SPIS Breadth Lecture, Week 3: "Use it or Lose it" & Seam Carving

Seam Carving Slides by: Zach Dodds and Colleen Lewis, Harvey Mudd College

Return to Recursion: Power Set!

```
>>> powerset([1, 2])
[[], [2], [1], [1, 2]]

>>> powerset([1, 2, 3])
[[], [3], [2], [2, 3], [1], [1, 3],
  [1, 2], [1, 2, 3]]
>>> powerset([1])


>>> powerset([])
```

The order in which the subsets are presented is unimportant but within each subset, the order should be consistent with the input set.

Use-It-Or-Lose-It

(also known as exhaustive search… recall from APS!)

The Knapsack Problem...

Kingdom of Shmorbodia

Item	Weight	Value
Spam	2	100
Tofu	3	112
Chocolate	4	125

Knapsack Capacity: 5? 6? 7?


```
>>> knapsack(7,     )
237
```

Prof. I. Lai thinks that a “greedy solution” is the way to go!

Prove why a greedy solution will not work.

The Knapsack Problem…

Kingdom of Shmorbodia

Item	Weight	Value
Spam	2	100
Tofu	3	112
Chocolate	4	125

Knapsack Capacity: 5? 6? 7?

Use it or lose it:

- Pick one element to be "it" (The first item in the knapsack)
- Compute the solution using "it", then recompute the solution without "it"
- Choose whichever option is better (More value)
- Base case(s)??

Let's try it!

The Knapsack Revisited…

Kingdom of Shmorbodia

Item	Weight	Value
Spam	2	100
Tofu	3	112
Chocolate	4	125

Knapsack Capacity: 5? 6? 7?

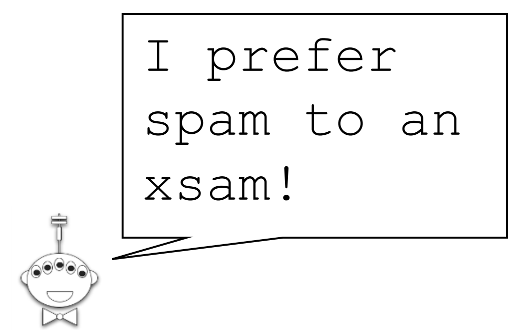

```
>>> knapsack(7, [ [2, 100], [3, 112], [4, 125] ])
[237, [ [3, 112], [4, 125] ] ]
```

Modify the knapsack code so that it also returns the list of items chosen

Comparing DNA via Longest Common Subsequence (LCS)

```
AGGACAT
ATTACGAT
```

```
>>> LCS(“AGGACAT”, “ATTACGAT”)
5
>>> LCS(“spam”, “sam!”)
3
>>> LCS(“spam”, “xsam”)
3
```



Recursive Approach…

```
def LCS(S1, S2):
    if BASE CASE
    else:
```

```
LCS("spam", "sam!")
```

Try this in your notes!

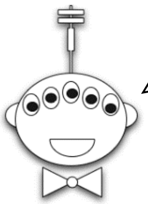
Power Set!

```
>>> powerset([1, 2])
[[], [2], [1], [1, 2]]

>>> powerset([1, 2, 3])
[[], [3], [2], [2, 3], [1], [1, 3],
  [1, 2], [1, 2, 3]]

>>> powerset([1])


>>> powerset([])
```



This really demonstrates the power of functional programming!

The order in which the subsets are presented is unimportant but within each subset, the order should be consistent with the input set.

A Useful Helper for **`powerset`**

Write a function **`addToEachList( elem, LoL )`** that takes an element **`elem`**, and a list of lists, **`LoL`**, and returns a list of lists, where **`elem`** has been added to each list in **`LoL`**. Use only recursion (no loops!)

A few examples

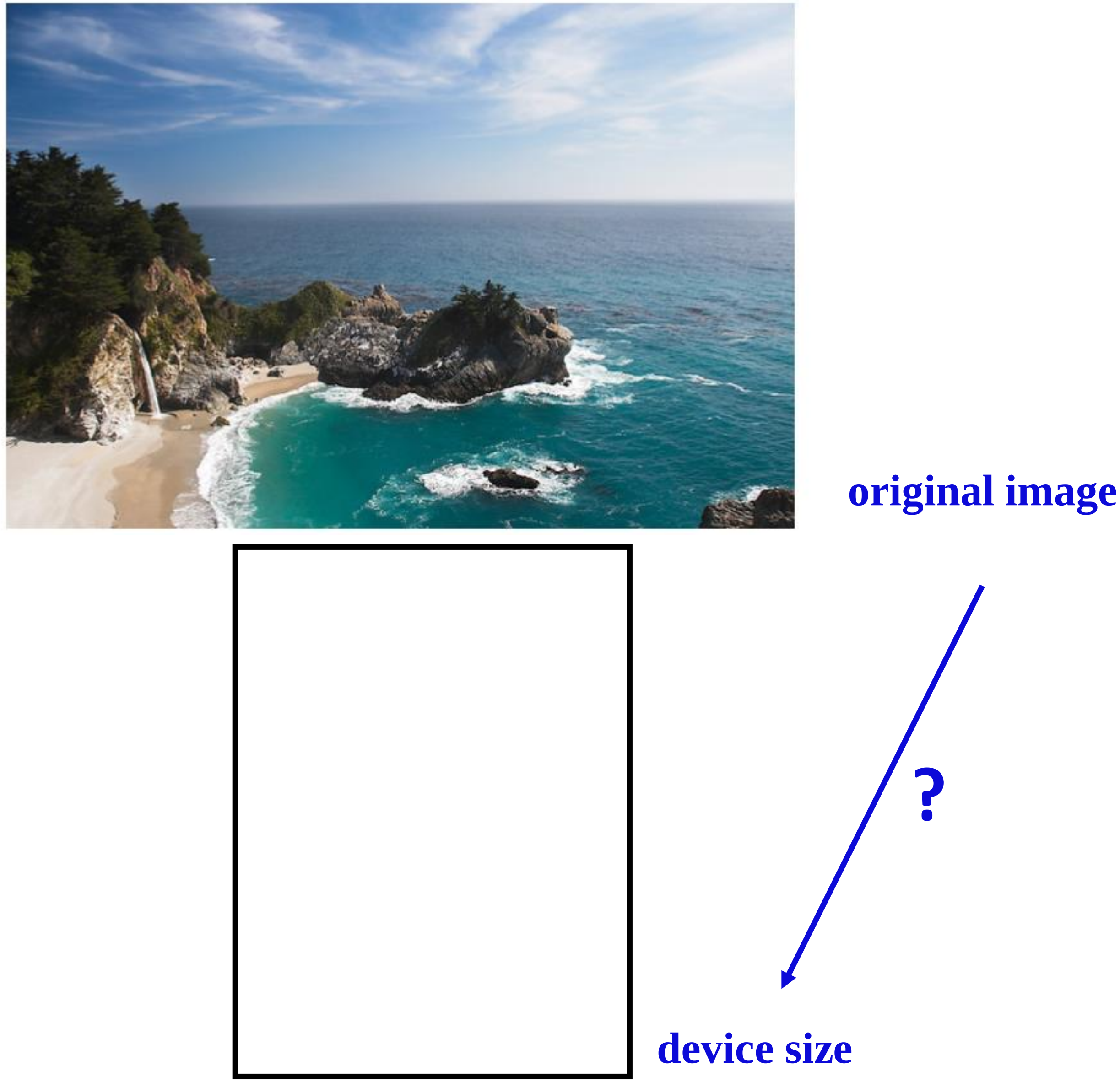
```
>>> addToEachList( 1, [[2], [3], [4]] )
[[1, 2], [1, 3], [1, 4]]
>>> addToEachList( 42, [[]])
[[42]]
>>> addToEachList( 42, [] )
[]
>>> addToEachList( 42, 43 )
ERROR
```

