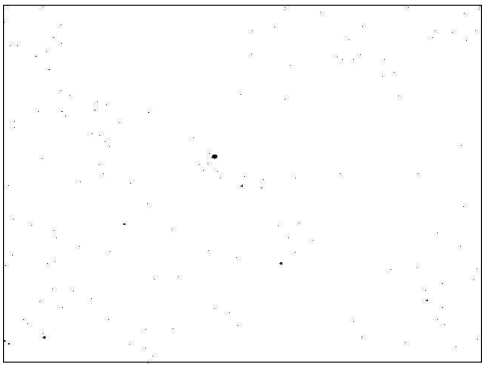
Multimedia Programming

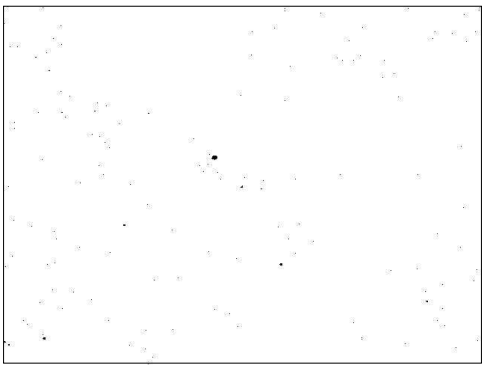
Counting Stars

Problem: how many stars are in this image?

Converted to black and white



Converted to black and white



How many black blobs?

A "blob" is a group of adjacent pixels

A "blob" is a group of adjacent pixels

Decide that "adjacent" means 4-way, not 8-way

A "blob" is a group of adjacent pixels

Decide that "adjacent" means 4-way, not 8-way

Want to count each blob once

A "blob" is a group of adjacent pixels

Decide that "adjacent" means 4-way, not 8-way

Want to count each blob once

Scan the image left to right, top to bottom

A "blob" is a group of adjacent pixels

Decide that "adjacent" means 4-way, not 8-way

Want to count each blob once

Scan the image left to right, top to bottom

Increment count each time we find a new blob

A "blob" is a group of adjacent pixels

Decide that "adjacent" means 4-way, not 8-way

Want to count each blob once

Scan the image left to right, top to bottom

Increment count each time we find a new blob

But how do we tell?

A "blob" is a group of adjacent pixels

Decide that "adjacent" means 4-way, not 8-way

Want to count each blob once

Scan the image left to right, top to bottom

Increment count each time we find a new blob

But how do we tell?

Turn black pixels red

A "blob" is a group of adjacent pixels

Decide that "adjacent" means 4-way, not 8-way

Want to count each blob once

Scan the image left to right, top to bottom

Increment count each time we find a new blob

But how do we tell?

