Program Design

Invasion Percolation: Resolving Ties

5	3	7	2	6	1	1	3	4
8	5	6	5	7	2	3	6	2
2	5	8	7	5	5	6	5	9
5	2	6	4	2	3	9	6	5
4	6	8	8	−1	2	7	3	9
7	6	4	5	2	8	6	8	5
5	4	2	5	8	4	5	5	8
5	7	5	1	5	3	8	5	5
4	5	1	9	7	8	6	5	1

How to handle the 3-way tie for lowest-valued neighbor?

5	3	7	2	6	1	1	3	4
8	5	6	5	7	2	3	6	2
2	5	8	7	5	5	6	5	9
5	2	6	4	2	3	9	6	5
4	6	8	8	-1	2	7	3	9
7	6	4	5	2	8	6	8	5
5	4	2	5	8	4	5	5	8
5	7	5	1	5	3	8	5	5
4	5	1	9	7	8	6	5	1

We're supposed to "choose one at random"

5	3	7	2	6	1	1	3	4
8	5	6	5	7	2	3	6	2
2	5	8	7	5	5	6	5	9
5	2	6	4	2	3	9	6	5
4	6	8	8	–1	2	7	3	9
7	6	4	5	2	8	6	8	5
5	4	2	5	8	4	5	5	8
5	7	5	1	5	3	8	5	5
4	5	1	9	7	8	6	5	1

We're supposed to "choose one at random"

But how do we keep track of the tied cells that we're supposed to choose from?

- Use a *set* of (x, y) coordinates to track cells

- Use a *set* of (x, y) coordinates to track cells
- And record the value stored in those cells

- Use a *set* of (x, y) coordinates to track cells
- And record the value stored in those cells
- Three cases to consider:

- Use a *set* of (x, y) coordinates to track cells
- And record the value stored in those cells
- Three cases to consider:

New value > current minimum	Ignore

- Use a *set* of (x, y) coordinates to track cells
- And record the value stored in those cells
- Three cases to consider:

New value > current minimum	Ignore
New value == current minimum	Add this cell to set

- Use a *set* of (x, y) coordinates to track cells
- And record the value stored in those cells
- Three cases to consider:

New value > current minimum	Ignore
New value == current minimum	Add this cell to set
New value < current minimum	Empty the set, then put this cell in it

5	4	7	2
4	5	6	5
2	3	3	7

```
# Z was 10
min_val = 11
min_set = {}
```

5	4	7	2
4	5	6	5
2	3	3	7

```
# 4 < 11
min_val = 4
min_set = {(12,23)}
```

5	4	7	2
4	5	6	5
2	3	3	7

```
# 7 > 4, so no change
min_val = 4
min_set = {(12,23)}
```

5	4	7	2
4	5	6	5
2	3	3	7

```
# 4 == 4, so add to set
min_val = 4
min_set = {(12,23), (11,22)}
```

5	4	7	2
4	5	6	5
2	3	3	7

```
# 5 > 4, so no change
min_val = 4
min_set = {(12,23), (11,22)}
```

5	4	7	2
4	5	6	5
2	3	3	7

```
# 3 < 4, so re-set
min_val = 3
min_set = {(12,21)}
```

5	4	7	2
4	5	6	5
2	3	3	7

```
# 3 == 3, so add to set
min_val = 3
min_set = {(12,21), (13,21)}
```

```
# Keep track of cells tied for lowest value
min_val = Z+1
min_set = set()
for x in range(N):
  for y in range(N):
    if ...is a neighbor...:
      if grid[x][y] == min_val:
        min_set.add((x, y))
      elif grid[x][y] < min_val:
        min_val = grid[x][y]
        min_set = set([(x, y)])
```

```
# Keep track of cells tied for lowest value
min_val = Z+1
min_set = set()
for x in range(N):
  for y in range(N):
    if ...is a neighbor...:
      if grid[x][y] == min_val:
        min_set.add((x, y))
      elif grid[x][y] < min_val:
        min_val = grid[x][y]
        min_set = set([(x, y)])
```