Seam Carving

http://www.ics.uci.edu/~dramanan/teaching/cs116_fall08/hw/Project/Seam/

The problem



Three possible solutions

original image

Seam Carving

Which one do you like best?

Seam Carving for Content-Aware Image Resizing

Shai Avidan
Mitsubishi Electric Research Labs

Ariel Shamir
The Interdisciplinary Center & MERL

Remove less important seams

Seam carving for content-aware image resizing

S Avidan, A Shamir - ACM Transactions on graphics (TOG), 2007 - dl.acm.org

Effective resizing of images should not only use geometric constraints, but consider the image content as well. We present a simple image operator called **seam carving** that supports content-aware image resizing for both reduction and expansion. A **seam** is an …

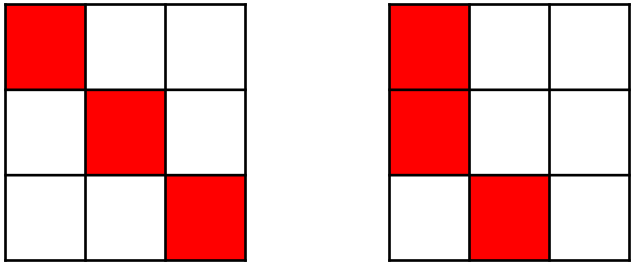
☆ 99 Cited by 1713 Related articles All 38 versions Web of Science: 424

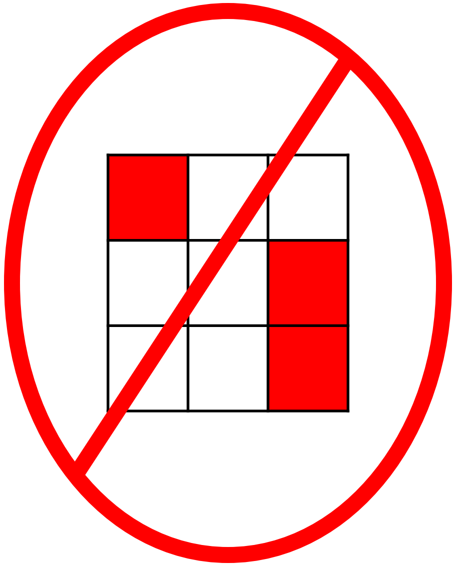
you're encouraged to read the original paper, and use it as a guide...

Definition of Seams (+ Demo)

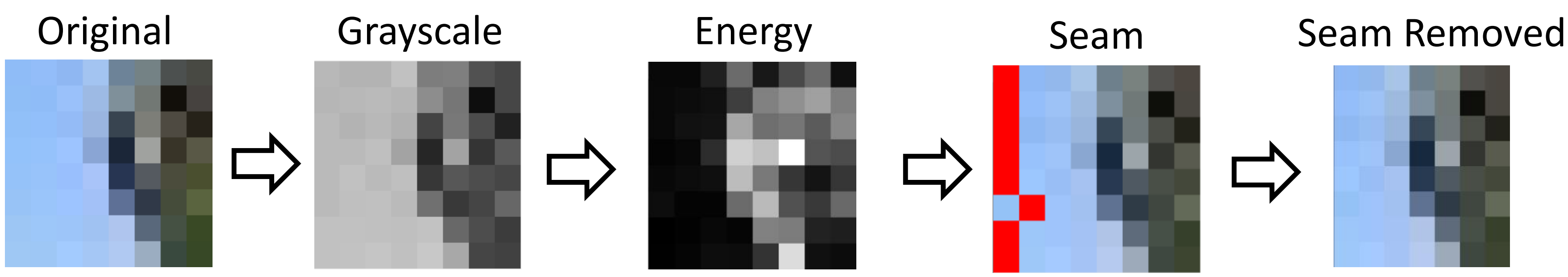
Seams

- Go from the top row to the bottom row
- Connect by an edge or a corner
- Seams are the lowest "energy" path