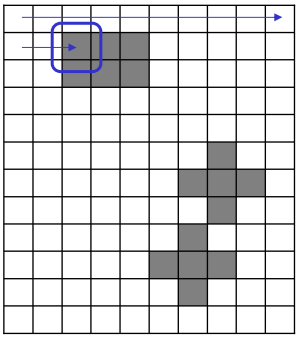
Turn black pixels red

If there is a red pixel before this one in scan order,

we have already counted this blob

Scan to black pixel

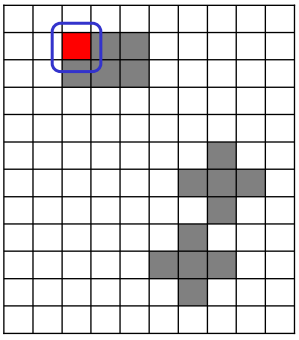
`count: 0`



Scan to black pixel

Mark it seen

`count: 0`



Scan to black pixel

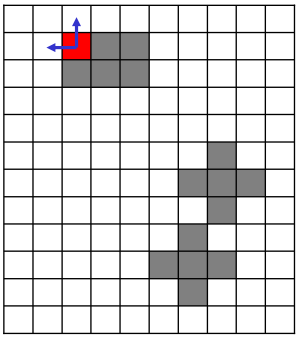
Mark it seen

Nothing red ahead of it

in scan order, so it must be

a new star

```
count: 1
```

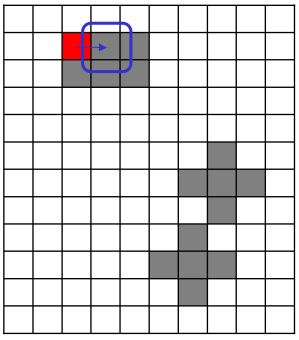


Scan to black pixel

Mark it seen

Keep scanning

```
count: 1
```



Scan to black pixel

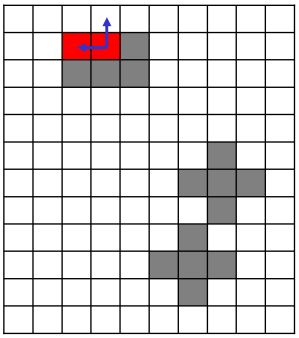
Mark it seen

Keep scanning

Does have red predecessor,

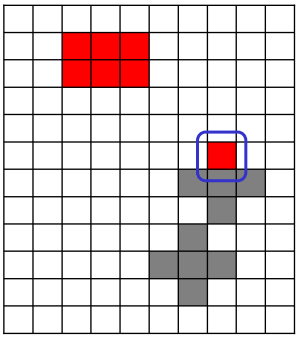
so this star already counted

`count: 1`



A new star!

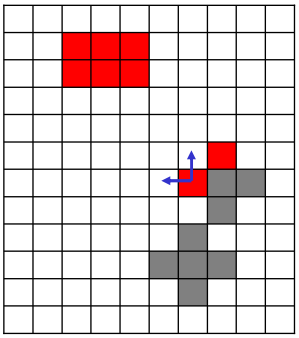
`count: 2`



A new star!

But this looks like a new star as well...

```
count: 2
```

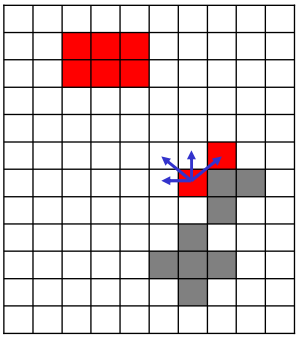


A new star!

But this looks like a new star as well...

So look in more directions

```
count: 2
```



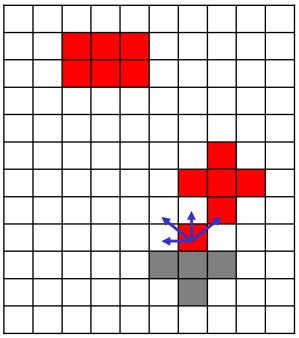
A new star!

But this looks like a new star as well...

So look in more directions

But this also gives the wrong answer

```
count: 2
```



A new star!

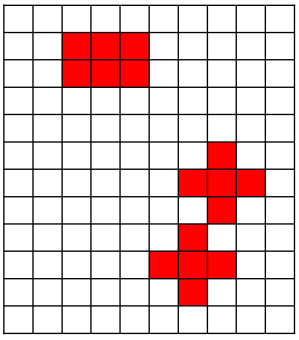
But this looks like a new star as well...

So look in more directions

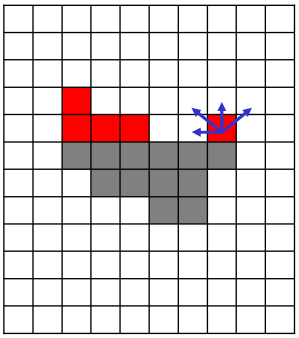
But this also gives the wrong answer

Change our definition so that these two blobs are one star?

