Warm up

How many arithmetic operations are required to multiply a $n \times m$ Matrix with a $m \times p$ Matrix?

(don’t overthink this)
How many arithmetic operations are required to multiply a $n \times m$ Matrix with a $m \times p$ Matrix?

- $m$ multiplications and additions per element
- $n \cdot p$ elements to compute
- Total cost: $m \cdot n \cdot p$ 

$n \cdot p(m + m-1)$
Today’s Keywords

• Dynamic Programming
• Matrix Chaining
• Seam Carving
• Longest Common Subsequence
CLRS Readings

• Chapter 15
Administrativa

• HW5 out by tomorrow morning
  – Due March 27 at 11pm
  – Seam Carving!
  – Dynamic Programming (implementation)
  – Java or Python

• Midterm
  – Grading underway! Should be returned tomorrow

• HW4 grading in-progress
Dynamic Programming

• Requires **Optimal Substructure**
  – Solution to larger problem contains the solutions to smaller ones

• Idea:
  1. Identify recursive structure of the problem
     • What is the “last thing” done?
  2. Select a good order for solving subproblems
     • “Top Down”: Solve each recursively
     • “Bottom Up”: Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory
Generic Top-Down Dynamic Programming Soln

mem = {}
def myDPalgo(problem):
    if mem[problem] not blank:
        return mem[problem]
    if baseCase(problem):
        solution = solve(problem)
        mem[problem] = solution
        return solution
    for subproblem of problem:
        subsolutions.append(myDPalgo(subproblem))
    solution = OptimalSubstructure(subsolutions)
    mem[problem] = solution
    return solution
Matrix Chaining

• Given a sequence of Matrices \((M_1, ..., M_n)\), what is the most efficient way to multiply them?
\[ c_1 = r_2 \]
\[ c_2 = r_3 \]

Order Matters!

\[
(\color{yellow}{M_1} \times \color{red}{M_2}) \times \color{blue}{M_3}
\]

- uses \((c_1 \cdot r_1 \cdot c_2) + c_2 \cdot r_1 \cdot c_3\) operations
Order Matters!

\[ c_1 = r_2 \]
\[ c_2 = r_3 \]

\[ M_1 \times (M_2 \times M_3) \]

- uses \( c_1 \cdot r_1 \cdot c_3 + (c_2 \cdot r_2 \cdot c_3) \) operations
Order Matters!

- $(M_1 \times M_2) \times M_3$
  - uses $(c_1 \cdot r_1 \cdot c_2) + c_2 \cdot r_1 \cdot c_3$ operations
  - $(10 \cdot 7 \cdot 20) + 20 \cdot 7 \cdot 8 = 2520$
- $M_1 \times (M_2 \times M_3)$
  - uses $c_1 \cdot r_1 \cdot c_3 + (c_2 \cdot r_2 \cdot c_3)$ operations
  - $10 \cdot 7 \cdot 8 + (20 \cdot 10 \cdot 8) = 2160$

$c_1 = r_2$
$c_2 = r_3$

$M_1 = 7 \times 10$
$M_2 = 10 \times 20$
$M_3 = 20 \times 8$
$c_1 = 10$
$c_2 = 20$
$c_3 = 8$
$r_1 = 7$
$r_2 = 10$
$r_3 = 20$
Dynamic Programming

• Requires **Optimal Substructure**
  – Solution to larger problem contains the solutions to smaller ones

• **Idea:**

  1. **Identify recursive structure of the problem**
     • What is the “last thing” done?

  2. Select a good order for solving subproblems
     • “Top Down”: Solve each recursively
     • “Bottom Up”: Iteratively solve smallest to largest

  3. Save solution to each subproblem in memory
1. Identify the Recursive Structure of the Problem

$$Best(1,n) = \text{cheapest way to multiply together } M_1 \text{ through } M_n$$

$$\begin{align*}
Best(1,4) &= \min \\
Best(2,4) + r_1 r_2 c_4 &= \min \\
\end{align*}$$

Diagram: 
- $M_1$ with $c_1$ and $r_1$ multiplies with $M_2$ with $c_2$ and $r_2$.
- $M_2$ with $c_2$ multiplies with $M_3$ with $c_3$ and $r_3$.
- $M_3$ with $c_3$ multiplies with $M_4$ with $c_4$ and $r_4$.

