CS4102 Algorithms

Spring 2023

Movie Time!

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

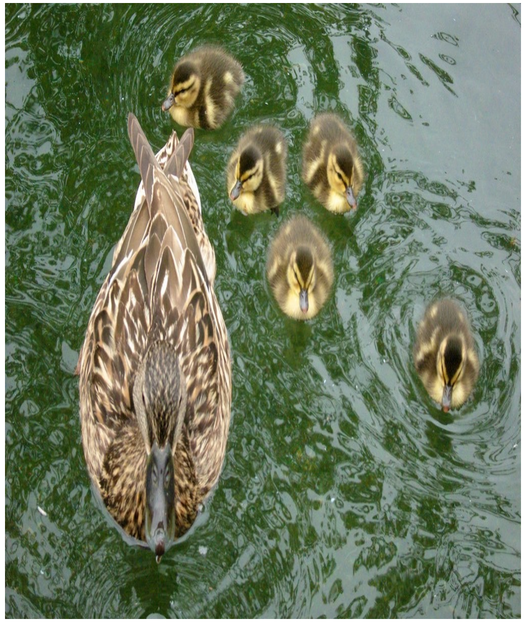
- Method for image resizing that doesn't scale/crop the image

Seam Carving

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

Cropped

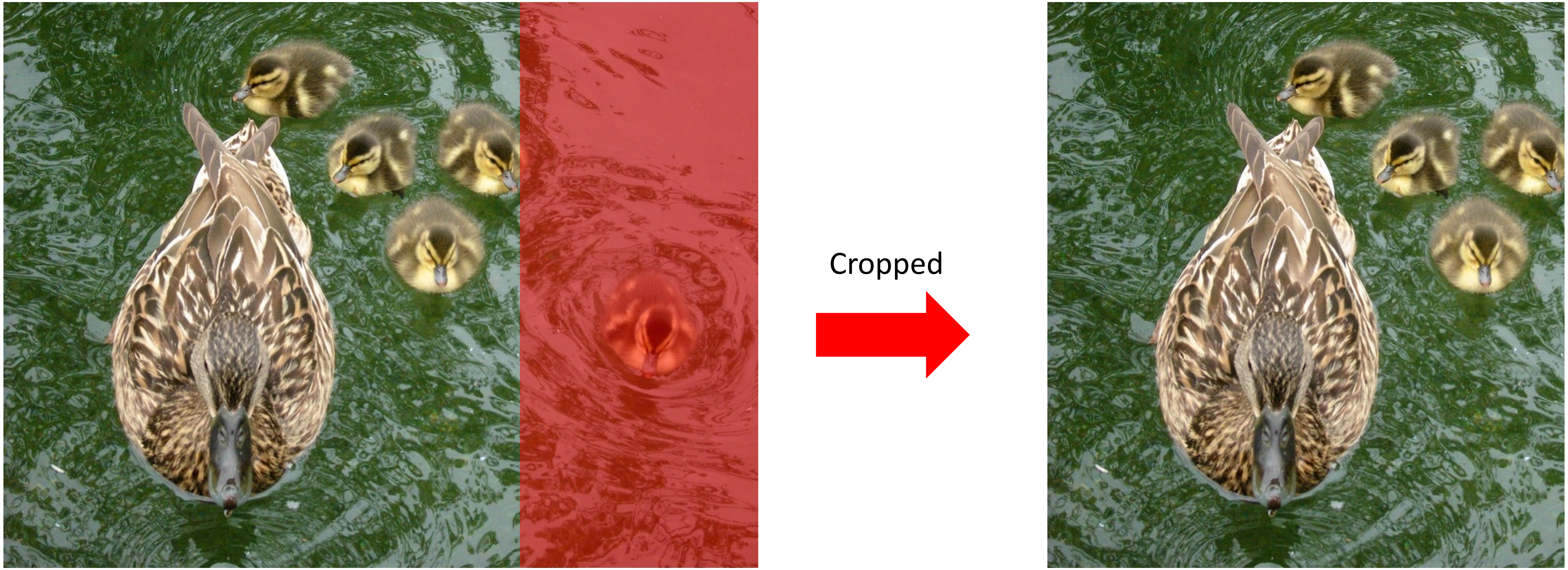
Scaled



Carved

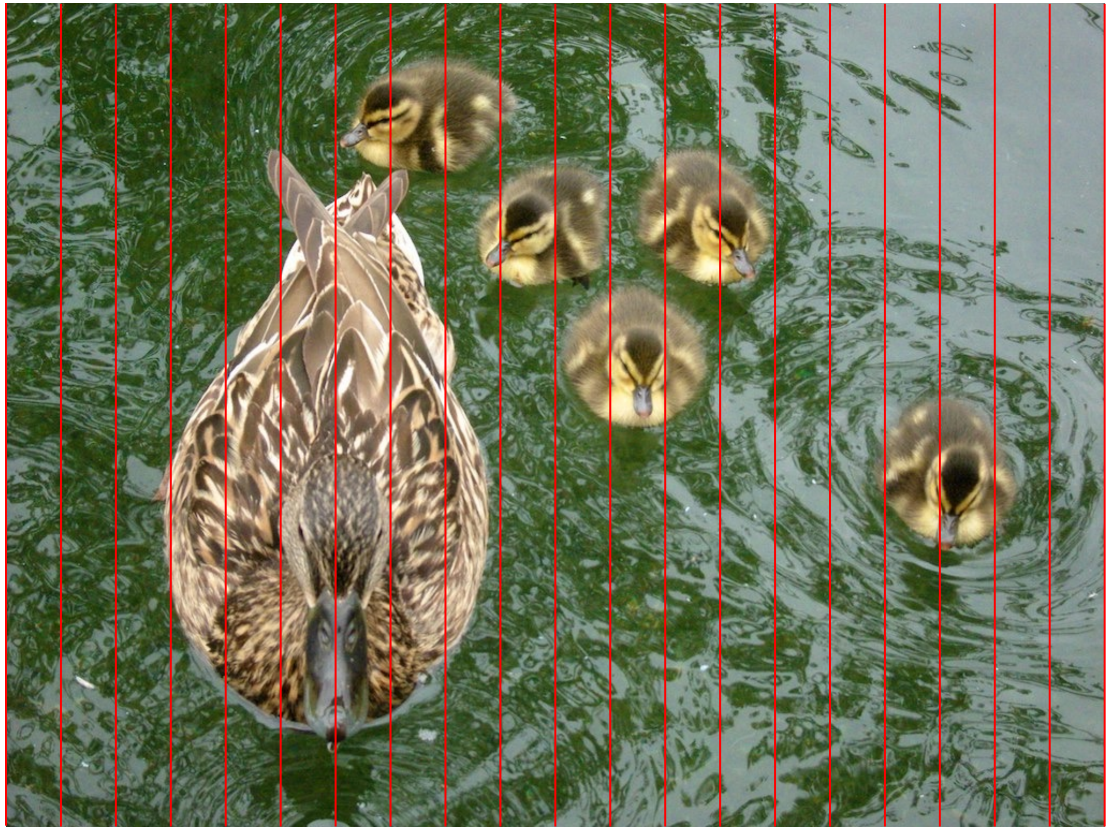
Cropping

- Removes a “block” of pixels

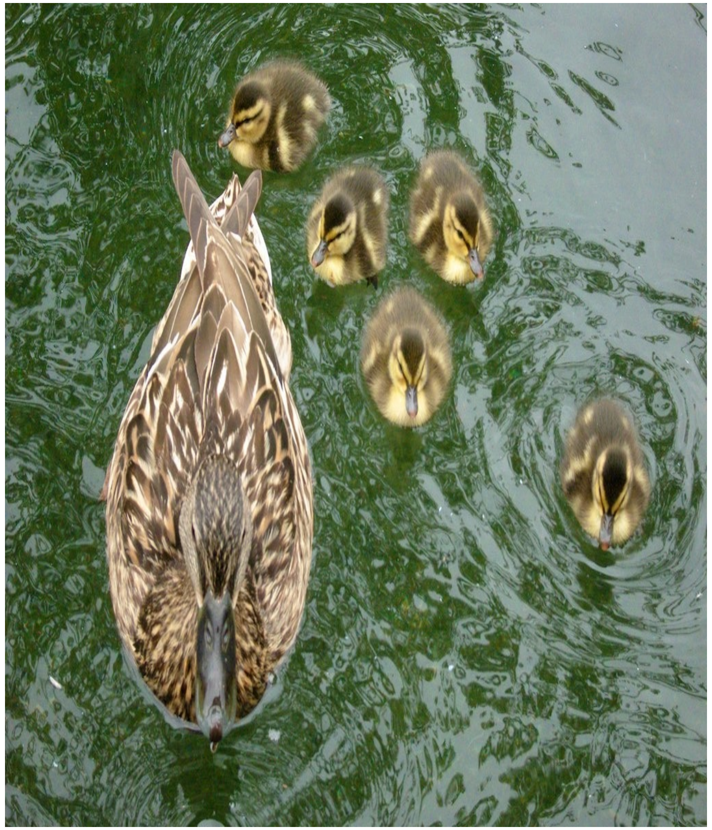


Scaling

- Removes “stripes” of pixels