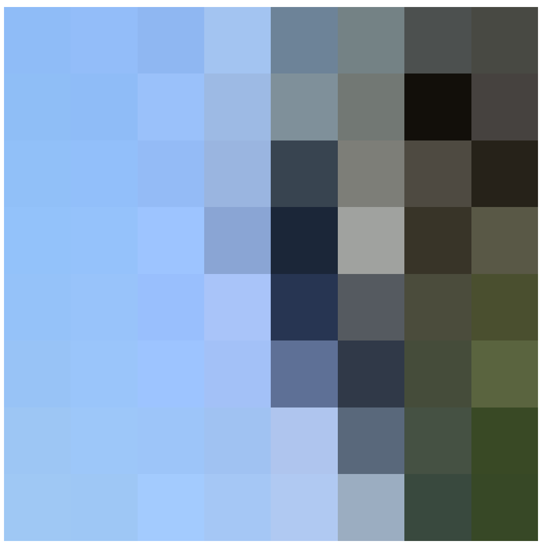




Seam Carving: The Big Picture

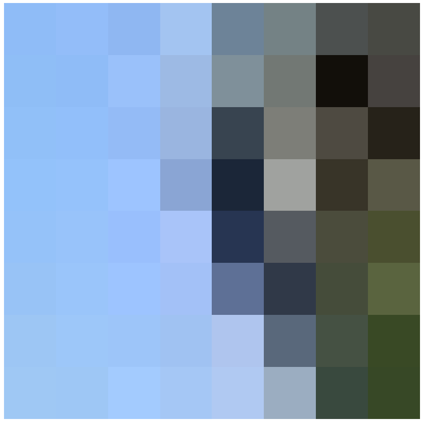


How pictures are represented

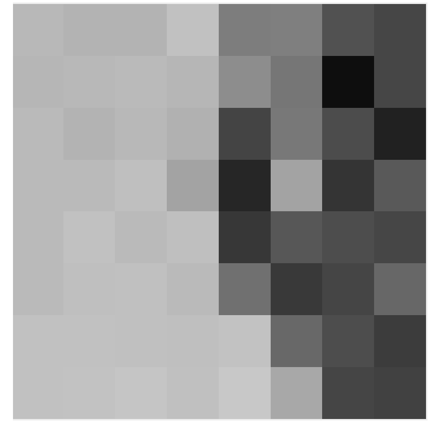


	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	R:144 G:189 B:254	R:143 G:185 B:245	R:147 G:184 B:236	R:168 G:197 B:231	R:110 G:128 B:140	R:121 G:130 B:127	R:82 G:81 B:78	R:76 G:70 B:64
y = 1	R:142 G:187 B:252	R:148 G:190 B:250	R:157 G:194 B:246	R:158 G:186 B:223	R:126 G:144 B:158	R:112 G:121 B:120	R:14 G:15 B:10	R:73 G:70 B:64
y = 2	R:146 G:192 B:254	R:143 G:185 B:243	R:152 G:189 B:242	R:153 G:181 B:220	R:54 G:71 B:87	R:112 G:122 B:124	R:75 G:77 B:71	R:34 G:34 B:25
y = 3	R:147 G:193 B:252	R:152 G:194 B:252	R:159 G:196 B:249	R:138 G:167 B:209	R:23 G:41 B:63	R:156 G:165 B:170	R:51 G:53 B:48	R:89 G:91 B:79
y = 4	R:149 G:194 B:251	R:156 G:199 B:254	R:153 G:190 B:245	R:166 G:194 B:241	R:40 G:57 B:82	R:79 G:89 B:98	R:73 G:79 B:71	R:68 G:72 B:58
y = 5	R:148 G:194 B:246	R:154 G:197 B:250	R:159 G:196 B:251	R:162 G:193 B:240	R:95 G:114 B:146	R:47 G:59 B:70	R:64 G:71 B:64	R:99 G:106 B:88
y = 6	R:155 G:199 B:246	R:157 G:199 B:249	R:159 G:196 B:251	R:164 G:194 B:244	R:176 G:196 B:231	R:94 G:107 B:123	R:70 G:80 B:71	R:54 G:64 B:43
y = 7	R:156 G:200 B:245	R:158 G:200 B:248	R:165 G:202 B:255	R:165 G:195 B:245	R:182 G:202 B:239	R:157 G:170 B:186	R:62 G:72 B:64	R:61 G:69 B:48

Step 1: Convert to Grayscale



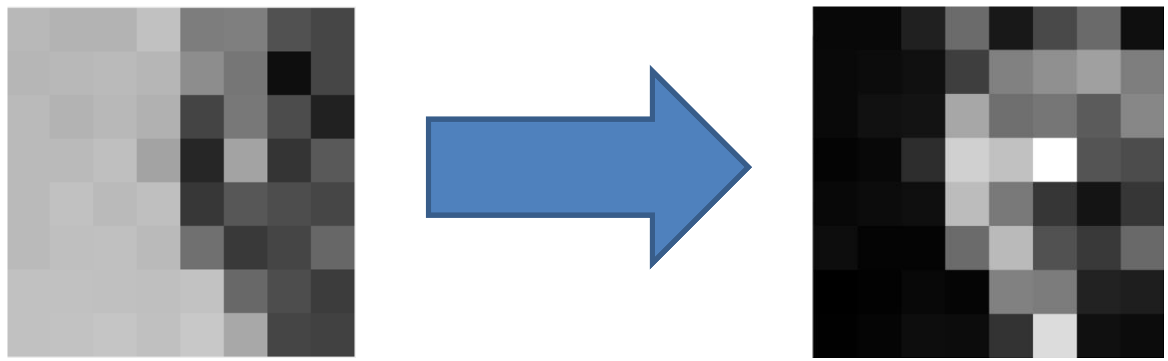
	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	R:144 G:189 B:254	R:143 G:185 B:245	R:147 G:184 B:236	R:168 G:197 B:231	R:110 G:128 B:140	R:121 G:130 B:127	R:82 G:81 B:78	R:76 G:70 B:64
y = 1	R:142 G:187 B:252	R:148 G:190 B:250	R:157 G:194 B:246	R:158 G:186 B:223	R:126 G:144 B:158	R:112 G:121 B:120	R:14 G:15 B:10	R:73 G:70 B:64
y = 2	R:146 G:192 B:254	R:143 G:185 B:243	R:152 G:189 B:242	R:153 G:181 B:220	R:54 G:71 B:87	R:112 G:122 B:124	R:75 G:77 B:71	R:34 G:34 B:25
y = 3	R:147 G:193 B:252	R:152 G:194 B:252	R:159 G:196 B:249	R:138 G:167 B:209	R:23 G:41 B:63	R:156 G:165 B:170	R:51 G:53 B:48	R:89 G:91 B:79
y = 4	R:149 G:194 B:251	R:156 G:199 B:254	R:153 G:190 B:245	R:166 G:194 B:241	R:40 G:57 B:82	R:79 G:89 B:98	R:73 G:79 B:71	R:68 G:72 B:58
y = 5	R:148 G:194 B:246	R:154 G:197 B:250	R:159 G:196 B:251	R:162 G:193 B:240	R:95 G:114 B:146	R:47 G:59 B:70	R:64 G:71 B:64	R:99 G:106 B:88
y = 6	R:155 G:199 B:246	R:157 G:199 B:249	R:159 G:196 B:251	R:164 G:194 B:244	R:176 G:196 B:231	R:94 G:107 B:123	R:70 G:80 B:71	R:54 G:64 B:43
y = 7	R:156 G:200 B:245	R:158 G:200 B:248	R:165 G:202 B:255	R:165 G:195 B:245	R:182 G:202 B:239	R:157 G:170 B:186	R:62 G:72 B:64	R:61 G:69 B:48