$M_4$ is the cheapest way to multiply $M_1$ through $M_n$. 

$$\text{min}$$
1. Identify the Recursive Structure of the Problem

\( Best(1, n) = \) cheapest way to multiply together \( M_1 \) through \( M_n \)

\( Best(1, 4) = \min \left\{ \begin{array}{l}
Best(2, 4) + r_1 r_2 c_4 \\
Best(1, 2) + Best(3, 4) + r_1 r_3 c_4
\end{array} \right. \)
1. Identify the Recursive Structure of the Problem

\[ Best(1, n) = \text{cheapest way to multiply together } M_1 \text{ through } M_n \]

\[ Best(1, 4) = \min \begin{cases} 
Best(2, 4) + r_1 r_2 c_4 \\
Best(1, 2) + Best(3, 4) + r_1 r_3 c_4 \\
Best(1, 3) + r_1 r_4 c_4 
\end{cases} \]
1. Identify the Recursive Structure of the Problem

- In general:

\[ \text{Best}(i, j) = \text{cheapest way to multiply together } M_i \text{ through } M_j \]

\[
\text{Best}(i, j) = \min_{k=i}^{j-1} \left( \text{Best}(i, k) + \text{Best}(k + 1, j) + r_i r_{k+1} c_j \right)
\]

\[ \text{Best}(i, i) = 0 \]

\[ \text{Best}(1, n) = \min \left[ \begin{array}{l}
\text{Best}(2, n) + r_1 r_2 c_n \\
\text{Best}(1,2) + \text{Best}(3, n) + r_1 r_3 c_n \\
\text{Best}(1,3) + \text{Best}(4, n) + r_1 r_4 c_n \\
\text{Best}(1,4) + \text{Best}(5, n) + r_1 r_5 c_n \\
\vdots \\
\text{Best}(1, n) + r_{n-1} c_{n-1} \\
\end{array} \right] \]
Dynamic Programming

• Requires **Optimal Substructure**
  – Solution to larger problem contains the solutions to smaller ones

• Idea:
  1. Identify recursive structure of the problem
     • What is the “last thing” done?
  2. Select a good order for solving subproblems
     • “Top Down”: Solve each recursively
     • “Bottom Up”: Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory
1. Identify the Recursive Structure of the Problem

• In general:

\[ Best(i,j) = \text{cheapest way to multiply together } M_i \text{ through } M_j \]

\[ Best(i,j) = \min_{k=i}^{j-1} (Best(i,k) + Best(k+1,j) + r_ir_{k+1}c_j) \]

\[ Best(i,i) = 0 \]

\[ Best(1,n) = \min \]

\[ \hspace{1cm} \begin{align*}
    Best(2,n) &+ r_1r_2c_n \\
    Best(1,2) &+ Best(3,n) + r_1r_3c_n \\
    Best(1,3) &+ Best(4,n) + r_1r_4c_n \\
    Best(1,4) &+ Best(5,n) + r_1r_5c_n \\
    \vdots \\
    Best(1,n-1) &+ r_1r_nc_n
\end{align*} \]
Dynamic Programming

• Requires **Optimal Substructure**
  – Solution to larger problem contains the solutions to smaller ones

• Idea:
  1. Identify recursive structure of the problem
    • What is the “last thing” done?
  2. Select a good order for solving subproblems
    • “Top Down”: Solve each recursively
      • “Bottom Up”: Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory
2. Select a good order for solving subproblems

$$Best(i, j) = \min_{k=i}^{j-1} \left( Best(i, k) + Best(k + 1, j) + r_i r_{k+1} c_j \right)$$

$$Best(i, i) = 0$$
2. Select a good order for solving subproblems

\[
Best(i, j) = \min_{k=i}^{j-1} \left( Best(i, k) + Best(k + 1, j) + r_i r_{k+1} c_j \right)
\]

\[
Best(i, i) = 0
\]

\[
Best(1,2) = \min \left\{ Best(1,1) + Best(2,2) + r_1 r_2 c_2 \right\}
\]

\[
\begin{array}{ccccccc}
& 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 15750 & & & & \\
2 & 0 & & & & & \\
3 & 0 & & & & & \\
4 & 0 & & & & & \\
5 & & & & & & \\
6 & & & & & & \\
\end{array}
\]
2. Select a good order for solving subproblems

