# Algorithm Design, Analysis & Complexity

## Lecture 4 - Dynamic Programming

Koushik Pal

University of Toronto

May 25, 2021

# Five steps of Dynamic Programming

1. Defining the optimal substructure / recursive structure
2. Array definition for memoization
3. Defining the recurrence relation in terms of the array
4. Bottom-up iterative algorithm
5. Find one optimum solution using the array values.

# 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 obtainable by cutting up the rod and selling the pieces.*

Optimal substructure: Once we make the first cut, we may consider the two pieces as independent instances of the rod cutting problem. The overall optimal solution incorporates optimal solutions to the two subproblems.

# Example

## Example

Let $n = 5$, and the table of prices be

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $P(i)$ | 5 | 7 | 10 | 13 | 17 |

If we don't cut the rod, we get a value of $17$.

If we cut the rod at index 1, we get two pieces of lengths $1$ and $4$, giving a total value of $5 + 13 = 18$.

If we cut the rod at indices 1 and 3, we get three pieces of lengths $1$, $2$ and $2$, giving a total value of $5 + 7 + 7 = 19$.

If we cut the rod at indices 1 and 4, we get three pieces of lengths $1$, $3$ and $1$, giving a total value of $5 + 10 + 5 = 20$.

If we cut the rod at indices $1$, $2$ and $4$, we get four pieces of lengths $1$, $1$, $2$ and $1$, giving a total value of $5 + 5 + 7 + 5 = 22$.

If we cut the rod at indices $1$, $2$, $3$ and $4$, we get five pieces of length $1$ each, giving a total value of $5 + 5 + 5 + 5 + 5 = 25$.

# Brute Force Solution

We can cut up a rod of length $n$ in $2^{n-1}$ different ways: for each $i = 1, 2, \ldots, n-1$, we have two options — to cut or not to cut — at distance $i$ from the left end.

Compare the value obtained for each option, and find the max.

Complexity: $\Theta(2^n)$.

# DP Solution

Step I. Observe that every decomposition can be viewed as a first piece plus decomposition of the remainder.
We, therefore, obtain the following recurrence relation:

$$OPT_j = \begin{cases} \max_{1 \le i \le j}\{P_i + OPT_{j-i}\} & \text{if } j > 0 \\ 0 & \text{if } j = 0. \end{cases}$$

Step II. Let $M$ be an array of length $n$ defined as follows: $M[j]$ stands for the maximum revenue generated on a rod of length $j$.

Step III. Write the recurrence in terms of the array $M$:

$$M[j] = \begin{cases} \max_{1 \le i \le j}\{P_i + M[j-i]\} & \text{if } j > 0 \\ 0 & \text{if } j = 0. \end{cases}$$

# DP Solution

Step IV. Write an iterative bottom-up algorithm to compute $M$.

```
procedure CutRod(P, n)
    Define M[0...n]
    M[0] = 0
    for j = 1 to n do
        q := −∞
        for i = 1 to j do
            if q < P[i] + M[j−i] then
                q = P[i] + M[j−i]
        M[j] = q
    return M[n]
```

**Complexity:** $\Theta(n^2)$.

# DP Solution

Step V. Produce an optimal solution (at which indices to cut the rod).

```
procedure CutRodExtended(P, n)
    Define M[0...n] and S[0...n]
    M[0] = 0
    for j = 1 to n do
        q := −∞
        for i = 1 to j do
            if q < P[i] + M[j−i] then
                q = P[i] + M[j−i]
                S[j] = i
        M[j] = q
    return M[n] and S
```

**Complexity:** $\Theta(n^2)$.

# DP Solution

Step V. Produce an optimal solution (continued...)

```
procedure PrintCutRodSolution(S, n)
    while n > 0 do
        print S[n]
        n = n − S[n]
```

**Complexity:** $\Theta(n)$.

# Matrix Chain Multiplication

## Problem

*Given a list of matrices $A_1, A_2, \ldots, A_n$, compute the product $A_1 A_2 \cdots A_n$ using the least number of computations.*

Optimal substructure: For $n \geq 2$, a fully parenthesized matrix product is the product of two fully parenthesized matrix sub-products, and the spit between the two sub-products may occur between $k^{th}$ and $(k+1)^{st}$ matrices for any $1 \leq k \leq n-1$.

# Properties of Matrix Multiplication

- Matrix multiplication is **NOT** commutative: $A_1 A_2 \neq A_2 A_1$.
- Matrix multiplication is associative: $A_1(A_2 A_3) = (A_1 A_2) A_3$.
- Let $\dim(A_1) = r_1 \times c_1$ and $\dim(A_2) = r_2 \times c_2$.
  Then $A_1 A_2$ is defined **if and only if** $c_1 = r_2$.
  In that case, $\dim(A_1 A_2) = r_1 \times c_2$.
  Also, total number of multiplications needed $= r_1 c_1 c_2$.

# Why order of multiplication matters?

Let $\dim(A_1) = c_0 \times c_1$, $\dim(A_2) = c_1 \times c_2$ and $\dim(A_3) = c_2 \times c_3$.

Suppose also, $c_0 = 2, c_1 = 20, c_2 = 5, c_3 = 50$.

We want to compute the product $A_1 A_2 A_3$ in two different ways:

- $A_1(A_2 A_3)$: number of multiplications needed

$$\begin{aligned} &= c_1 c_2 c_3 + c_0 c_1 c_3 \\ &= 5000 + 2000 \\ &= 7000 \end{aligned}$$

- $(A_1 A_2) A_3$: number of multiplications needed

$$\begin{aligned} &= c_0 c_1 c_2 + c_0 c_2 c_3 \\ &= 200 + 500 \\ &= 700 \end{aligned}$$

Thus, $(A_1 A_2) A_3$ takes much less computation than $A_1(A_2 A_3)$.