Red, Green, and Blue are all this:

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	184	180	179	193	125	127	81	70
y = 1	182	185	189	182	141	119	14	70
y = 2	186	180	184	177	68	120	76	33
y = 3	187	189	191	163	38	163	52	89
y = 4	188	193	186	191	55	87	77	70
y = 5	187	191	192	189	112	57	69	103
y = 6	193	193	192	191	194	105	77	60
y = 7	193	194	197	192	200	168	69	65

Step 2: Calculate Energy



	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	184	180	179	193	125	127	81	70
y = 1	182	185	189	182	141	119	14	70
y = 2	186	180	184	177	68	120	76	33
y = 3	187	189	191	163	38	163	52	89
y = 4	188	193	186	191	55	87	77	70
y = 5	187	191	192	189	112	57	69	103
y = 6	193	193	192	191	194	105	77	60
y = 7	193	194	197	192	200	168	69	65

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	**187**	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	**162**	12	9

Step 2: Calculate Energy

$$energy = abs(horizontal\_derivative) + abs(vertical\_derivative)$$

$$horizontal\_derivative = i(x+1, y) - i(x, y)$$

$$vertical\_derivative = i(x, y+1) - i(x, y)$$

	x = 0	x = 1	x = 2
y = 0	184	180	179
y = 1	182	185	189
y = 2	186	180	184

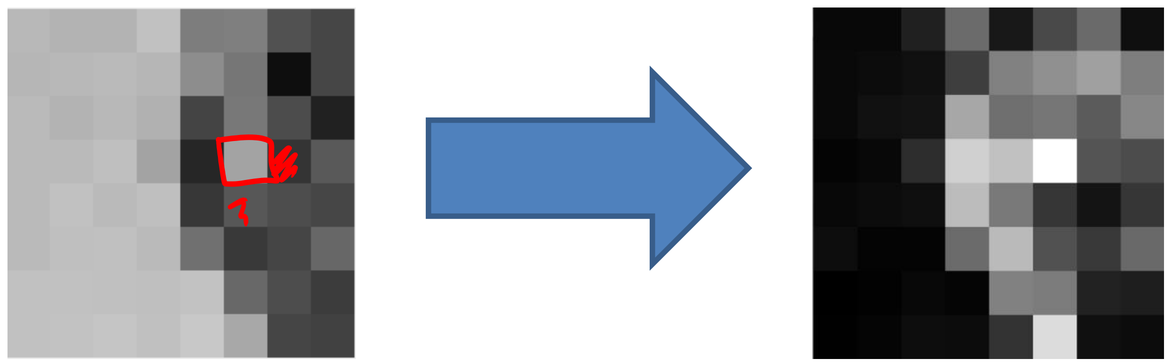
	x = 0	x = 1
y = 0	6	6
y = 1	7	9

Check that these are right.

$$energy = abs(i(x+1, y) - i(x, y)) + abs(i(x, y+1) - i(x, y))$$

Step 2: Calculate Energy

These are just numbers, but we CAN visualize them as grayscale values.



	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	184	180	179	193	125	127	81	70
y = 1	182	185	189	182	141	119	14	70
y = 2	186	180	184	177	68	120	76	33
y = 3	187	189	191	163	38	163	52	89
y = 4	188	193	186	191	55	87	77	70
y = 5	187	191	192	189	112	57	69	103
y = 6	193	193	192	191	194	105	77	60
y = 7	193	194	197	192	200	168	69	65

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	**187**	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	**162**	12	9

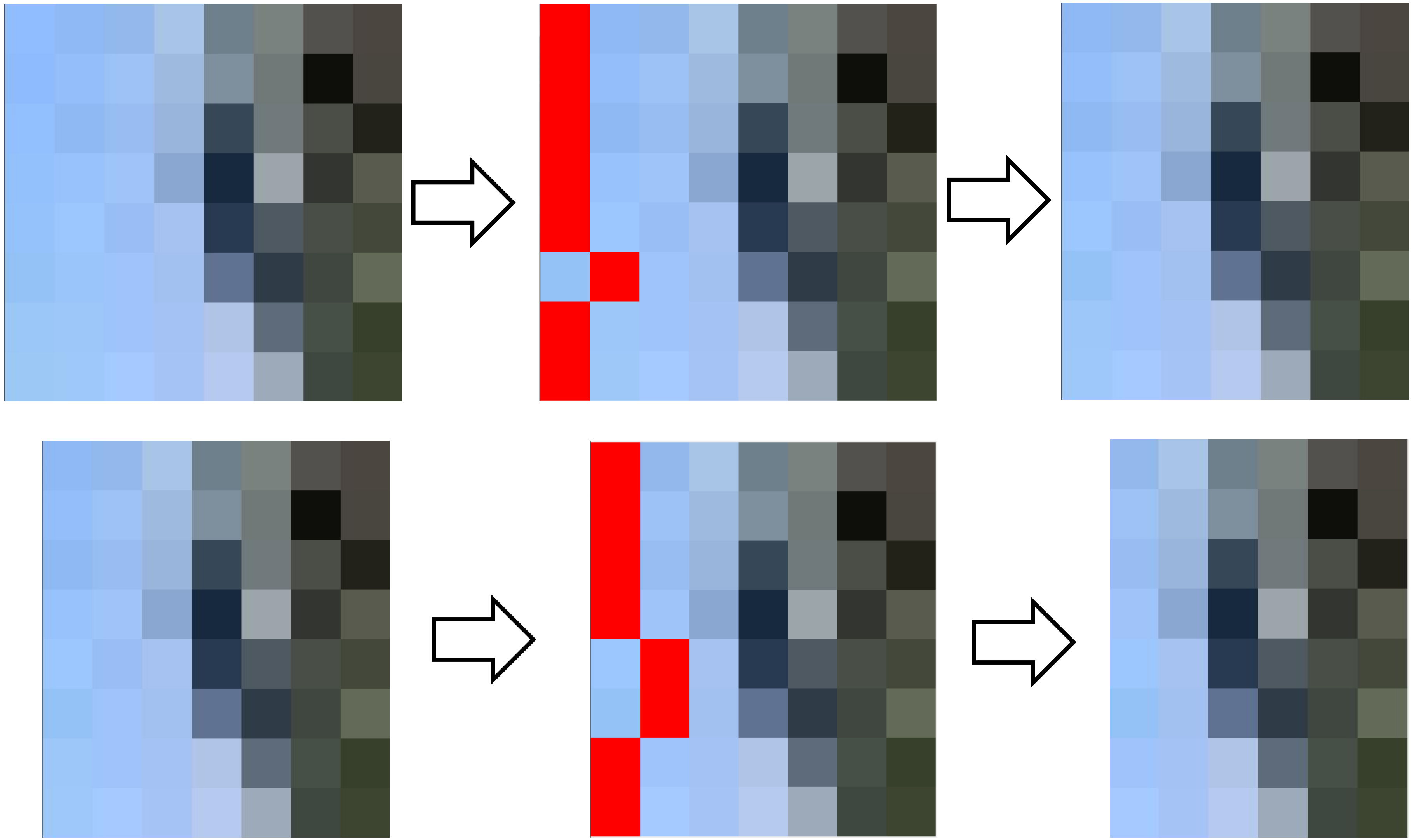
Step 3: Find the minimum path (from top to bottom)

