Program Design

Invasion Percolation: Resolving Ties

Copyright © Software Carpentry 2010
This work is licensed under the Creative Commons Attribution License
See http://software-carpentry.org/license.html for more information.
How to handle the 3-way tie for lowest-valued neighbor?
We're supposed to "choose one at random"
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?

<table>
<thead>
<tr>
<th>5</th>
<th>3</th>
<th>7</th>
<th>2</th>
<th>6</th>
<th>1</th>
<th>1</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>-1</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
- 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 \(\Rightarrow\) Ignore

  New value == current minimum \(\Rightarrow\) 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
# Z was 10

min_val = 11

min_set = {}
# 4 < 11

\[
\text{min_val} = 4 \\
\text{min_set} = \{(12,23)\}
\]
<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>4</th>
<th>7</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

# 7 > 4, so no change

min_val = 4

min_set = \{(12,23)\}
# 4 == 4, so add to set
min_val = 4
min_set = {(12,23), (11,22)}
# 5 > 4, so no change
min_val = 4
min_set = {(12, 23), (11, 22)}
# 3 < 4, so re-set

$$\text{min\_val} = 3$$

$$\text{min\_set} = \{(12, 21)\}$$
# 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)])
# 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

```python
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)...

Program Design
Invasion Percolation
Resolving Ties
# 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

```python
min_val = Z+1
min_set = set()
```

```python
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)
# 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