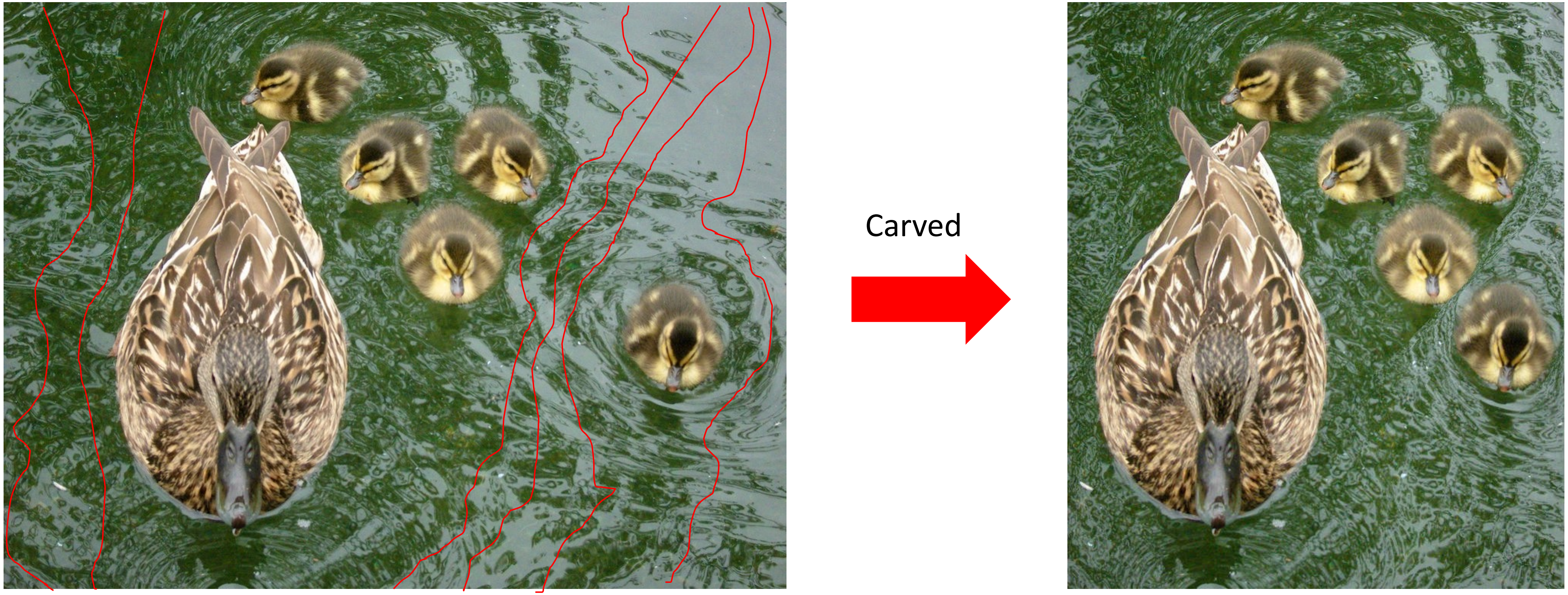


Scaled



Seam Carving

- Removes “least energy seam” of pixels
- https://trekhleb.dev/js-image-carver/



Seattle Skyline

Energy of a Seam

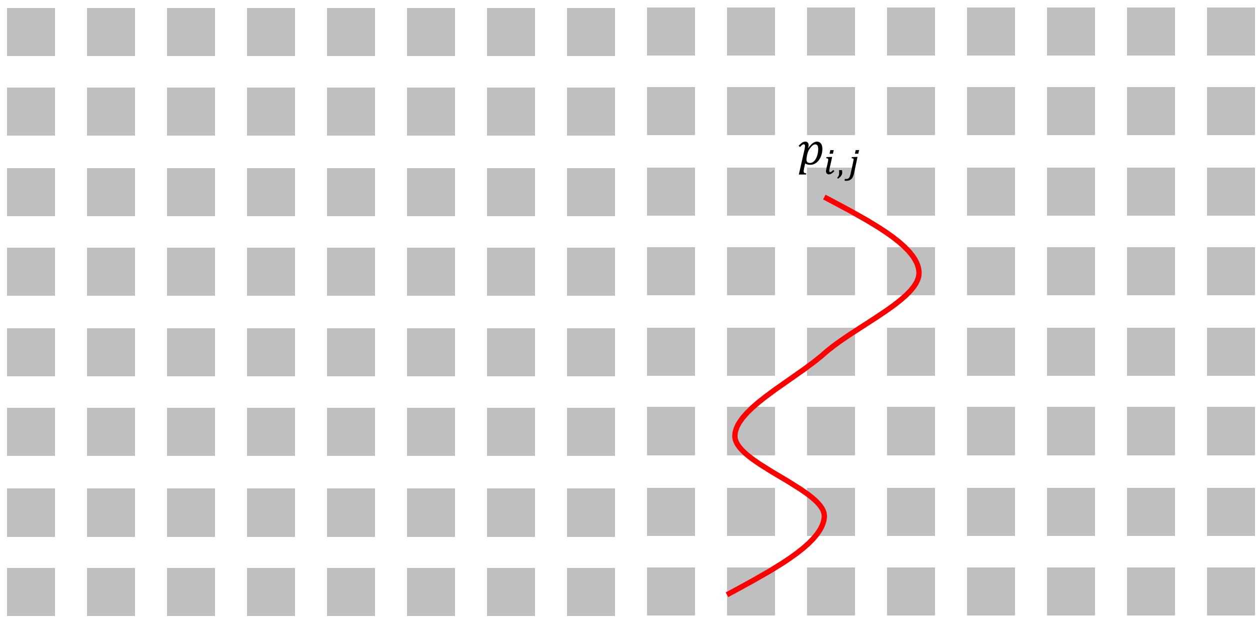
- $e(p) =$ energy of pixel p
  - Many choices
  - E.g.: change of gradient
    - how much the color of this pixel differs from its neighbors
  - Particular choice doesn't matter, we use it as a "black box"
- Energy of seam: sum of the energies of each pixel on the seam