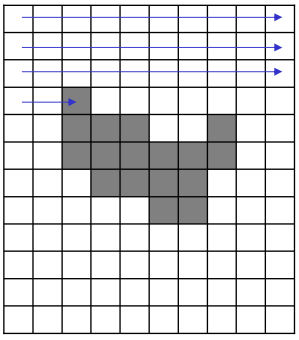
```
count: 2
```



Still gives the wrong answer

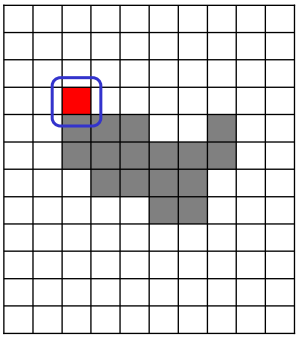


Solution: use *flood fill* to color in each star when it is first encountered



Solution: use *flood fill* to color in each star when it is first encountered

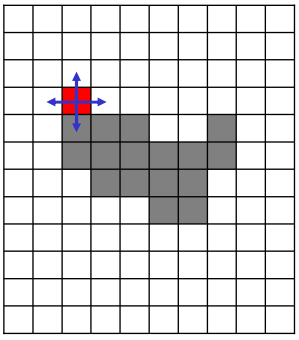
Find an uncolored pixel



Solution: use *flood fill* to color in each star when it is first encountered

Find an uncolored pixel

Look at its neighbors

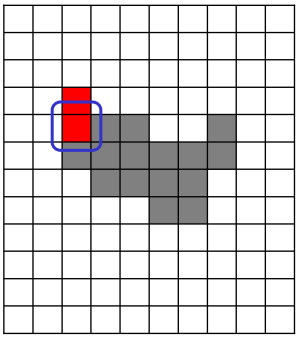


Solution: use *flood fill* to color in each star when it is first encountered

Find an uncolored pixel

Look at its neighbors

For each that needs coloring...



Solution: use *flood fill* to color in each star when it is first encountered

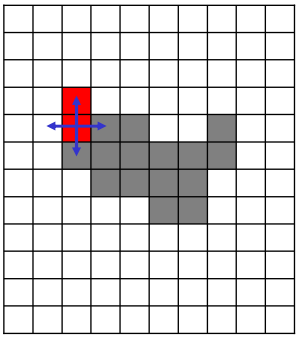
Find an uncolored pixel

Look at its neighbors

For each that needs coloring...

Look at its neighbors, and

for each that needs coloring...



Solution: use *flood fill* to color in each star when it is first encountered

Find an uncolored pixel

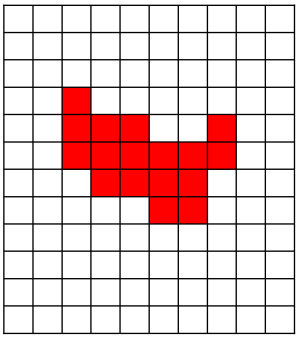
Look at its neighbors

For each that needs coloring...

Look at its neighbors, and

for each that needs coloring...

Stop when whole star colored



Solution: use *flood fill* to color in each star when it is first encountered

Find an uncolored pixel

Look at its neighbors

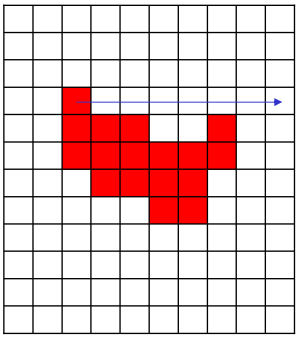
For each that needs coloring...

Look at its neighbors, and

for each that needs coloring...

Stop when whole star colored

Then start scanning again



```
def count(picture):
  xsize, ysize = picture.size
  temp = picture.load()
  result = 0
  for x in range(xsize):
    for y in range(ysize):
      if temp[x, y] == BLACK:
        result += 1
        fill(temp, xsize, ysize, x, y)
  return result
```



```
def count(picture):
  xsize, ysize = picture.size
  temp = picture.load()
  result = 0
  for x in range(xsize):
    for y in range(ysize):
      if temp[x, y] == BLACK:
        result += 1
        fill(temp, xsize, ysize, x, y)
  return result
```

Multimedia Programming Counting Stars

```
def count(picture):
  xsize, ysize = picture.size
  temp = picture.load()
  result = 0
  for x in range(xsize):
    for y in range(ysize):
      if temp[x, y] == BLACK:
        result += 1
        fill(temp, xsize, ysize, x, y)
  return result
```



```
def count(picture):
  xsize, ysize = picture.size
  temp = picture.load()
  result = 0
  for x in range(xsize):
    for y in range(ysize):
      if temp[x, y] == BLACK:
        result += 1
        fill(temp, xsize, ysize, x, y)
  return result
```

Use a *work queue*

Use a *work queue*

Keep list of (x, y) coordinates to be examined

Use a *work queue*

Keep list of (x, y) coordinates to be examined

Loop until queue is empty:

Use a *work queue*

Keep list of (x, y) coordinates to be examined

Loop until queue is empty:

- Take (x, y) coordinates from queue

Use a *work queue*

Keep list of (x, y) coordinates to be examined

Loop until queue is empty:

- Take (x, y) coordinates from queue
- If black, fill it in and add neighbors to queue


```
def fill(pic, xsize, ysize, x_start, y_start):
  queue = [(x_start, y_start)]
  while queue:
    x, y, queue = queue[0][0], queue[0][1], queue[1:]
    if pic[x, y] == BLACK:
      pic[x, y] = RED
      if x > 0:           queue.append((x-1, y))
      if x < (xsize-1):   queue.append((x+1, y))
      if y > 0:           queue.append((x, y-1))
      if y < (ysize-1):   queue.append((x, y+1))
```

```
def fill(pic, xsize, ysize, x_start, y_start):
  queue = [(x_start, y_start)]
  while queue:
    x, y, queue = queue[0][0], queue[0][1], queue[1:]
    if pic[x, y] == BLACK:
      pic[x, y] = RED
      if x > 0:           queue.append((x-1, y))
      if x < (xsize-1):   queue.append((x+1, y))
      if y > 0:           queue.append((x, y-1))
      if y < (ysize-1):   queue.append((x, y+1))
```



```
def fill(pic, xsize, ysize, x_start, y_start):
  queue = [(x_start, y_start)]
  while queue:
    x, y, queue = queue[0][0], queue[0][1], queue[1:]
    if pic[x, y] == BLACK:
      pic[x, y] = RED
      if x > 0:           queue.append((x-1, y))
      if x < (xsize-1):   queue.append((x+1, y))
      if y > 0:           queue.append((x, y-1))
      if y < (ysize-1):   queue.append((x, y+1))
```

```
def fill(pic, xsize, ysize, x_start, y_start):
  queue = [(x_start, y_start)]
  while queue:
    x, y, queue = queue[0][0], queue[0][1], queue[1:]
    if pic[x, y] == BLACK:
      pic[x, y] = RED
      if x > 0:           queue.append((x-1, y))
      if x < (xsize-1):   queue.append((x+1, y))
      if y > 0:           queue.append((x, y-1))
      if y < (ysize-1):   queue.append((x, y+1))
```



```
def fill(pic, xsize, ysize, x_start, y_start):
  queue = [(x_start, y_start)]
  while queue:
    x, y, queue = queue[0][0], queue[0][1], queue[1:]
    if pic[x, y] == BLACK:
      pic[x, y] = RED
      if x > 0:           queue.append((x-1, y))
      if x < (xsize-1):   queue.append((x+1, y))
      if y > 0:           queue.append((x, y-1))
      if y < (ysize-1):   queue.append((x, y+1))
```

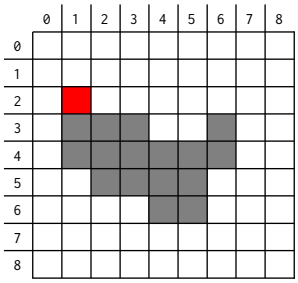
```
def fill(pic, xsize, ysize, x_start, y_start):
  queue = [(x_start, y_start)]
  while queue:
    x, y, queue = queue[0][0], queue[0][1], queue[1:]
    if pic[x, y] == BLACK:
      pic[x, y] = RED
      if x > 0:            queue.append((x-1, y))
      if x < (xsize-1):    queue.append((x+1, y))
      if y > 0:            queue.append((x, y-1))
      if y < (ysize-1):    queue.append((x, y+1))
```



```
def fill(pic, xsize, ysize, x_start, y_start):
  queue = [(x_start, y_start)]
  while queue:
    x, y, queue = queue[0][0], queue[0][1], queue[1:]
    if pic[x, y] == BLACK:
      pic[x, y] = RED
      if x > 0:            queue.append((x-1, y))
      if x < (xsize-1):    queue.append((x+1, y))
      if y > 0:            queue.append((x, y-1))
      if y < (ysize-1):    queue.append((x, y+1))
```

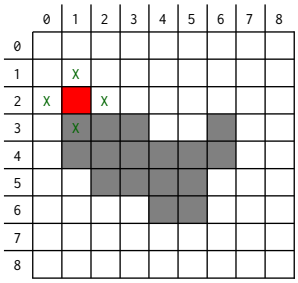
Filling in action

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



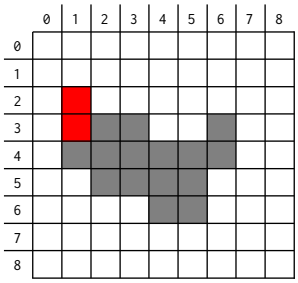
Filling in action

```
[(0, 2),
 (2, 2),
 (1, 1),
 (1, 3)]
```



