An evil-genius guide to computer programming

Daniel Lemire

Montreal 🇨🇦

blog: https://lemire.me

twitter: @lemire

GitHub: https://github.com/lemire/

Week 5

Creative Programming

Bitwise operations

- AND: `a & b` : 1 if and only if both `a` and `b` are 1
- OR: `a | b` : 0 if and only if both `a` and `b` are 0
- XOR: `a ^ b` : 0 if and only if both `a` and `b` are the same
- ANDNOT: `a &~ b` : 1 if and only if (`a` is 1 and `b` is 0)

Bitset

- One bit per binary attribute
- E.g., 1 bit for female/male
- E.g., 1 bit for citizen/non-citizen
- so forth

- A room can have a door on the North wall (or not)
- A room can have a door on the South wall (or not)
- A room can have a door on the East wall (or not)
- A room can have a door on the West wall (or not)

```
room = {"North": True, "South": False, "East": True, "West": False}
```

Do it in four bits

```
North = 1
South = 2
East = 4
West = 8
room = North | East
```

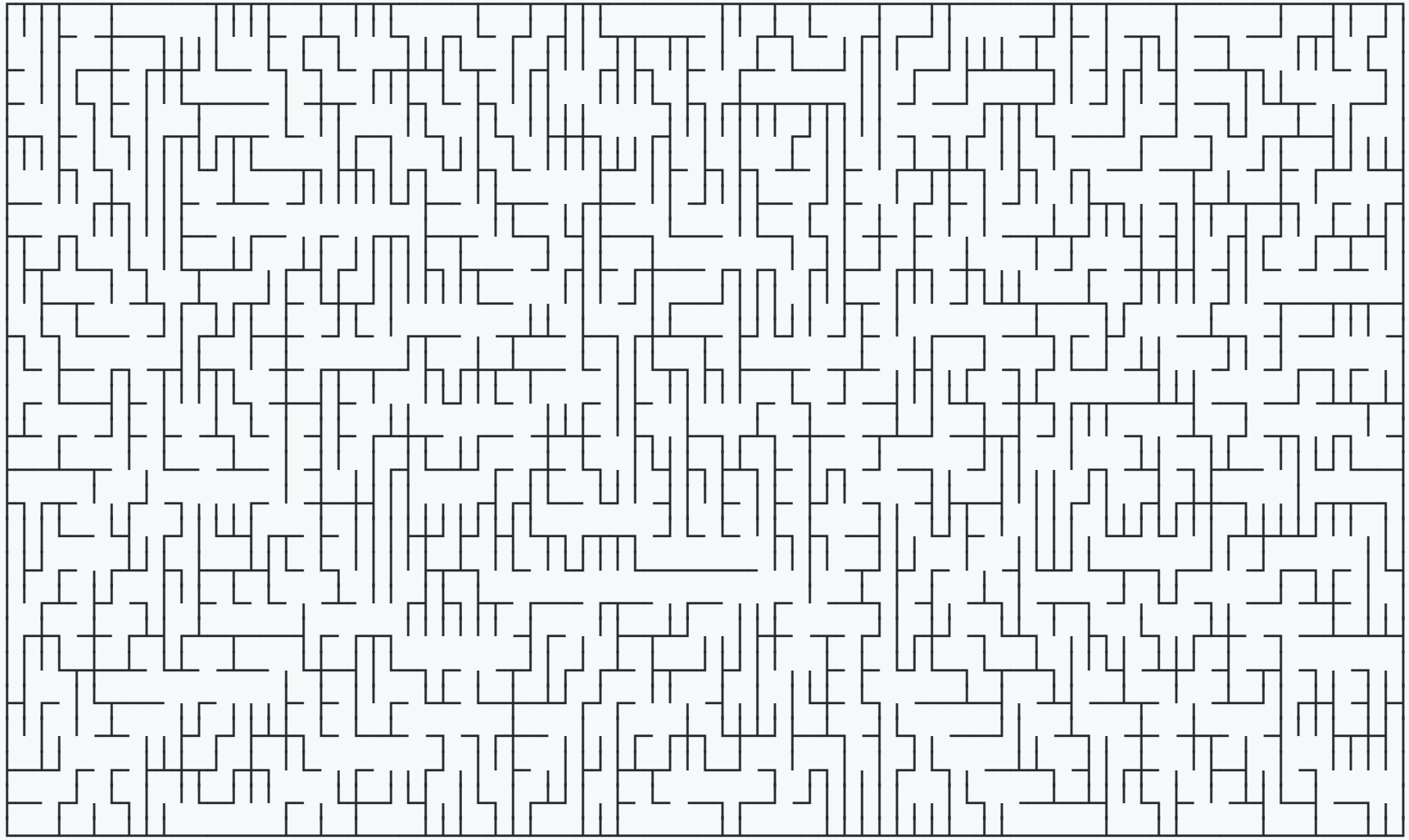
```
room |= North
```

```
room & North == North
```

```
room ^= North
```

```
room &= ~North
```