This is the interesting step.
We'll come back to this!

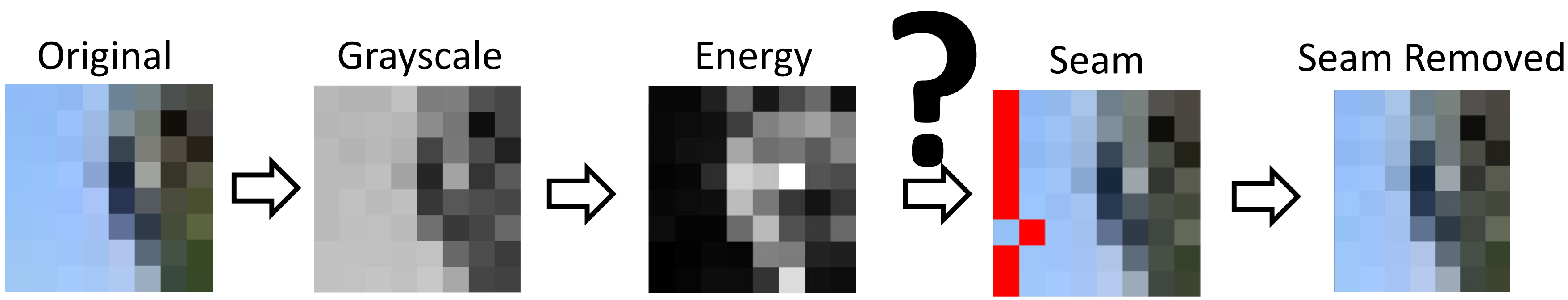
	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

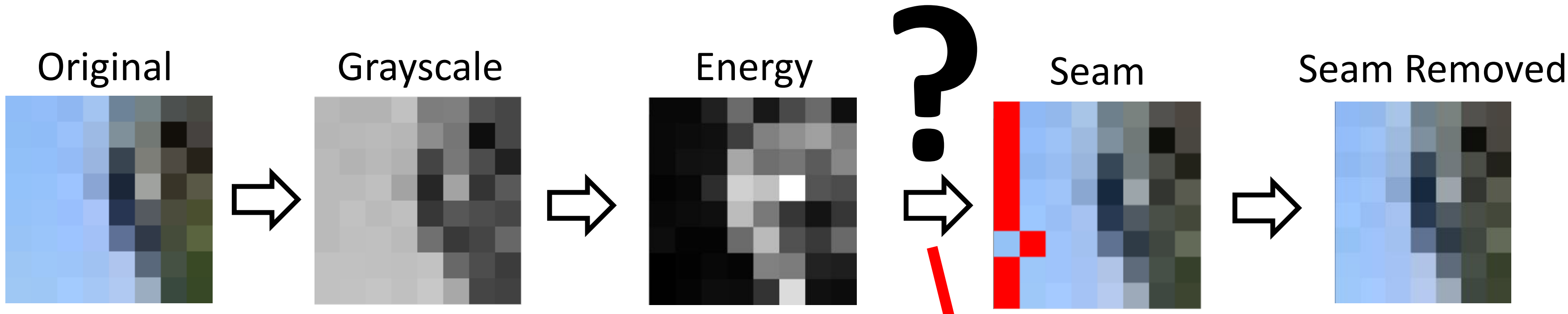
Step 4: Remove the pixels from the minimum path



Seam Carving
The Big Picture
END



How do we pick a seam?



Seams

- Go from the top row to the bottom row
- Connect by an edge or a corner
- **Seams are the lowest "energy" path**

We'll discuss one possible method using **dynamic programming**, and look at benefits of this over a recursive use-it-or-lose-it

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	8	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

Write Recursive Pseudocode to find a vertical path cost based upon `getEnergy`

```
def getPathCost( pic, x, y ):
```

Which starting x

First time will be h

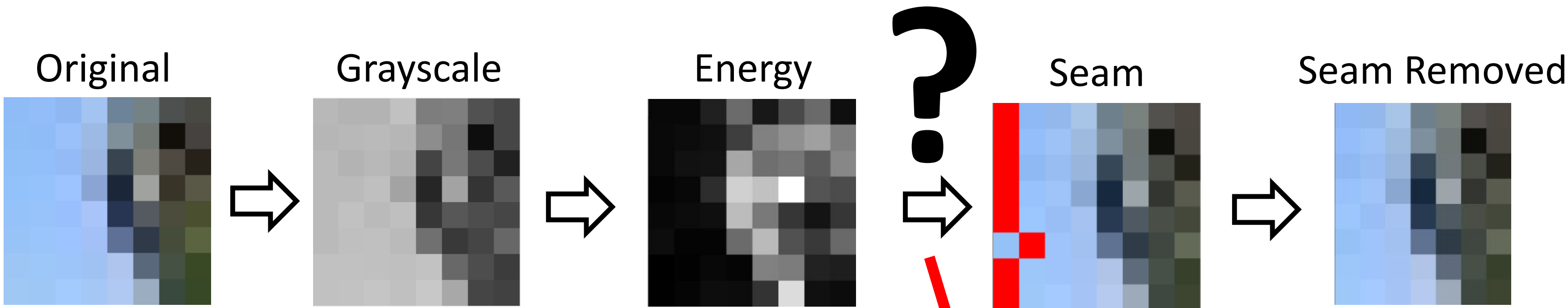
	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