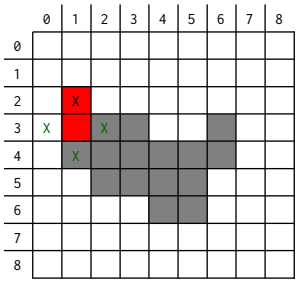
Filling in action

```
[(1, 3)]
```



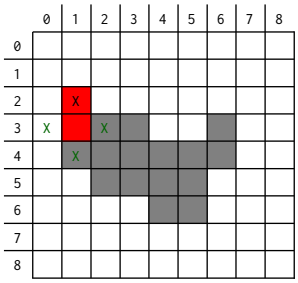
Filling in action

```
[(1, 2),
 (0, 3),
 (2, 3),
 (1, 4)]
```



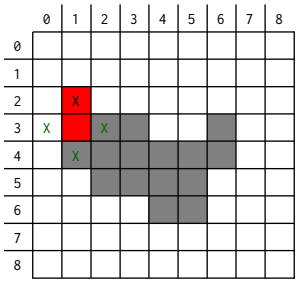
Filling in action

```
[(1, 2),
 (0, 3),
 (2, 3),
 (1, 4)]
```



Filling in action

```
[(1, 2),
 (0, 3),
 (2, 3),
 (1, 4)]
```



Exercise: never look at a pixel twice

Or use a *recursive* algorithm

Or use a *recursive* algorithm

Keep the work to be done on the call stack

Or use a *recursive* algorithm

Keep the work to be done on the call stack

```
def fill(pic, xsize, ysize, x, y):
  if pic[x, y] != BLACK:
    return
  pic[x, y] = RED
  if x > 0:          fill(pic, xsize, ysize, x-1, y)
  if x < (xsize-1):  fill(pic, xsize, ysize, x+1, y)
  if y > 0:          fill(pic, xsize, ysize, x, y-1)
  if y < (ysize-1):  fill(pic, xsize, ysize, x, y+1)
```


Or use a *recursive* algorithm

Keep the work to be done on the call stack

```
def fill(pic, xsize, ysize, x, y):
  if pic[x, y] != BLACK:
    return
  pic[x, y] = RED
  if x > 0:          fill(pic, xsize, ysize, x-1, y)
  if x < (xsize-1):  fill(pic, xsize, ysize, x+1, y)
  if y > 0:          fill(pic, xsize, ysize, x, y-1)
  if y < (ysize-1):  fill(pic, xsize, ysize, x, y+1)
```

Or use a *recursive* algorithm

Keep the work to be done on the call stack

```
def fill(pic, xsize, ysize, x, y):
  if pic[x, y] != BLACK:
    return
  pic[x, y] = RED
  if x > 0:           fill(pic, xsize, ysize, x-1, y)
  if x < (xsize-1):   fill(pic, xsize, ysize, x+1, y)
  if y > 0:           fill(pic, xsize, ysize, x, y-1)
  if y < (ysize-1):   fill(pic, xsize, ysize, x, y+1)
```


Or use a *recursive* algorithm

Keep the work to be done on the call stack

```
def fill(pic, xsize, ysize, x, y):
  if pic[x, y] != BLACK:
    return
  pic[x, y] = RED
  if x > 0:           fill(pic, xsize, ysize, x-1, y)
  if x < (xsize-1):   fill(pic, xsize, ysize, x+1, y)
  if y > 0:           fill(pic, xsize, ysize, x, y-1)
  if y < (ysize-1):   fill(pic, xsize, ysize, x, y+1)
```

Or use a *recursive* algorithm

Keep the work to be done on the call stack

```
def fill(pic, xsize, ysize, x, y):
  if pic[x, y] != BLACK:
    return
  pic[x, y] = RED
  if x > 0:          fill(pic, xsize, ysize, x-1, y)
  if x < (xsize-1):  fill(pic, xsize, ysize, x+1, y)
  if y > 0:          fill(pic, xsize, ysize, x, y-1)
  if y < (ysize-1):  fill(pic, xsize, ysize, x, y+1)
```


Call stack holds pixels currently being examined

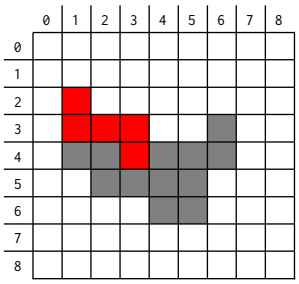
most recent call → `fill(..., 3, 4)`

`fill(..., 3, 3)`

`fill(..., 2, 3)`

`fill(..., 1, 3)`

first call → `fill(..., 1, 2)`



created by

Paul Gries, Jeremy Nickurak, and Greg Wilson

November 2010