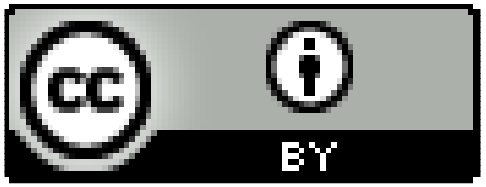
Program Design

Invasion Percolation: Bugs



Let's try running the program we just created

Let's try running the program we just created

```
$ python invperc.py 3 10 17983
2 cells filled
```

Let's try running the program we just created

```
$ python invperc.py 3 10 17983
2 cells filled
```

Excellent: 2 cells *should* be filled in a 3×3 grid

Let's try running the program we just created

```
$ python invperc.py 3 10 17983
2 cells filled
```

Excellent: 2 cells *should* be filled in a 3×3 grid

Let's try a larger grid

Let's try running the program we just created

```
$ python invperc.py 3 10 17983
2 cells filled
```

Excellent: 2 cells *should* be filled in a 3×3 grid

Let's try a larger grid

```
$ python invperc.py 5 10 27187
```

Let's try running the program we just created

```
$ python invperc.py 3 10 17983
2 cells filled
```

Excellent: 2 cells *should* be filled in a 3×3 grid

Let's try a larger grid

```
$ python invperc.py 5 10 27187
...a minute passes...
```

Let's try running the program we just created

```
$ python invperc.py 3 10 17983
2 cells filled
```

Excellent: 2 cells *should* be filled in a 3×3 grid

Let's try a larger grid

```
$ python invperc.py 5 10 27187
...a minute passes...
ctrl-C
```

Time to fire up the debugger...

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

The initial grid looks right

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

The initial grid looks right

Still looks good after filling the middle cell

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

The initial grid looks right

Still looks good after filling the middle cell

Remember, we're using -1 to mark filled cells

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

The initial grid looks right

Still looks good after filling the middle cell

Next cell filled correctly

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

The initial grid looks right

Still looks good after filling the middle cell

Next cell filled correctly

Then the program goes into an infinite loop

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

The initial grid looks right

Still looks good after filling the middle cell

Next cell filled correctly

Then the program goes into an infinite loop

In the `find_candidates` function

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

The initial grid looks right

Still looks good after filling

the middle cell

Next cell filled correctly

Then the program goes into an infinite loop

In the `find_candidates` function

min_set == {(2, 2), (1, 2)}

min_val == -1

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

The initial grid looks right

Still looks good after filling the middle cell

Next cell filled correctly

Then the program goes into an infinite loop

In the `find_candidates` function

min_set == {(2, 2), (1, 2)}

min_val == -1 ← Uh oh...

Our marker value is less than all the actual values in the grid...

Our marker value is less than all the actual values in the grid...

...and once we have marked two cells...

Our marker value is less than all the actual values in the grid...

...and once we have marked two cells...

...each of those marked cells is adjacent to a marked cell

Our marker value is less than all the actual values in the grid...

...and once we have marked two cells...

...each of those marked cells is adjacent to a marked cell

At least it's easy to fix

Old (buggy) code

```
def find_candidates(grid):
  N = len(grid)
  min_val = sys.maxint
  min_set = set()
  for x in range(N):
    for y in range(N):
      if is_candidate(grid, x, y):
      ...handle == min_val and < min_val cases...
```

New (correct) code

```
def find_candidates(grid):
  N = len(grid)
  min_val = sys.maxint
  min_set = set()
  for x in range(N):
    for y in range(N):
      if grid[x][y] == FILLED:
        continue # skip to next cell
      if is_candidate(grid, x, y):
      ...handle == min_val and < min_val cases...
```

Great—we found one bug

Great—we found one bug

How many others *haven't* we found?

Great—we found one bug

How many others *haven't* we found?

How do we *validate* and *verify* this program?