\[ Best(i, j) = \min_{k=i}^{j-1} \left( Best(i, k) + Best(k+1, j) + r_i r_{k+1} c_j \right) \]

\[ Best(i, i) = 0 \]

\[ Best(2,3) = \min \left\{ Best(2,2) + Best(3,3) + r_2 r_3 c_3 \right\} \]
2. Select a good order for solving subproblems

\[ \text{Best}(i, j) = \min_{k=i}^{j-1} \left( \text{Best}(i, k) + \text{Best}(k + 1, j) + r_k r_{k+1} c_j \right) \]

\[ \text{Best}(i, i) = 0 \]
2. Select a good order for solving subproblems

\[
Best(i, j) = \min_{k=i}^{j-1} (Best(i, k) + Best(k + 1, j) + r_i r_{k+1} c_j)
\]

\[
Best(i, i) = 0
\]

\[
\begin{align*}
    r_1 r_2 c_3 &= 30 \cdot 35 \cdot 5 = 5250 \\
    r_1 r_3 c_3 &= 30 \cdot 15 \cdot 5 = 2250
\end{align*}
\]

\[
Best(1,3) = \min \left\{ \begin{array}{l}
    0 \\
    15750
\end{array} \right\}
\]

\[
\begin{array}{ccccccc}
    & 0 & 15750 & 7875 & & & \\
\hline
1 & 0 & 15750 & 7875 & 2625 & & \\
2 & 0 & 2625 & 750 & 0 & 1000 & \\
3 & 0 & 750 & 0 & 1000 & 5000 & \\
4 & & & & & 0 & 5000 \\
5 & & & & & 0 & 5000 \\
6 & & & & & 0 & 5000 \\
\end{array}
\]
2. Select a good order for solving subproblems

\[ \text{Best}(i, j) = \min_{k=i}^{j-1} \left( \text{Best}(i, k) + \text{Best}(k + 1, j) + \frac{r_ir_{k+1}c_j}{r_i} \right) \]

\[ \text{Best}(i, i) = 0 \]

To find \( \text{Best}(i, j) \): Need all preceding terms of row \( i \) and column \( j \)

Conclusion: solve in order of diagonal

\[
\begin{array}{cccccccc}
& & & & & & & \\
& & & & & & & \\
& & & & & & 1 \text{st} & \\
0 & & 15750 & & 7875 & & & \\
& & & & & & & \\
& & & 0 & & 2625 & & \\
0 & & & & & & & \\
& & & 0 & & & 750 & \\
& & 0 & & & & & \\
& & 0 & & & 1000 & & \\
& & & 0 & & 5000 & & \\
& & & & 0 & & & \\
& & & & & 0 & & \\
\end{array}
\]
## Longest Common Subsequence

\[ \text{Best}(i, j) = \min_{k=i}^{j-1} \left( \text{Best}(i, k) + \text{Best}(k + 1, j) + r_i r_{k+1} c_j \right) \]

\[ \text{Best}(i, i) = 0 \]

\[
\begin{array}{cccccc}
  & 1 & 2 & 3 & 4 & 5 & 6 \\
0 & 0 & 15750 & 7875 & 9375 & 11875 & 15125 \\
1 & & 0 & 2625 & 4375 & 7125 & 10500 \\
2 & & & 0 & 750 & 2500 & 5375 \\
3 & & & & 0 & 5000 & 3500 \\
4 & & & & & 0 & 5000 \\
5 & & & & & & 0 \\
6 & & & & & & \\
\end{array}
\]

\[ \text{Best}(1,6) = \min \left\{ \begin{array}{c}
\text{Best}(1,1) + \text{Best}(2,6) + r_1 r_2 c_6 \\
\text{Best}(1,2) + \text{Best}(3,6) + r_1 r_3 c_6 \\
\text{Best}(1,3) + \text{Best}(4,6) + r_1 r_4 c_6 \\
\text{Best}(1,4) + \text{Best}(5,6) + r_1 r_5 c_6 \\
\text{Best}(1,5) + \text{Best}(6,6) + r_1 r_6 c_6 \\
\end{array} \right\} \]
Run Time

1. Initialize $Best[i, i]$ to be all 0s

2. Starting at the main diagonal, working to the upper-right, fill in each cell using:
   1. $Best[i, i] = 0$
   2. $Best[i, j] = \min_{k=i}^{j-1} (Best(i, k) + Best(k + 1, j) + r_i r_{k+1} c_j)$