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem is the (optimal) solutions to a smaller one plus one “decision”
- Idea:
  1. Identify the substructure of the problem
     - What are the options for the “last thing” done? What subproblem comes from each?
  2. Save the solution to each subproblem in memory
  3. Select an order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

Identify Recursive Structure

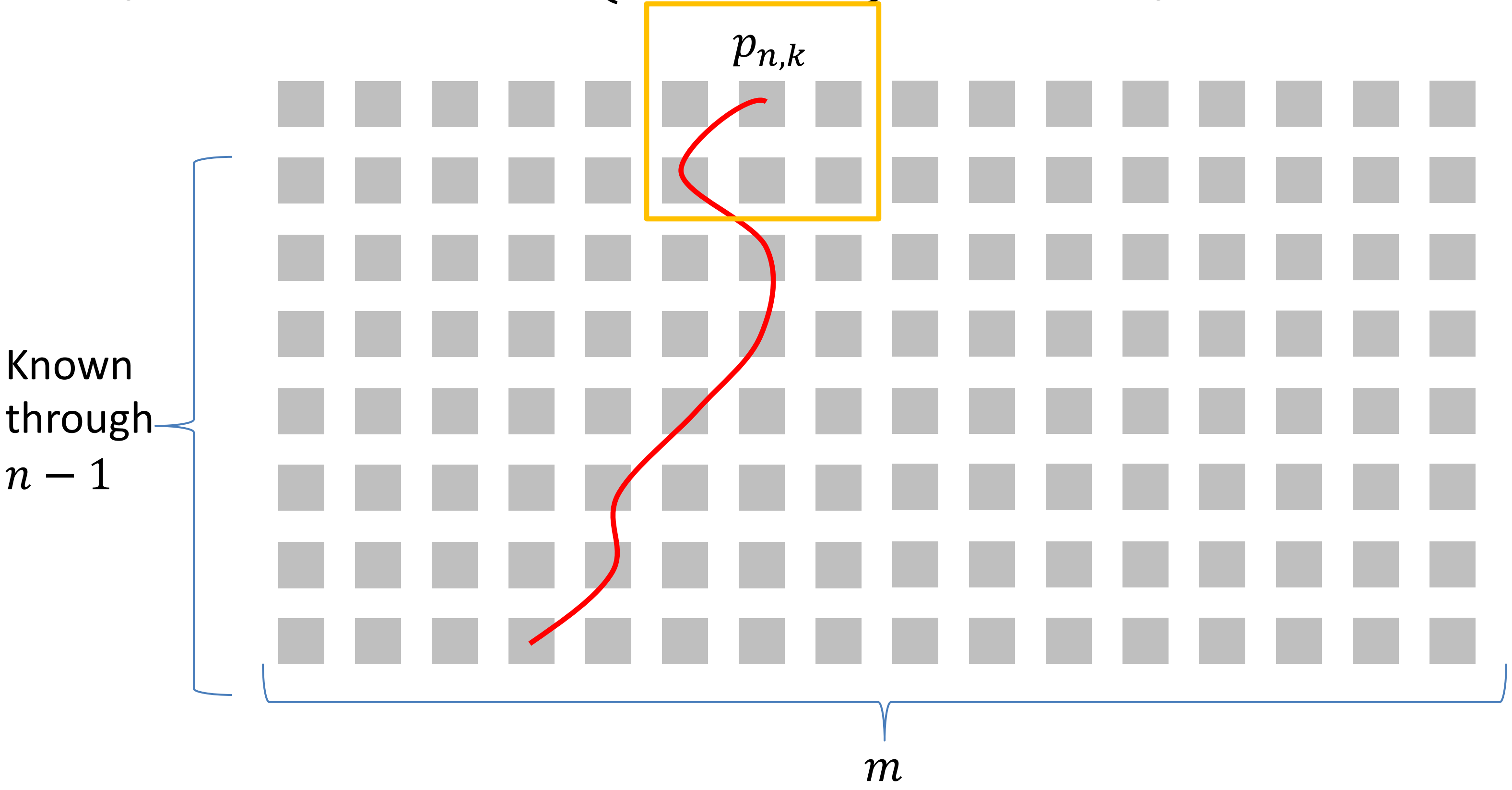
Let $S(i,j)$ = least energy seam from the bottom of the image up to pixel $p_{i,j}$