"If x is either 0 or N-1, or y is either 0 or N-1"

```
num_filled += 1
if x in (0, N-1) or y in (0, N-1):
  break
```

"If x is either 0 or N-1, or y is either 0 or N-1"

I.e., if either coordinate is on the grid's edge

```
num_filled += 1
if x in (0, N-1) or y in (0, N-1):
  break
```

Sound similar, but fails

```
num_filled += 1
if x is (0, N-1) or y is (0, N-1):
  break
```

Sound similar, but fails

"If x is the tuple (0, N-1) or y is the tuple (0, N-1)"

```
num_filled += 1
if x is (0, N-1) or y is (0, N-1):
  break
```

Sound similar, but fails

"If x is the tuple (0, N-1) or y is the tuple (0, N-1)"

Neither x nor y will ever be a two-valued tuple

```
num_filled += 1
if x is (0, N-1) or y is (0, N-1):
  break
```

Sound similar, but fails

"If x is the tuple (0, N-1) or y is the tuple (0, N-1)"

Neither x nor y will ever be a two-valued tuple

So the loop will never exit

```
num_filled += 1
if x is (0, N-1) or y is (0, N-1):
  break
```

Sounds like what we'd say to another person

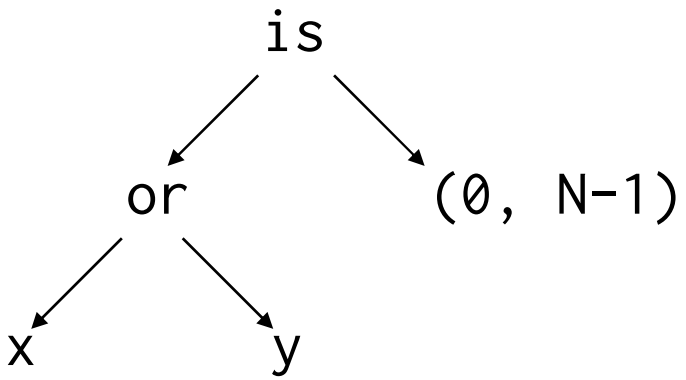
```
num_filled += 1
if x or y is (0, N-1):
  break
```

Sounds like what we'd say to another person

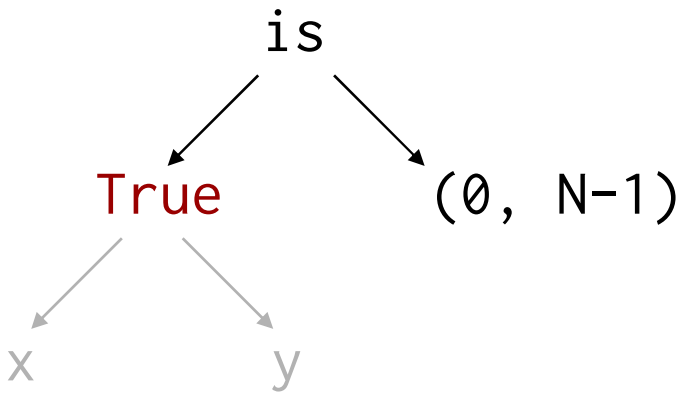
But how will the computer interpret it?

```
num_filled += 1
if x or y is (0, N-1):
  break
```