Create a maze (Prim's algorithm)



```
UNVISITED = 0
UP = 1
DOWN = 2
LEFT = 4
RIGHT = 8
```

```
opposite = {
    UP : DOWN,
    DOWN : UP,
    LEFT : RIGHT,
    RIGHT : LEFT,
}
```

```
array = [[UNVISITED for i in range(number_of_columns)] for j in range(number_of_rows)]
```

```
import random
initial_location = (random.randint(0,number_of_rows-1), random.randint(0,number_of_columns-1))
```

```
removable = []
```

```
def add_removable_walls(location):
    if(location[0] > 0):
        if(array[location[0] - 1][location[1]] == UNVISITED):
            removable.append((location,UP))
...
```

```
def add_removable_walls(location):
    if(location[0] > 0):
        if(array[location[0] - 1][location[1]] == UNVISITED):
            removable.append((location,UP))
    if(location[0] < number_of_rows - 1):
        if(array[location[0] + 1][location[1]] == UNVISITED):
            removable.append((location,DOWN))
    if(location[1] > 0):
        if(array[location[0]][location[1]-1] == UNVISITED):
            removable.append((location,LEFT))
    if(location[1] < number_of_columns - 1):
        if(array[location[0]][location[1]+1] == UNVISITED):
            removable.append((location,RIGHT))
```

```
add_removable_walls(initial_location)
```

```
room, direction = random.choice(removable)
removable.remove((room, direction))
```

```
move = {
    LEFT : (0,-1),
    RIGHT : (0,1),
    UP : (-1,0),
    DOWN : (1,0),
}
```

```
m = move[direction]
target = (room[0]+m[0], room[1]+m[1])
```

```
array[initial_location[0]][initial_location[1]] |= direction
array[target[0]][target[1]] |= opposite[direction]
add_removable_walls(target)
```

```
while (len(removable)>0):
    ...
```

```
while (len(removable)>0):
    room, direction = random.choice(removable)
    removable.remove((room, direction))
    ...
```

```
while (len(removable)>0):
    room, direction = random.choice(removable)
    removable.remove((room, direction))
    # check that it is still a good choice
    m = move[direction]
    target = (room[0]+m[0], room[1]+m[1])
    ...
```

```
while (len(removable)>0):
    room, direction = random.choice(removable)
    removable.remove((room, direction))
    # check that it is still a good choice
    m = move[direction]
    target = (room[0]+m[0], room[1]+m[1])
    if array[target[0]][target[1]] != UNVISITED:
        continue
```

```
while (len(removable)>0):
    room, direction = random.choice(removable)
    removable.remove((room, direction))
    # check that it is still a good choice
    m = move[direction]
    target = (room[0]+m[0], room[1]+m[1])
    if array[target[0]][target[1]] != UNVISITED:
        continue
    array[room[0]][room[1]] |= direction
    array[target[0]][target[1]] |= opposite[direction]
```

```
while (len(removable)>0):
    room, direction = random.choice(removable)
    removable.remove((room, direction))
    # check that it is still a good choice
    m = move[direction]
    target = (room[0]+m[0], room[1]+m[1])
    if array[target[0]][target[1]] != UNVISITED:
        continue
    array[room[0]][room[1]] |= direction
    array[target[0]][target[1]] |= opposite[direction]
    add_removable_walls(target)
```

```
character = [' ' for i in range(16)]
```

```
character = [' ' for i in range(16)]
character[LEFT] = '-'
character[RIGHT] = '-'
character[UP] = '|'
character[DOWN] = '|'
```