Computing $S(n,k)$

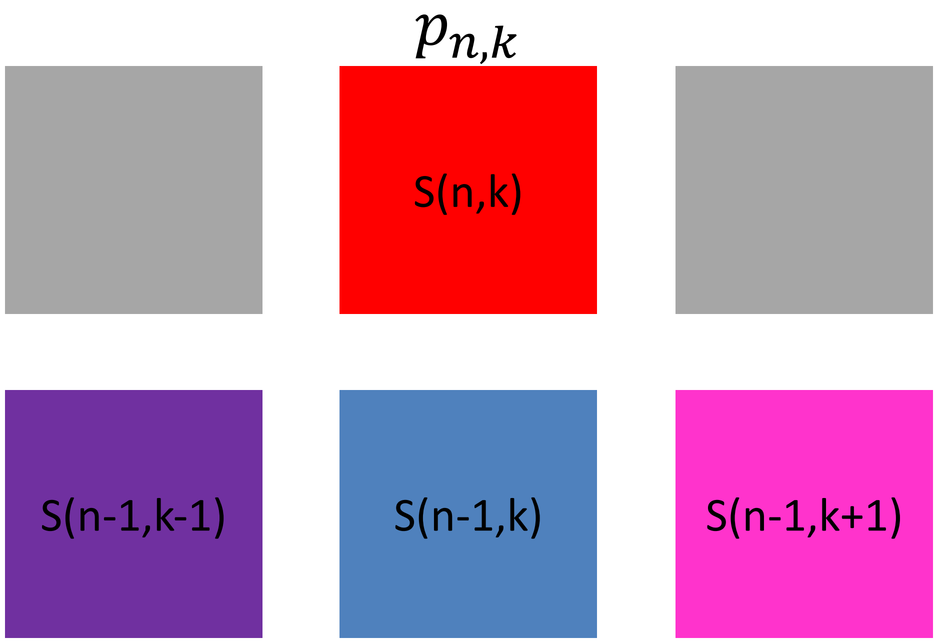
Assume we know the least energy seams for all of row $n-1$

(i.e. we know $S(n-1,\ell)$ for all ℓ)



Computing $S(n,k)$

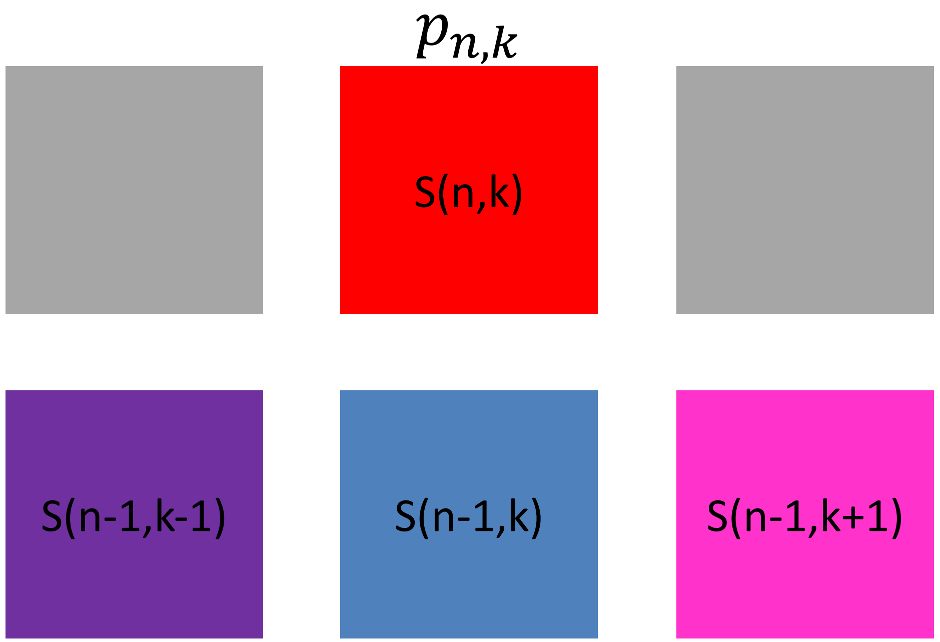
Assume we know the least energy seams for all of row $n-1$ (i.e. we know $S(n-1,\ell)$ for all ℓ)



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 ℓ)