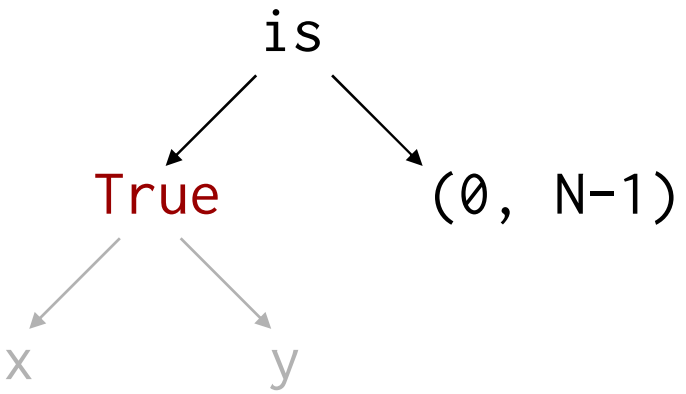
Interpretation #1



Interpretation #1

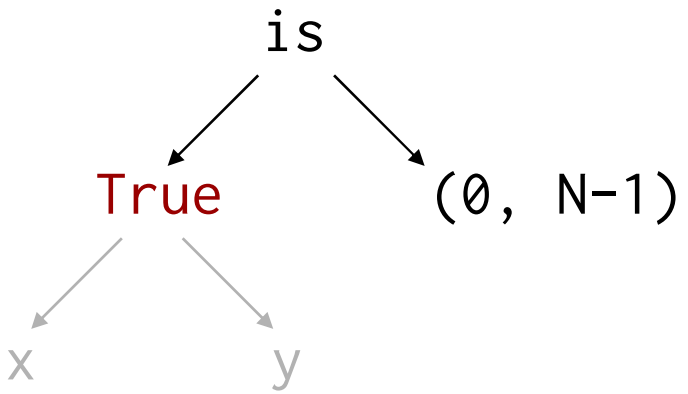


Interpretation #1



True isn't a two-integer tuple

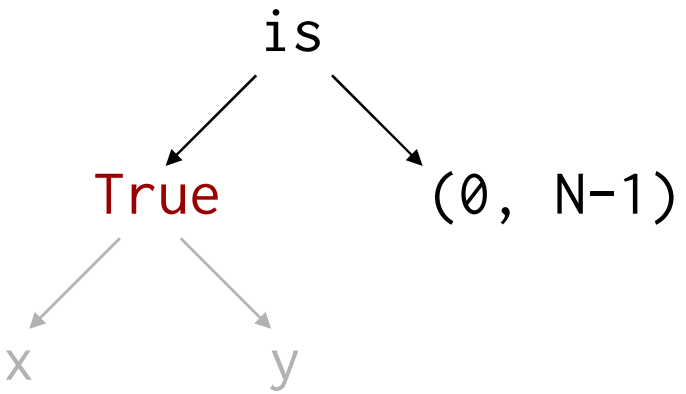
Interpretation #1



True isn't a two-integer tuple

(Neither is False)

Interpretation #1

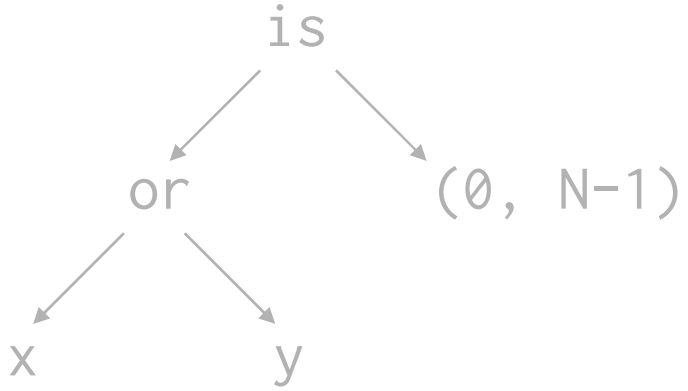


True isn't a two-integer tuple

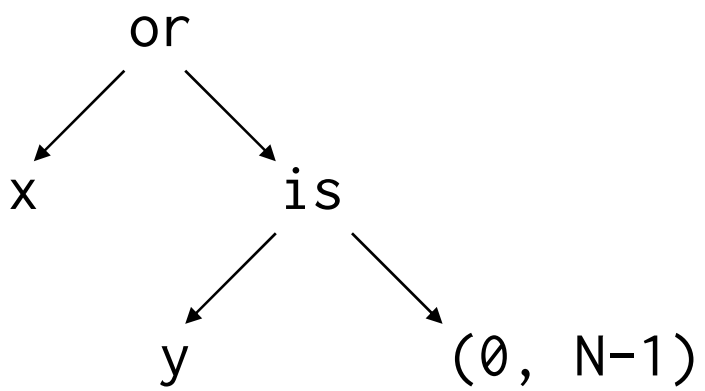
(Neither is False)