   $\Theta(n^2)$ cells in the Array

   $\Theta(n)$ options for each cell

$\Theta(n^3)$ overall run time
Backtrack to find the best order

“remember” which choice of $k$ was the minimum at each cell

$$\text{Best}(i, j) = \min_{k=i}^{j-1} \left( \text{Best}(i, k) + \text{Best}(k + 1, j) + r_ir_{k+1}c_j \right)$$

$\text{Best}(i, i) = 0$

$$\begin{array}{cccccc}
  & 1 & 2 & 3 & 4 & 5 & 6 \\
 0 & 0 & 15750 & 7875 & 9375 & 11875 & 15125 \\
 1 & 0 & 2625 & 4375 & 7125 & 10500 & \\
 2 & 0 & 750 & 2500 & 5375 & & \\
 3 & & & & & & \\
 4 & & & & & & \\
 5 & & & & & & \\
 6 & & & & & & \\
\end{array}$$

$\text{Best}(1,6) = \min_	ext{red} \begin{array}{cccccc}
  & 1 & 2 & 3 & 4 & 5 & 6 \\
 0 & 0 & 15750 & 7875 & 9375 & 11875 & 15125 \\
 1 & 0 & 2625 & 4375 & 7125 & 10500 & \\
 2 & 0 & 750 & 2500 & 5375 & & \\
 3 & & & & & & \\
 4 & & & & & & \\
 5 & & & & & & \\
 6 & & & & & & \\
\end{array}$

$\text{Best}(1,6) = \min_	ext{pink} \begin{array}{cccccc}
  & 1 & 2 & 3 & 4 & 5 & 6 \\
 0 & 0 & 15750 & 7875 & 9375 & 11875 & 15125 \\
 1 & 0 & 2625 & 4375 & 7125 & 10500 & \\
 2 & 0 & 750 & 2500 & 5375 & & \\
 3 & & & & & & \\
 4 & & & & & & \\
 5 & & & & & & \\
 6 & & & & & & \\
\end{array}$
Dynamic Programming

• Requires **Optimal Substructure**
  – Solution to larger problem contains the solutions to smaller ones

• Idea:
  1. Identify recursive structure of the problem
     • What is the “last thing” done?
  2. Select a good order for solving subproblems
     • “Top Down”: Solve each recursively
     • “Bottom Up”: Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory
In Season 9 Episode 7 “The Slicer” of the hit 90s TV show *Seinfeld*, George discovers that, years prior, he had a heated argument with his new boss, Mr. Kruger. This argument ended in George throwing Mr. Kruger’s boombox into the ocean. How did George make this discovery?

https://www.youtube.com/watch?v=pSB3HdmLcY4
Seam Carving

• Method for image resizing that doesn’t scale/crop the image
Seam Carving

• Method for image resizing that doesn’t scale/crop the image
Seam Carving

- Method for image resizing that doesn’t scale/crop the image

Cropped  Scaled  Carved
Cropping

• Removes a “block” of pixels
Scaling

• Removes “stripes” of pixels
Seam Carving

• Removes “least energy seam” of pixels

• [http://rsizr.com/](http://rsizr.com/)
Seattle Skyline
Energy of a Seam

• Sum of the energies of each pixel
  – \( e(p) = \text{energy of pixel } p \)

• Many choices
  – E.g.: change of gradient (how much the color of this pixel differs from its neighbors)
  – Particular choice doesn’t matter, we use it as a “black box”
Identify Recursive Structure

Let \( S(i, j) \) = least energy seam from the bottom of the image up to pixel \( p_{i,j} \)
Finding the Least Energy Seam

Want the least energy seam going from bottom to top, so delete:

\[
\min_{k=1}^{m}(S(n, k))
\]
Computing $S(n, k)$

Assume we know the least energy seams for all of row $n - 1$
(i.e. we know $S(n - 1, \ell)$ for all $\ell$)

Known through $n - 1$
Computing $S(n, k)$

Assume we know the least energy seams for all of row $n - 1$ (i.e. we know $S(n - 1, \ell)$ for all $\ell$)
Repeated Seam Removal

Only need to update pixels dependent on the removed seam

$2n$ pixels change

$\Theta(2n)$ time to update pixels

$\Theta(n + m)$ time to find min+backtrack