# Brute Force Solution

Let $P(n)$ denote the number of alternative parenthesizations possible.

Clearly, $P(1) = 1$.

When $n \geq 2$, recall the optimal substructure: a fully parenthesized matrix product is the product of two fully parenthesized matrix sub-products, and the spit between the two sub-products may occur between $k^{th}$ and $(k+1)^{st}$ matrices for any $1 \leq k \leq n-1$.

Therefore, $P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$.

Exercise: Show that the solution to this recurrence is $\Omega(2^n)$.

Finally, compare the number of computations for each option, and find the min.

Complexity: $\Omega(2^n)$.

# DP Solution

Step I. Any decomposition of $A_i \dots A_j$ can be written as

$$OPT_{i,k} + OPT_{k+1,j} + c_{i-1}c_kc_j,$$

where $c_i$ is the number of columns of matrix $A_i$, with $c_0$ as the number of rows of $A_1$.

So, an optimal parenthesization of $A_i \dots A_j$ would be given by

$$OPT_{i,j} = \begin{cases} \min_{i \le k < j}\{OPT_{i,k} + OPT_{k+1,j} + c_{i-1}c_kc_j\} & \text{if } i < j \\ 0 & \text{if } i = j. \end{cases}$$

Step II. Define 2-D array $M$ as follows: $M[i,j]$ represents the minimum number of computations needed to compute the product $A_i \dots A_j$.

Step III. Write the recurrence in terms of the array $M$:

$$M[i,j] = \begin{cases} \min_{i \le k < j}\{M[i,k] + M[k+1,j] + c_{i-1}c_kc_j\} & \text{if } i < j \\ 0 & \text{if } i = j. \end{cases}$$

# DP Solution

## Step IV.

**procedure** $\textsc{MatrixChainOrder}(c_0, \ldots, c_n)$
  Define 2-D arrays $M$ and $S$ of dimension $n \times n$
  **for** $i = 1$ to $n$ **do**
    $M[i, i] = 0$
  **for** $\ell = 2$ to $n$ **do**
    **for** $i = 1$ to $n - \ell + 1$ **do**
      $j := i + \ell - 1$
      $M[i, j] = \infty$
      **for** $k = i$ to $j - 1$ **do**
        $q := M[i, k] + M[k+1, j] + c_{i-1} c_k c_j$
        **if** $q < M[i, j]$ **then**
          $M[i, j] = q$
          $S[i, j] = k$
  **return** $M[1, n]$ and $S$

**Complexity:** $\Theta(n^3)$.

# DP Solution

## Step V.

**procedure** PrintOptimalParenthesisWrapper($S$)
    **return** PrintOptimalParenthesis($S, 1, n$)

**procedure** PrintOptimalParenthesis($S, i, j$)
    **if** $i == j$ **then**
        print "$A_i$"
    **else**
        print "("
        PrintOptimalParenthesis($S, i, S[i, j]$)
        PrintOptimalParenthesis($S, S[i, j] + 1, j$)
        print ")"

**Complexity:** $\Theta(n)$.

# Subset Sum

## Problem

*Given $n$ items $\{1, \ldots, n\}$ each with a nonnegative weight $w_i$, and a bound $W$, find a subset $S$ of the items so that $\sum_{i \in S} w_i \leq W$ and, subject to this restriction, $\sum_{i \in S} w_i$ is as large as possible.*

## Problem (More generalized Knapsack problem)

*Given $n$ items $\{1, \ldots, n\}$ each with a nonnegative weight $w_i$ and a nonnegative value $v_i$, and a bound $W$, find a subset $S$ of the items so that $\sum_{i \in S} w_i \leq W$ and, subject to this restriction, $\sum_{i \in S} v_i$ is as large as possible.*

# DP Solution

Step I. Let $\mathcal{O}$ be an optimal solution and $OPT_{i,w}$ denote the best possible solution using a subset of items from $\{1, \ldots, i\}$ and bound $w$.
If $i \notin \mathcal{O}$, then $OPT_{i,w} = OPT_{i-1,w}$.
If $i \in \mathcal{O}$, then $OPT_{i,w} = w_i + OPT_{i-1,w-w_i}$.
Finally,

$$OPT_{i,w} = \begin{cases} \max\{OPT_{i-1,w},\ w_i + OPT_{i-1,w-w_i}\} & \text{if } w_i \leq w \\ OPT_{i-1,w} & \text{if } w_i > w \\ 0 & \text{if } i = 0. \end{cases}$$

Step II. Define 2-D array $M$ as follows: $M[i,w]$ denotes $OPT_{i,w}$.

Step III. Write the recurrence in terms of the array $M$:

$$M[i,w] = \begin{cases} \max\{M[i-1,w],\ w_i + M[i-1,w-w_i]\} & \text{if } w_i \leq w \\ M[i-1,w] & \text{if } w_i > w \\ 0 & \text{if } i = 0. \end{cases}$$

# DP Solution

## Step IV.

**procedure** $\textsc{SubsetSum}(n, w_1, w_2, \ldots, w_n, W)$
    Define 2-D array $M[0 \ldots n, 0 \ldots W]$
    **for** $w = 0$ to $W$ **do**
        $M[0, w] = 0$
    **for** $i = 1$ to $n$ **do**
        **for** $w = 0$ to $W$ **do**
            **if** $w < w_i$ **then**
                $M[i, w] = M[i-1, w]$
            **else**
                $M[i, w] = \max\{M[i-1, w],\ w_i + M[i-1, w-w_i]\}$
    **return** $M[n, W]$

**Complexity:** $\Theta(nW)$ (pseudo-polynomial time complexity).

**Step V.** Find an optimal solution. (Exercise)