So this definitely isn't right

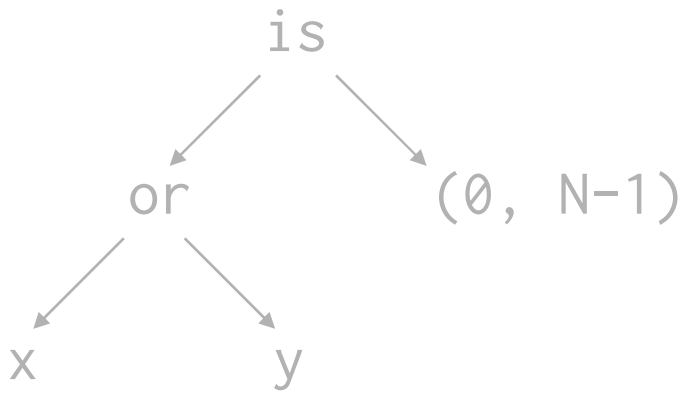
Interpretation #1



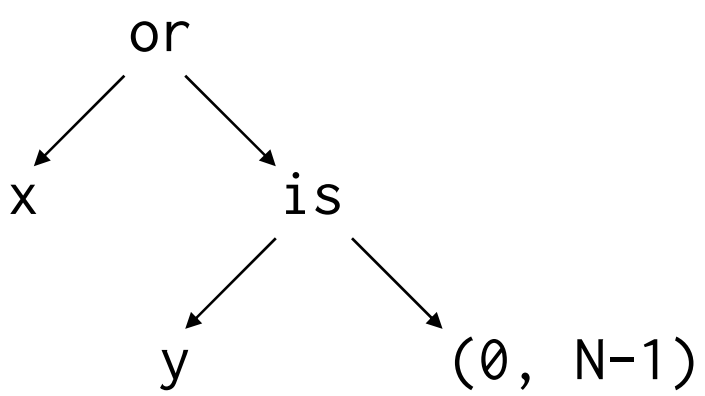
Interpretation #2



Interpretation #1

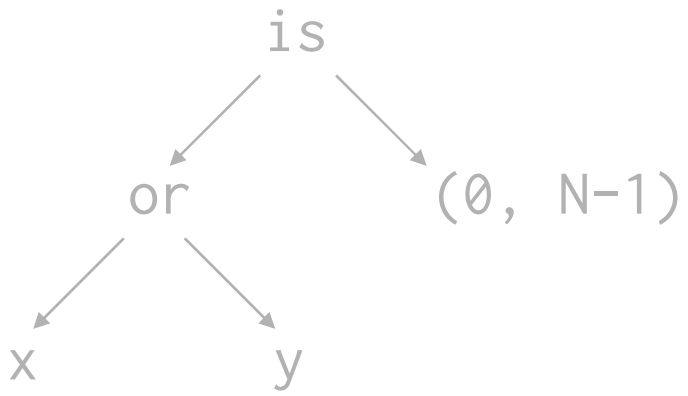


Interpretation #2

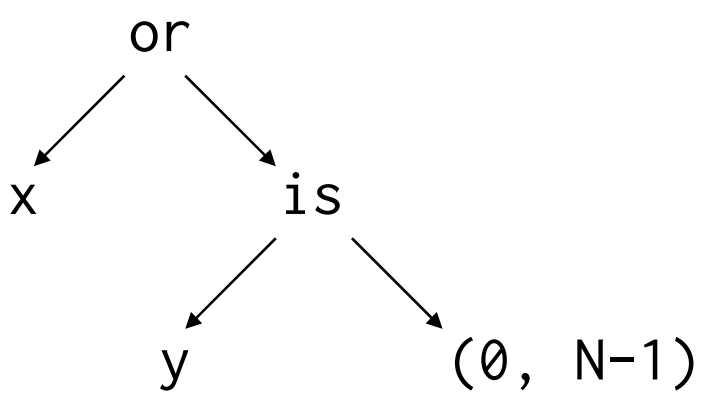


y is never a two-integer tuple

Interpretation #1



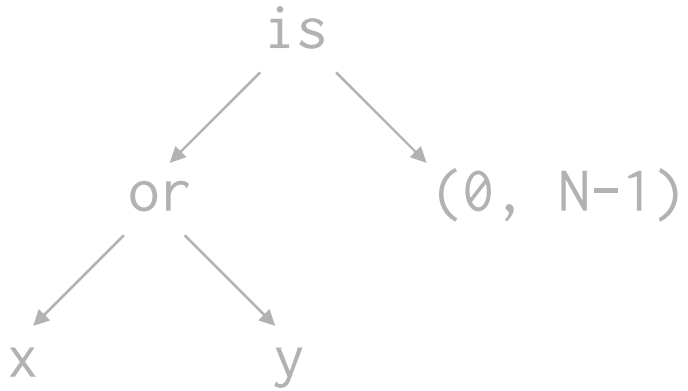