Write Recursive Pseudocode to find a vertical path *cost* based upon `getEnergy`

```
def getPathCost( pic, x, y ):
   (w, h) = pic.size
   if y == 0:
      return getEnergy(x, y)
   # 3 possible branches
   best = Infinity
   for xnew in [x-1, x, x+1]:
     if xnew >= 0 and xnew < w:
       next = getPathCost(pic, xnew, y-1)
     if next < best:
        best = next
   return best + getEnergy(x, y)
```

OK, so what's the problem???

How do we pick a seam?



We'll discuss one possible method using **dynamic programming**, and later look at benefits of this over a recursive use-it-or-lose-it

Seams

- Go from the top row to the bottom row
- Connect by an edge or a corner
- **Seams are the lowest "energy" path**

Identify Minimal Paths

Store Information in `int[][] table`

$$\text{table}[x][0] = \text{getEnergy}[x][0]$$

$$\text{table}[x][y] = \text{getEnergy}[x][y] + \min \begin{cases} \text{table}[x-1][y-1] \\ \text{table}[x][y-1] \\ \text{table}[x+1][y-1] \end{cases}$$

`getEnergy`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

`table`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0								
y = 1								
y = 2								
y = 3								
y = 4								
y = 5								
y = 6								
y = 7								

Fill out the y=0 row of `table` based upon the base case

$$\text{table}[x][0] = \text{getEnergy}[x][0]$$

$$\text{table}[x][y] = \text{getEnergy}[x][y] + \min \begin{cases} \text{table}[x-1][y-1] \\ \text{table}[x][y-1] \\ \text{table}[x+1][y-1] \end{cases}$$

`getEnergy`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

`table`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1								
y = 2								
y = 3								
y = 4								
y = 5								
y = 6								
y = 7								

Fill out each row of `table` based upon the 2nd rule

$$\text{table}[x][0] \quad = \quad \text{getEnergy}[x][0]$$

$$\text{table}[x][y] \quad = \quad \text{getEnergy}[x][y] + \min \begin{cases} \text{table}[x-1][y-1] \\ \text{table}[x][y-1] \\ \text{table}[x+1][y-1] \end{cases}$$

`getEnergy`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	$\boxed{7}$	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

`table`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1								
y = 2								
y = 3								
y = 4								
y = 5								
y = 6								
y = 7								

Fill out each row of `table` based upon the 2nd rule (cont.)

$$\text{table}[x][0] = \text{getEnergy}[x][0]$$

$$\text{table}[x][y] = \text{getEnergy}[x][y] + \min\begin{cases}\text{table}[x-1][y-1]\\ \text{table}[x][y-1]\\ \text{table}[x+1][y-1]\end{cases}$$

`getEnergy`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

`table`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	13							
y = 2								
y = 3								
y = 4								
y = 5								
y = 6								
y = 7								

What do the cells in `getEnergy` and `table` mean?

This is the cost of including this pixel in the path

$$\text{table}[x][0] = \text{getEnergy}[x][0]$$

$$\text{table}[x][y] = \text{getEnergy}[x][y] + \min\begin{cases}\text{table}[x-1][y-1]\\ \text{table}[x][y-1]\\ \text{table}[x+1][y-1]\end{cases}$$

`getEnergy`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	**106**	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

`table`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	13	15	18	64	113	**124**	129	104
y = 2								
y = 3								
y = 4								
y = 5								
y = 6								
y = 7								

This is the cost of the shortest path that includes this pixel

Final version of `table`

How do we find the shortest path from this?

O(height * width)

$$\text{table}[x][0] = \text{getEnergy}[x][0]$$

$$\text{table}[x][y] = \text{getEnergy}[x][y] + \min \begin{cases} \text{table}[x-1][y-1] \\ \text{table}[x][y-1] \\ \text{table}[x+1][y-1] \end{cases}$$

`getEnergy`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	7	9	12	46	95	106	118	93
y = 2	7	13	14	123	82	87	67	99
y = 3	3	6	33	153	142	187	62	56
y = 4	6	9	11	138	89	40	15	40
y = 5	10	3	3	79	137	60	42	77
y = 6	0	2	6	4	95	91	25	22
y = 7	1	4	10	9	38	162	12	9

`table`

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	13	15	18	64	113	124	129	104
y = 2	20	26	29	141	146	200	171	203
y = 3	23	26	59	182	283	333	233	227
y = 4	29	32	37	197	271	273	242	267
y = 5	39	32	35	116	334	302	284	319
y = 6	32	34	38	39	211	375	309	306
y = 7	33	36	44	47	77	373	318	315

This is the cost of the shortest path that includes this pixel

Keep track of the parent

parent

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
y = 1								
y = 2								
y = 3								
y = 4								
y = 5								
y = 6								
y = 7								

table

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	13	15	18	64	113	124	129	104
y = 2	20	26	29	141	146	200	171	203
y = 3	23	26	59	182	283	333	233	227
y = 4	29	32	37	197	271	273	242	267
y = 5	39	32	35	116	334	302	284	319
y = 6	32	34	38	39	211	375	309	306
y = 7	33	36	44	47	77	373	318	315