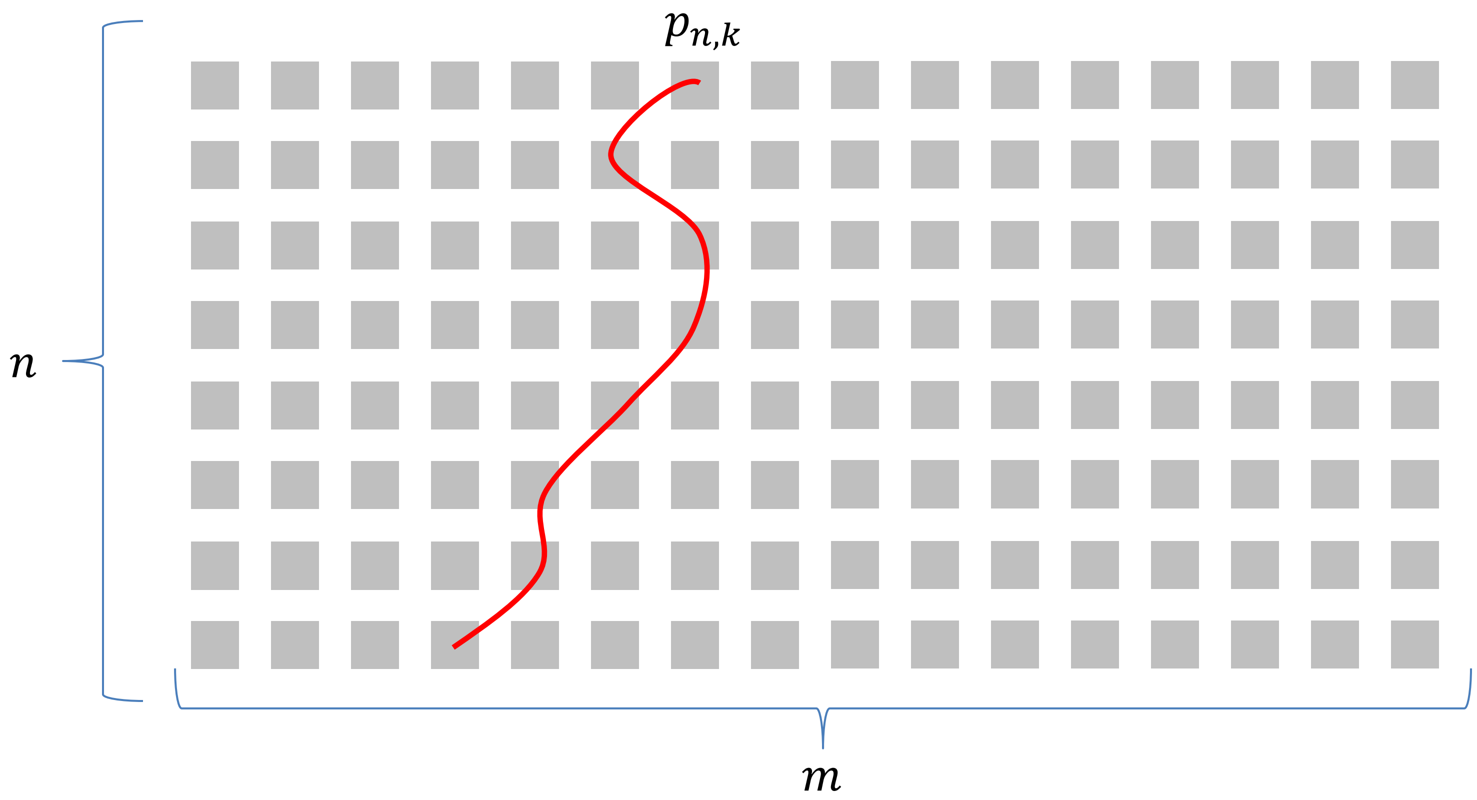
$$S(n,k) = min \begin{cases} S(n-1,k-1) + e(p_{n,k}) \\ S(n-1,k) + e(p_{n,k}) \\ S(n-1,k+1) + e(p_{n,k}) \end{cases}$$