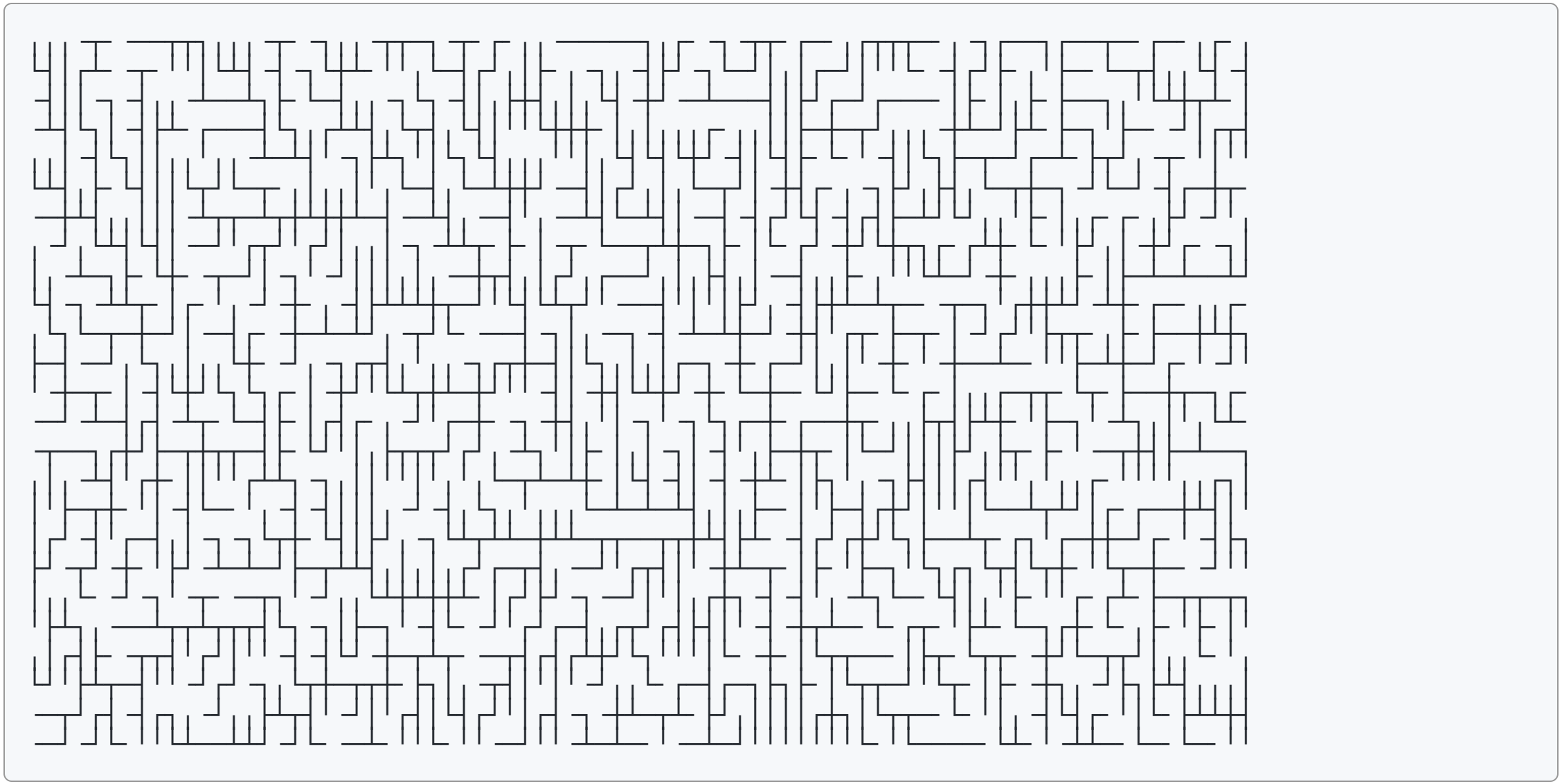
```
character[UP|RIGHT] = '└'
character[UP|LEFT] = '┘'
character[DOWN|RIGHT] = '┌'
character[DOWN|LEFT] = '┐'
character[UP|DOWN] = '│'
character[LEFT|RIGHT] = '─'
```

```
character[LEFT|RIGHT|UP] = '┴'
character[LEFT|RIGHT|DOWN] = '┬'
character[RIGHT|DOWN|UP] = '├'
character[LEFT|DOWN|UP] = '┤'

character[LEFT|RIGHT|DOWN|UP] = '┼'
```

```
def print_map(array):
    for row in array:
        print("".join([character[x] for x in row]))

print_map(array)
```



```
wall_array = [[UP|DOWN|LEFT|RIGHT for i in range(1+number_of_columns)] for j in range(1+number_of_rows)]
```

```
for r in range(0,number_of_rows+1):
    wall_array[r][0] ^= LEFT
    wall_array[r][number_of_columns] ^= RIGHT
```

```
for c in range(0,number_of_columns+1):
    wall_array[0][c] ^= UP
    wall_array[number_of_rows][c] ^= DOWN
```

```
def in_range(r,c):
    return r>=0 and c >=0 and r < number_of_rows and c < number_of_columns
```

```
for r in range(number_of_rows+1):
    for c in range(number_of_columns+1):
        if in_range(r,c):
            if (array[r][c] & UP) == UP :
                wall_array[r][c] &= ~RIGHT
            if (array[r][c] & LEFT) == LEFT :
                wall_array[r][c] &= ~DOWN
        if in_range(r-1,c-1):
            if (array[r-1][c-1] & DOWN) == DOWN :
                wall_array[r][c] &= ~LEFT
            if (array[r-1][c-1] & RIGHT) == RIGHT :
                wall_array[r][c] &= ~UP
        if in_range(r-1,c):
            if (array[r-1][c] & DOWN) == DOWN :