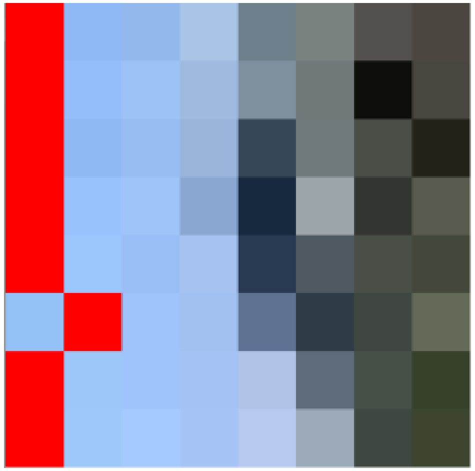
Keep track of the parent

parent

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
y = 1	x=0	x=1	x=1	x=4	x=4	x=4	x=7	x=7
y = 2	x=0	x=0	x=1	x=2	x=3	x=4	x=7	x=7
y = 3	x=0	x=0	x=1	x=2	x=3	x=4	x=6	x=6
y = 4	x=0	x=0	x=1	x=2	x=3	x=6	x=7	x=7
y = 5	x=0	x=0	x=1	x=2	x=3	x=6	x=6	x=6
y = 6	x=1	x=1	x=1	x=2	x=3	x=6	x=6	x=6
y = 7	x=0	x=0	x=1	x=2	x=3	x=4	x=7	x=7

table

	x = 0	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6	x = 7
y = 0	6	6	24	79	18	54	78	11
y = 1	13	15	18	64	113	124	129	104
y = 2	20	26	29	141	146	200	171	203
y = 3	23	26	59	182	283	333	233	227
y = 4	29	32	37	197	271	273	242	267
y = 5	39	32	35	116	334	302	284	319
y = 6	32	34	38	39	211	375	309	306
y = 7	33	36	44	47	77	373	318	315



You can backtrack to find the path from the smallest total path (in the last row)

How do we pick a seam? (USING DYNAMIC PROGRAMMING)

- Fill out a 2D array "`table`" based upon the rules below

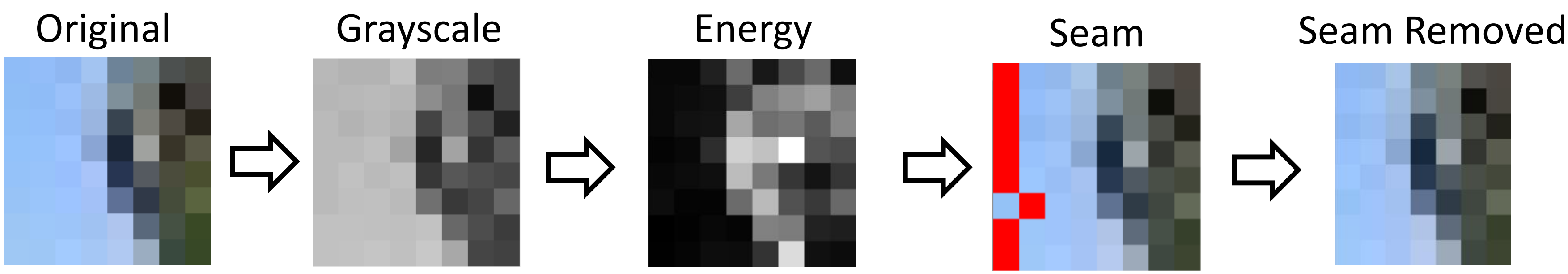
- Fill out a 2D array "`parent`" based upon which x value has the minimum path in the row above.

- Pick the path with the minimum weight (in the last row) and backtrack through `parent` to find the path.

$$\text{table}[x][0] = \text{getEnergy}[x][0]$$

$$\text{table}[x][y] = \text{getEnergy}[x][y] + \min \begin{cases} \text{table}[x-1][y-1] \\ \text{table}[x][y-1] \\ \text{table}[x+1][y-1] \end{cases}$$

How do we pick a seam?



Seams

- Go from the top row to the bottom row
- Connect by an edge or a corner
- **Seams are the lowest "energy" path**

Not perfect…

Straightforward seam carving does not always work!

Target size

What's conspiring against the algorithm here?

Problems?

Straightforward seam carving does not always work!

What's conspiring against the algorithm here?

Not the face!

Protecting the results from a face-detector makes a huge difference…

What's been lost instead?

Applications: Seams can remove people!