Interpretation #2



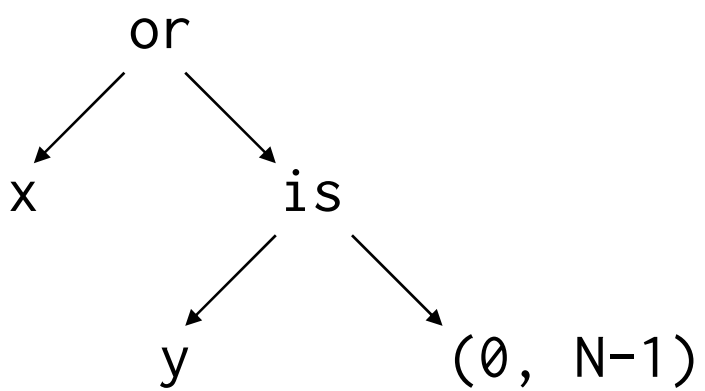
y is never a two-integer tuple

So this is just "x or False"

Interpretation #1



Interpretation #2

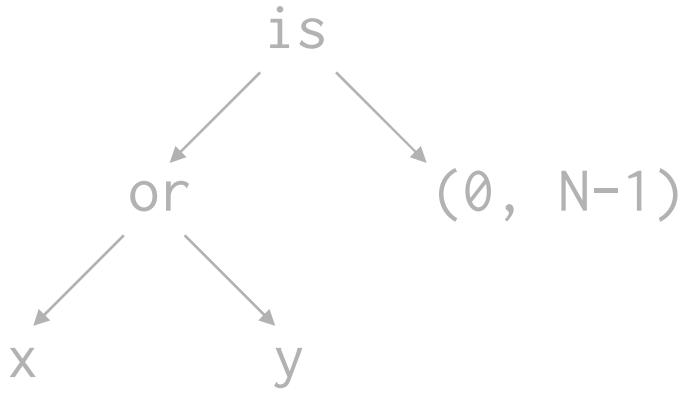


y is never a two-integer tuple

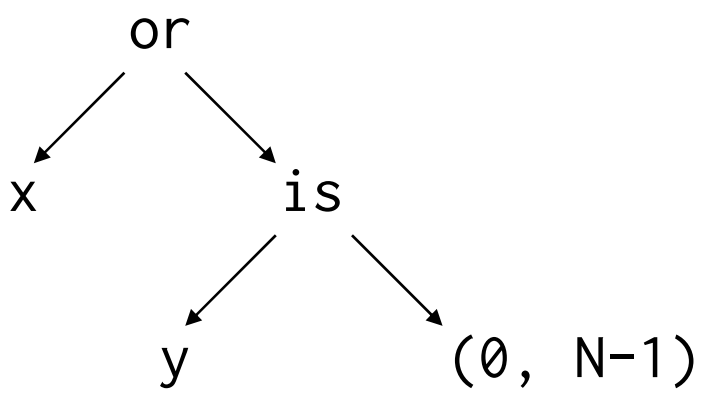
So this is just "x or False"

Which is just "x is not 0"