Finding the Least Energy Seam

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

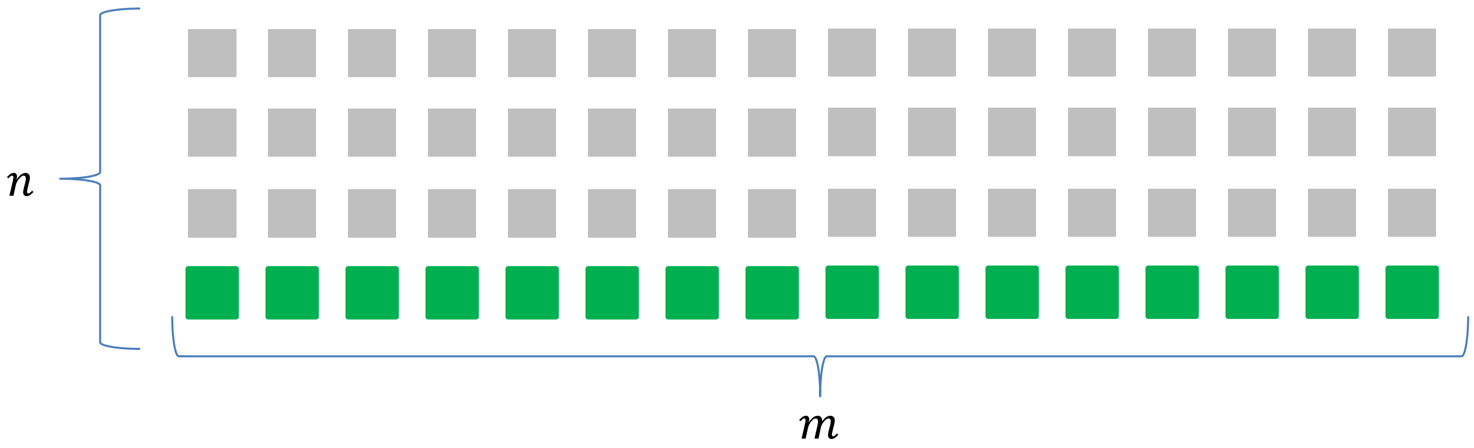
$$\min_{k=1}^{m}(S(n,k))$$



Bring It All Together

Start from bottom of image (row 1), solve up to top

Initialize $S(1,k) = e(p_{1,k})$ for each pixel in row 1



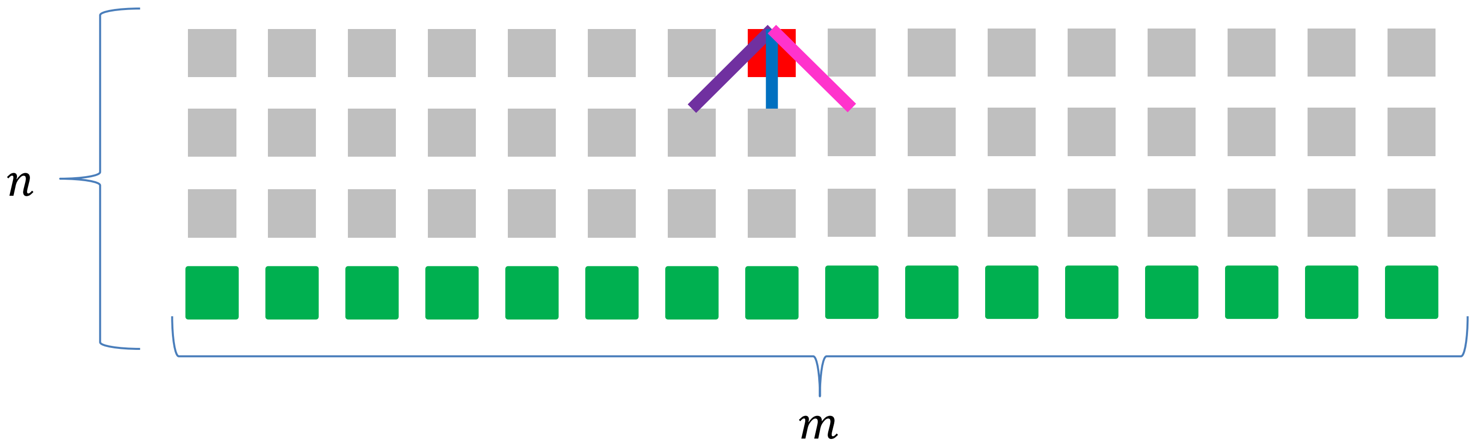
Energy of the seam initialized to the energy of that pixel

Bring It All Together

Start from bottom of image (row 1), solve up to top

Initialize $S(1,k) = e(p_{1,k})$ for each pixel $p_{1,k}$

For $i > 2$ find $S(i,k) = \min_{k=1}^{m} \begin{cases} S(i-1,k-1) + e(p_{i,k}) \\ S(i-1,k) + e(p_{i,k}) \\ S(i-1,k+1) + e(p_{i,k}) \end{cases}$



Energy of the seam initialized to the energy of that pixel

Top Down Least Energy Seam Pseudocode

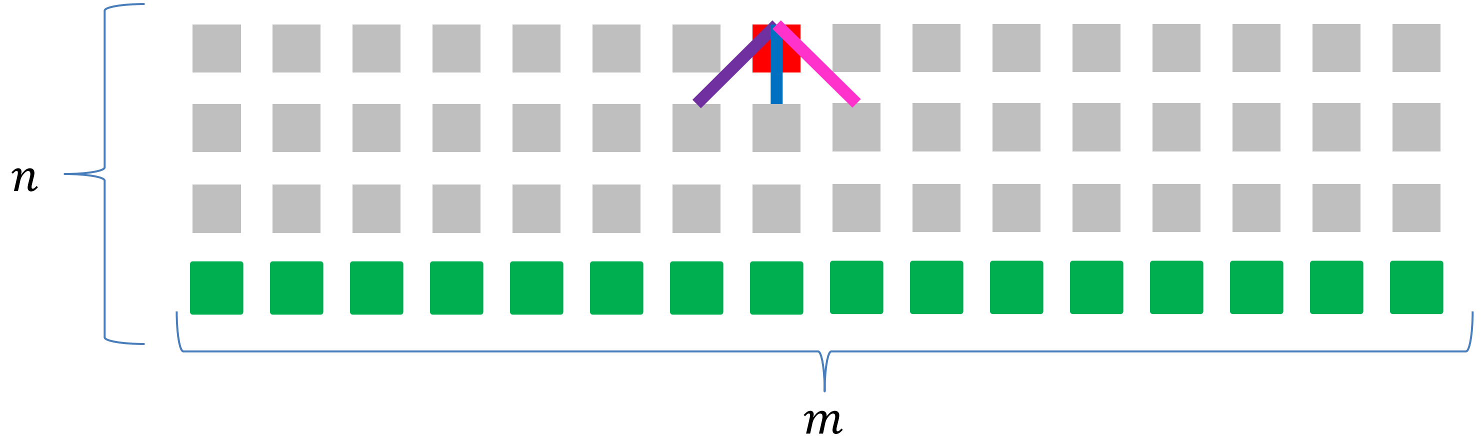
Bring It All Together

Start from bottom of image (row 1), solve up to top

Initialize $S(1,k) = e(p_{1,k})$ for each pixel $p_{1,k}$

$$\text{For } i \geq 2 \text{ find } S(i,k) = \min_{k=1}^{m} \begin{cases} S(i-1,k-1) + e(p_{i,k}) \\ S(i-1,k) + e(p_{i,k}) \\ S(i-1,k+1) + e(p_{i,k}) \end{cases}$$

