1(A_2 A_3)) = 10 \cdot 100 \cdot 50 + 100 \cdot 5 \cdot 50 = 75,000 \]
Problem

- Given a sequence of (not necessarily square) matrices \( \langle A_1, A_2, \ldots, A_n \rangle \) where the dimensions of matrix \( A_i \) is \( p_{i-1} \times p_i \). We want to find the order of matrix multiplication operations to compute the product \( A_1 \cdot A_2 \cdot \ldots \cdot A_n \) while minimizing the number of scalar multiplication operations.

- **Hint:** Number of scalar multiplication operations to compute \( X \cdot Y \) with dimensions \( p \times q \) and \( q \times r \), respectively, is \( p \cdot q \cdot r \)
Recursive Implementation

Matrix-Chain-Multiplication(p, s, e)

if e <= s then return 0

min_cost = ∞

for i = s to e – 1


cost = Matrix-Chain-Multiplication(p, s, i) +
Matrix-Chain-Multiplication(p, i+1, e) +
p[s-1] * p[i+1] * p[e]

min_cost = Min(cost, min_cost)

return min_cost
With Memoization

- Matrix-Chain-Multiplication\(p, s, e\)
  - if \(e < s\) then return 0
  - return table\(s,e\) if exists
  - \(\text{min\_cost} = \infty\)
  - for \(i = s\) to \(e - 1\)
    - cost = Matrix-Chain-Multiplication\(p, s, i\) + Matrix-Chain-Multiplication\(p, i+1, e\) + \(p[s-1] * p[i+1] * p[e]\)
    - \(\text{min\_cost} = \text{Min}(\text{cost}, \text{min\_cost})\)
  - table\(s,e\) = \(\text{min\_cost}\)
  - return \(\text{min\_cost}\)
Recurrence Relation

- $m[i, j]$: Minimum cost for multiplying matrices $i$ through $j$; $1 \leq i \leq j \leq n$

- $m[i, j] = \begin{cases} 
0 & i = j \\
\min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & i < j 
\end{cases}$
Bottom-up Approach

Final answer

Cost[$A_i \ldots A_j$]

$A_1 \ A_2 \ \ldots \ A_n$