Interpretation #1



Interpretation #2



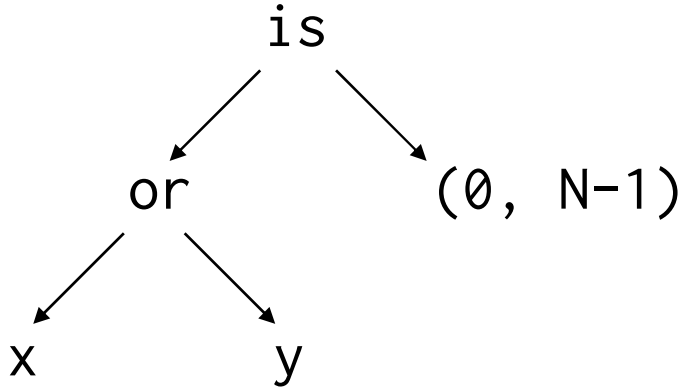
y is never a two-integer tuple

So this is just "x or False"

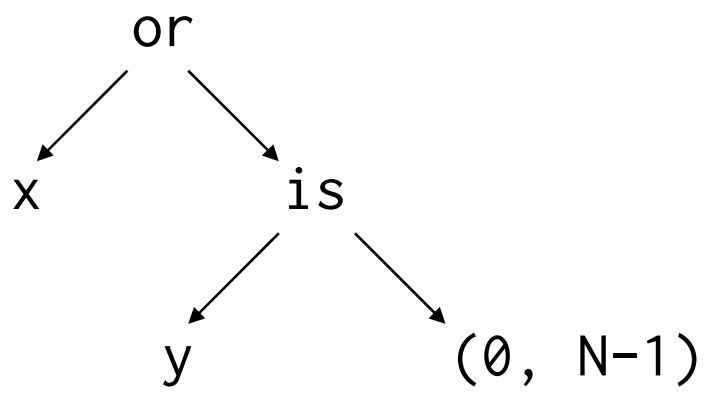
Which is just "x is not 0"

Not what we want either

Interpretation #1

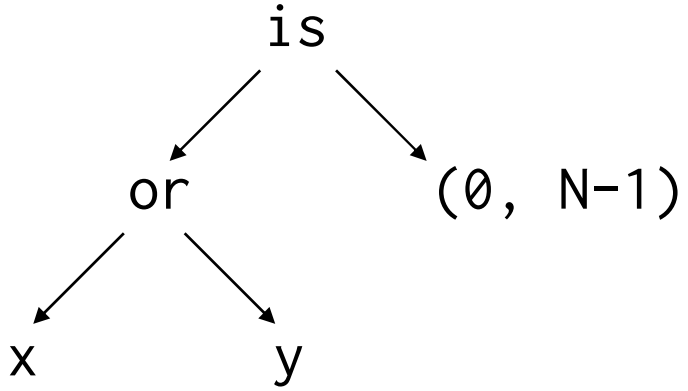


Interpretation #2

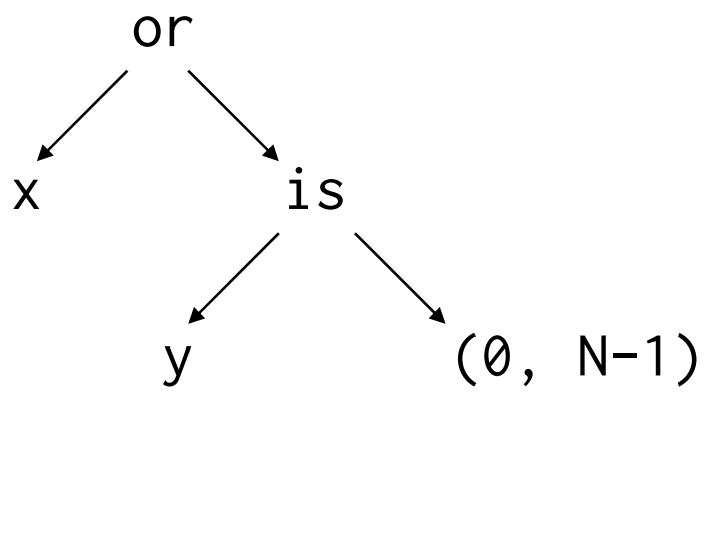


This is what Python actually does

Interpretation #1



Interpretation #2



This is what Python actually does

"a or b is c" binds like "x + y * z"

created by

Greg Wilson

May 2010