Applications: *Lost a shoe?*

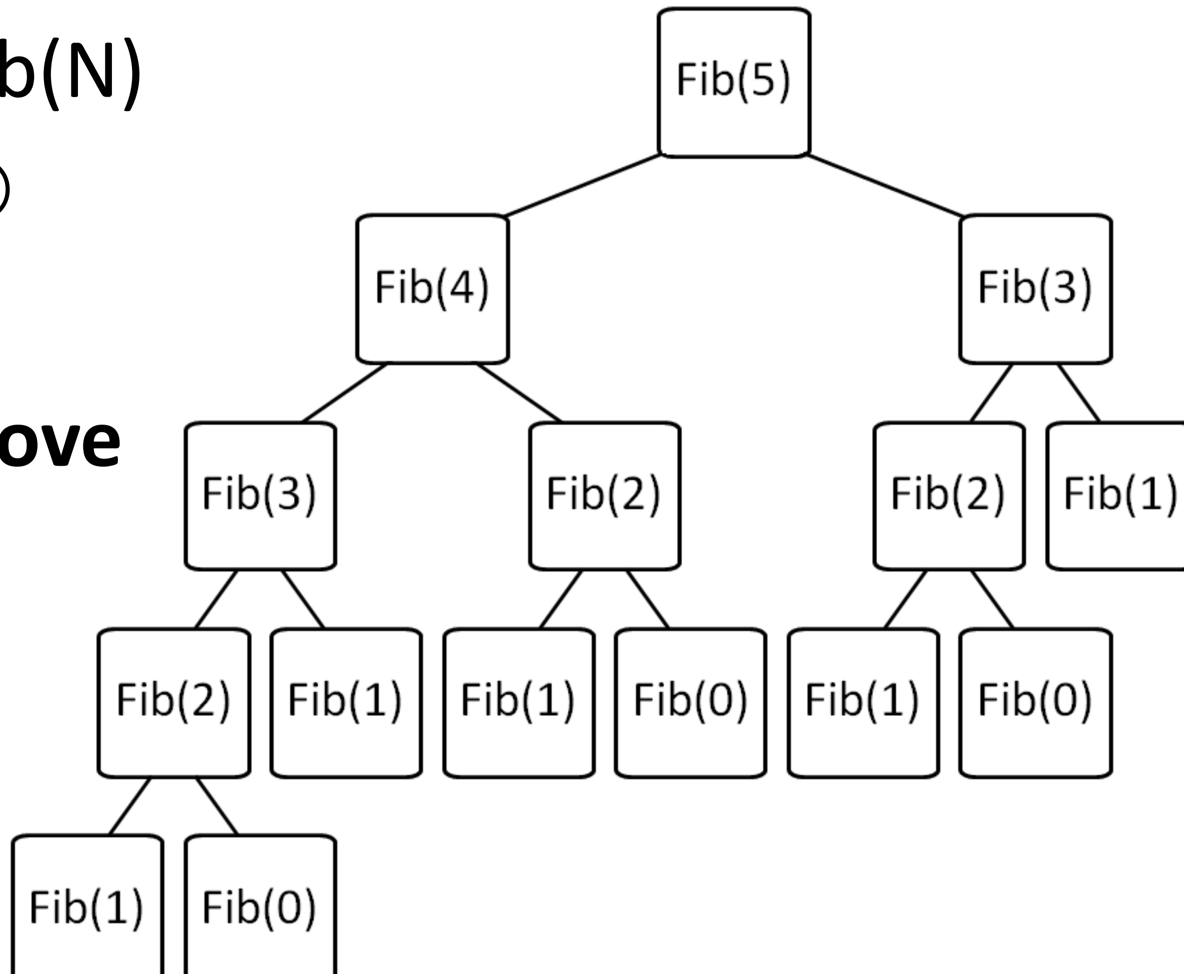
Original

All others had 1 shoe removed! Which shoe?

Another DP example: **Fibonacci**

Brute force Fib(N) is $O(2^N)$ ☹

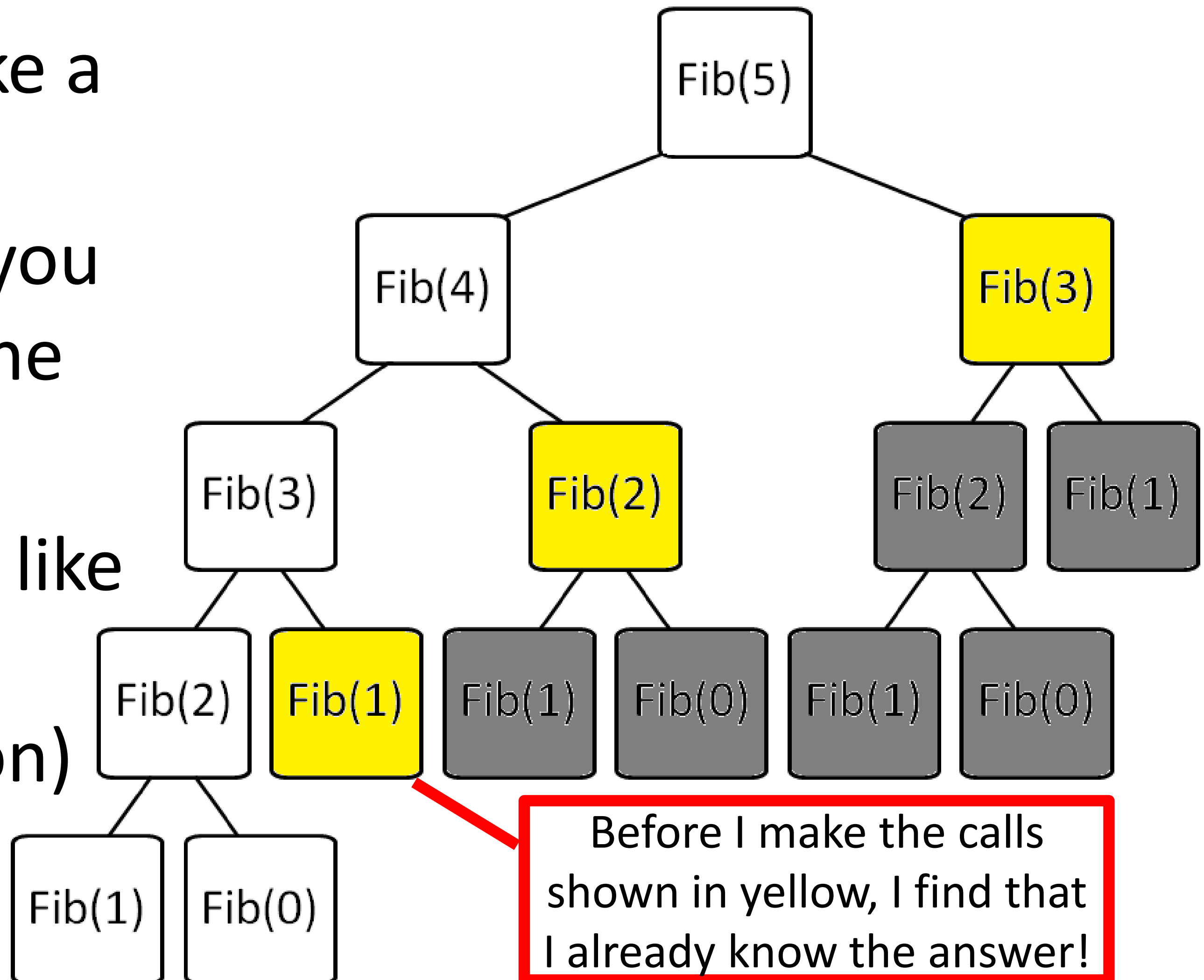
Can we improve this?



Fibonacci with Memoization

Keep track of recursive calls you've made and store the result!

- Before you make a recursive call, check to see if you already know the answer.
- **Top-down** (just like the regular recursive version)



Fibonacci with Dynamic Programming

Work from the bottom-up!

- Instead of doing the recursion, work from the bottom-up to create a table of answers.
- We will **NEVER** make a recursive call, because we will always have already calculated the required components (e.g. `fib(n-1) + fib(n-2)`)

Fib(0)	Fib(1)	Fib(2)	Fib(3)	Fib(4)	Fib(5)
0	1	1	2	3	5

Consider: **Fibonacci**

- Brute force of Fib(N)
  - is $O(2^N)$ ☹

- Memoization of Fib(N)
  - is O(N) + *overhead for keeping track of what recursive calls have been made*

- Dynamic Programming of Fib(N)
  - is O(N) + *overhead for making a table (O(N))*

Expanding the image?

original image

Will this work?

I want to add 42 pixels in width

```
for (int i=0; i<42; i++){

     (1) find a seam
     (2) duplicate the seam
}
```

Expanding the image?

original image

best-seam (least-energy) expansion

The best 50% of seams used for more uniform expansion

Expanding the image?

200% of original (scaling)

200% of original (seam-insertion)