All actual grid values are less than this

```
# Keep track of cells tied for lowest value
min_val = Z+1
min_set = set()
for x in range(N):
  for y in range(N):
    if ...is a neighbor...:
      if grid[x][y] == min_val:
        min_set.add((x, y))
      elif grid[x][y] < min_val:
        min_val = grid[x][y]
        min_set = set([(x, y)])
```

Coordinates of all neighbors whose value is min_val

```
# Keep track of cells tied for lowest value
min_val = Z+1
min_set = set()
for x in range(N):
  for y in range(N):
    if ...is a neighbor...:
      if grid[x][y] == min_val:
        min_set.add((x, y))
      elif grid[x][y] < min_val:
        min_val = grid[x][y]
        min_set = set([(x, y)])
```

Only look at neighbors

```
# Keep track of cells tied for lowest value
min_val = Z+1
min_set = set()
for x in range(N):
  for y in range(N):
    if ...is a neighbor...:
      if grid[x][y] == min_val:
        min_set.add((x, y))
      elif grid[x][y] < min_val:
        min_val = grid[x][y]
        min_set = set([(x, y)])
```

Do nothing for Case 1 (new cell's value greater than current minimum)...

```
# Keep track of cells tied for lowest value
min_val = Z+1
min_set = set()
for x in range(N):
  for y in range(N):
    if ...is a neighbor...:
      if grid[x][y] == min_val:
        min_set.add((x, y))
      elif grid[x][y] < min_val:
        min_val = grid[x][y]
        min_set = set([(x, y)])
```

Do nothing for Case 1 (new cell's value greater than current minimum)...

...because there's nothing to do

```
# Keep track of cells tied for lowest value
min_val = Z+1
min_set = set()
for x in range(N):
  for y in range(N):
    if ...is a neighbor...:
      if grid[x][y] == min_val:
        min_set.add((x, y))
      elif grid[x][y] < min_val:
        min_val = grid[x][y]
        min_set = set([(x, y)])
```

Case 2: add another cell to the current set of candidates

```
# Keep track of cells tied for lowest value
min_val = Z+1
min_set = set()
for x in range(N):
  for y in range(N):
    if ...is a neighbor...:
      if grid[x][y] == min_val:
        min_set.add((x, y))
      elif grid[x][y] < min_val:
        min_val = grid[x][y]
        min_set = set([(x, y)])
```

Case 3: a new minimum,

so re-start the set

```
# Keep track of cells tied for lowest value
min_val = Z+1
min_set = set()
for x in range(N):
  for y in range(N):
    if ...is a neighbor...:
      if grid[x][y] == min_val:
        min_set.add((x, y))
      elif grid[x][y] < min_val:
        min_val = grid[x][y]
        min_set = set([(x, y)])
```

All actual grid values are less than this

This case runs the first time an actual cell is examined

```
# Choose a cell
from random import ...,
                    choice

min_val = Z+1
min_set = set()
...loop...
assert min_set, "No cells found"
candidates = list(min_set)
x, y = choice(candidates)
```

Use the `random` library again

software carpentry

```
# Choose a cell
from random import ...,
                   choice

min_val = Z+1
min_set = set()
...loop...
assert min_set, "No cells found"
candidates = list(min_set)
x, y = choice(candidates)
```

Fail early, often, and loudly

```
# Choose a cell
from random import ...,
                   choice


min_val = Z+1
min_set = set()
...loop...
assert min_set, "No cells found"
candidates = list(min_set)
x, y = choice(candidates)
```

Because `choice`

needs something

indexable

```
# Choose a cell
from random import ...,
                   choice

min_val = Z+1
min_set = set()
...loop...
assert min_set, "No cells found"
candidates = list(min_set)
x, y = choice(candidates)
```

Choose one

created by

Greg Wilson

2010-05-19