Pick smallest from top row, backtrack, removing those pixels



Energy of the seam initialized to the energy of that pixel

Seam Carving Pseudocode

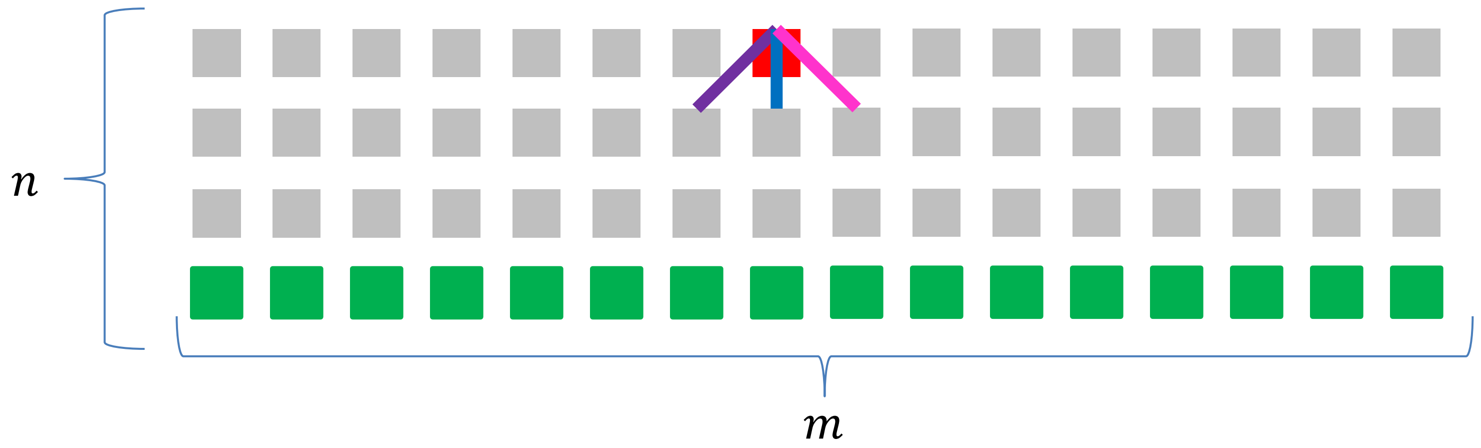
Run Time?

Start from bottom of image (row 1), solve up to top

Initialize $S(1,k) = e(p_{1,k})$ for each pixel $p_{1,k}$ $\Theta(m)$

For $i \geq 2$ find $S(i,k) = \min_{k=1}^{m} \begin{cases} S(i-1,k-1) + e(p_{i,k}) \\ S(i-1,k) + e(p_{i,k}) \\ S(i-1,k+1) + e(p_{i,k}) \end{cases}$ $\Theta(n \cdot m)$

Pick smallest from top row, backtrack, removing those pixels $\Theta(n+m)$



Energy of the seam initialized to the energy of that pixel

Seam Carving Bottom Up

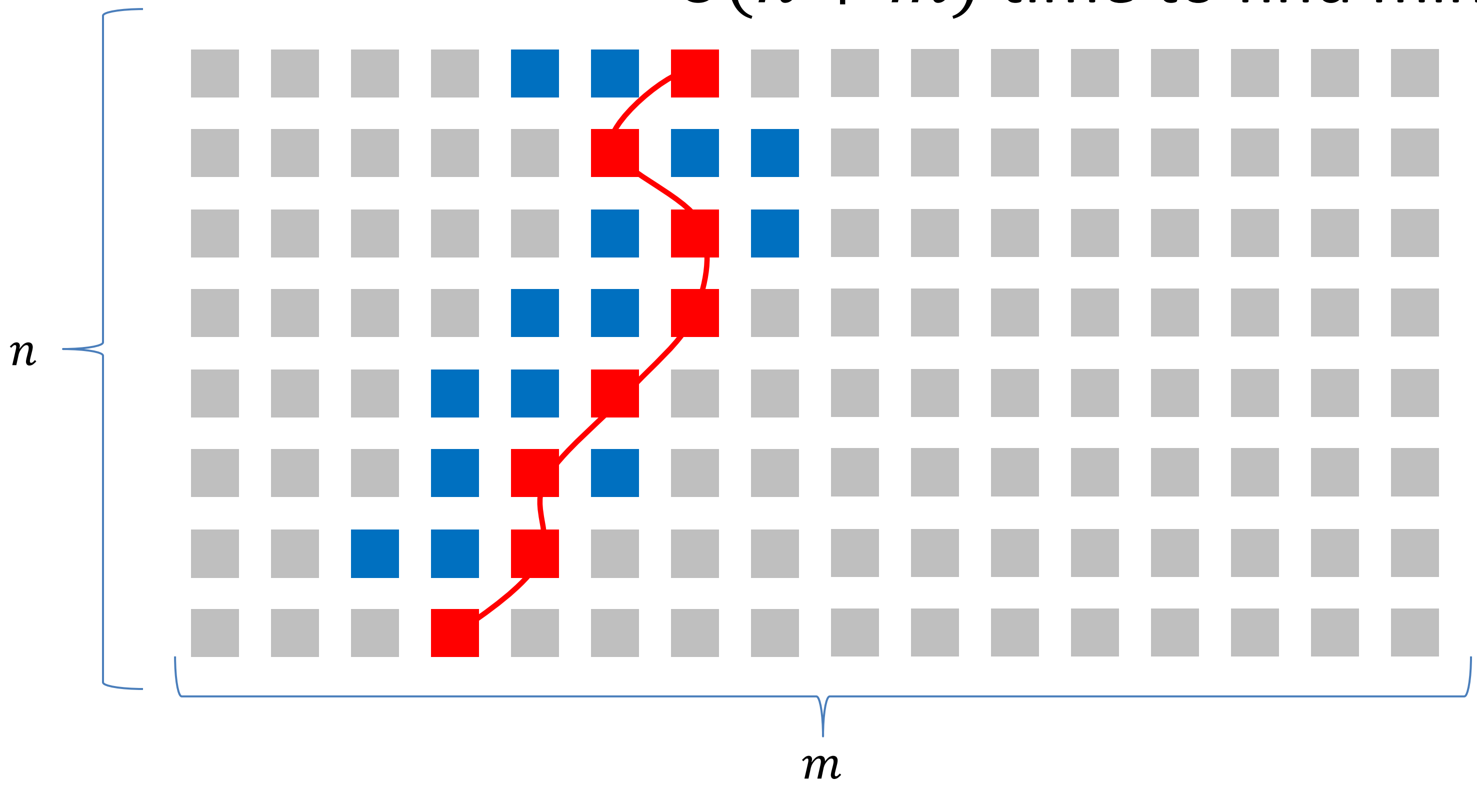
Repeated Seam Removal

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



Longest Common Subsequence

Given two sequences X and Y, find the length of their longest common subsequence

Example:

$X = ATCTGAT$

$Y = TGCATA$

$LCS = TCTA$

Brute force: Compare every subsequence of X with Y

$\Omega(2^n)$

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem is the (optimal) solutions to a smaller one plus one "decision"
- Idea:
  1. Identify the substructure of the problem
     - What are the options for the "last thing" done? What subproblem comes from each?
  2. Save the solution to each subproblem in memory
  3. Select an order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest

1. Identify Recursive Structure

Let $LCS(i,j)$ = length of the LCS for the first i characters of X, first j character of Y

Find $LCS(i,j)$:

Case 1: $X[i] = Y[j]$

$X = ATCTGCGT$

$Y = TGCATAT$

$LCS(i,j) = LCS(i-1,j-1) + 1$

Case 2: $X[i] \neq Y[j]$

$X=ATCTGCGA$

$Y=TGCATAT$

$LCS(i,j) = LCS(i,j-1)$

$X=ATCTGCGT$

$Y=TGCATAC$

$LCS(i,j) = LCS(i-1,j)$

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1,j-1)+1 & \text{if } X[i] = Y[j] \\ \max(LCS(i,j-1), LCS(i-1,j)) & \text{otherwise} \end{cases}$$

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem is the (optimal) solutions to a smaller one plus one “decision”
- Idea:
  1. Identify the substructure of the problem
     - What are the options for the “last thing” done? What subproblem comes from each?
  2. Save the solution to each subproblem in memory
  3. Select an order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

1. Identify Recursive Structure

Let $LCS(i,j) =$ length of the LCS for the first i characters of X, first j character of Y

Find $LCS(i,j)$:

Case 1: $X[i] = Y[j]$

$$X = ATCTGCGT$$
$$Y = TGCATAT$$
$$LCS(i,j) = LCS(i-1, j-1) + 1$$

Case 2: $X[i] \neq Y[j]$

$$X{=}ATCTGCGA$$
$$Y{=}TGCATAT$$
$$LCS(i,j) = LCS(i, j-1)$$

$$X{=}ATCTGCGT$$
$$Y{=}TGCATAC$$
$$LCS(i,j) = LCS(i-1, j)$$

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

Save to M[i,j]

Read from M[i,j] if present

```
X = “alkjdflaksjdf”
Y = “lakjsdflkasjdlfs”
M = 2d array of len(X) rows and len(Y) columns, initialized to -1
def LCS(int i, int j):
        # returns the length of the LCS shared between the length-i prefix of  X and length-j prefix of Y
        # memoization
        if M[i,j] > -1:
                return M[i,j]
        #base case:
        if i == 0 or j == 0:
                ans = 0
        elif X[i] == Y[j]:
                ans = LCS(i-1, j-1) + 1
        else:
                ans = max( LCS(i, j-1), LCS(i-1, j) )
        M[i,j] = ans
        return ans
print(LCS(len(X)+1, len(Y)+1)) # the answer for the entirety of X and Y
```

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem is the (optimal) solutions to a smaller one plus one “decision”
- Idea:
  1. Identify the substructure of the problem
     - What are the options for the “last thing” done? What subproblem comes from each?
  2. Save the solution to each subproblem in memory
  3. Select an order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

3. Solve in a Good Order

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1,j-1)+1 & \text{if } X[i] = Y[j] \\ \max(LCS(i,j-1), LCS(i-1,j)) & \text{otherwise} \end{cases}$$

$Y =$ \ $X =$		0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
T	1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
G	2	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**
C	3	**0**	**0**	**1**	**2**	**2**	**2**	**2**	**2**
A	4	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**
T	5	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**
A	6	**0**	**1**	**2**	**2**	**3**	**3**	**4**	**4**

To fill in cell (i,j) we need cells $(i-1,j-1),(i-1,j),(i,j-1)$
Fill from Top->Bottom, Left->Right (with any preference)

Run Time?

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$ \ $X =$		0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
T	1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
G	2	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**
C	3	**0**	**0**	**1**	**2**	**2**	**2**	**2**	**2**
A	4	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**
T	5	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**
A	6	**0**	**1**	**2**	**2**	**3**	**3**	**4**	**4**

Run Time: $\Theta(n \cdot m)$ (for $|X| = n$, $|Y| = m$)

Reconstructing the LCS

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$X =$; $Y =$

		0	A 1	T 2	C 3	$\boxed{T}$ 4	$\boxed{G}$ 5	$\boxed{A}$ 6	$\boxed{T}$ 7
	0	0	0	0	0	0	0	0	0
$\boxed{T}$	1	0	0	1	1	1	1	1	1
$\boxed{G}$	2	0	0	1	1	1	2	2	2
C	3	0	0	1	2	2	2	2	2
$\boxed{A}$	4	0	1	1	2	2	2	3	3
$\boxed{T}$	5	0	1	2	2	3	3	3	4
A	6	0	1	2	2	3	3	4	4

Start from bottom right,
if symbols matched, print that symbol then go diagonally
else go to largest adjacent

Reconstructing the LCS

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$ \ $X =$		0	$\boxed{A}$ 1	$\boxed{T}$ 2	$\boxed{C}$ 3	T 4	G 5	$\boxed{A}$ 6	$\boxed{T}$ 7
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
$\boxed{T}$	1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
G	2	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**
$\boxed{C}$	3	**0**	**0**	**1**	**2**	**2**	**2**	**2**	**2**
$\boxed{A}$	4	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**
$\boxed{T}$	5	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**
A	6	**0**	**1**	**2**	**2**	**3**	**3**	**4**	**4**

Start from bottom right,
if symbols matched, print that symbol then go diagonally
else go to largest adjacent

Reconstructing the LCS

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

		$X =$ 0	A 1	$\boxed{T}$ 2	$\boxed{C}$ 3	$\boxed{T}$ 4	G 5	$\boxed{A}$ 6	T 7
$Y =$	0	0	0	0	0	0	0	0	0
$\boxed{T}$	1	0	0	1	1	1	1	1	1
G	2	0	0	1	1	1	2	2	2
$\boxed{C}$	3	0	0	1	2	2	2	2	2
A	4	0	1	1	2	2	2	3	3
$\boxed{T}$	5	0	1	2	2	3	3	3	4
$\boxed{A}$	6	0	1	2	2	3	3	4	4

Start from bottom right,
if symbols matched, print that symbol then go diagonally
else go to largest adjacent