                wall_array[r][c] &= ~RIGHT
            if (array[r-1][c] & LEFT) == LEFT :
                wall_array[r][c] &= ~UP
        if in_range(r,c-1):
            if (array[r][c-1] & UP) == UP :
                wall_array[r][c] &= ~LEFT
            if (array[r][c-1] & RIGHT) == RIGHT :
                wall_array[r][c] &= ~DOWN
```

```
for r in range(number_of_rows+1):
    for c in range(number_of_columns+1):
        if in_range(r,c):
            if (array[r][c] & UP) == UP :
                wall_array[r][c] &= ~RIGHT
            if (array[r][c] & LEFT) == LEFT :
                wall_array[r][c] &= ~DOWN
        if in_range(r-1,c-1):
            if (array[r-1][c-1] & DOWN) == DOWN :
                wall_array[r][c] &= ~LEFT
            if (array[r-1][c-1] & RIGHT) == RIGHT :
                wall_array[r][c] &= ~UP
        if in_range(r-1,c):
            if (array[r-1][c] & DOWN) == DOWN :
                wall_array[r][c] &= ~RIGHT
            if (array[r-1][c] & LEFT) == LEFT :
                wall_array[r][c] &= ~UP
        if in_range(r,c-1):
            if (array[r][c-1] & UP) == UP :
                wall_array[r][c] &= ~LEFT
            if (array[r][c-1] & RIGHT) == RIGHT :
                wall_array[r][c] &= ~DOWN
```

```
for r in range(number_of_rows+1):
    for c in range(number_of_columns+1):
        if in_range(r,c):
            if (array[r][c] & UP) == UP :
                wall_array[r][c] &= ~RIGHT
            if (array[r][c] & LEFT) == LEFT :
                wall_array[r][c] &= ~DOWN
    ...
```

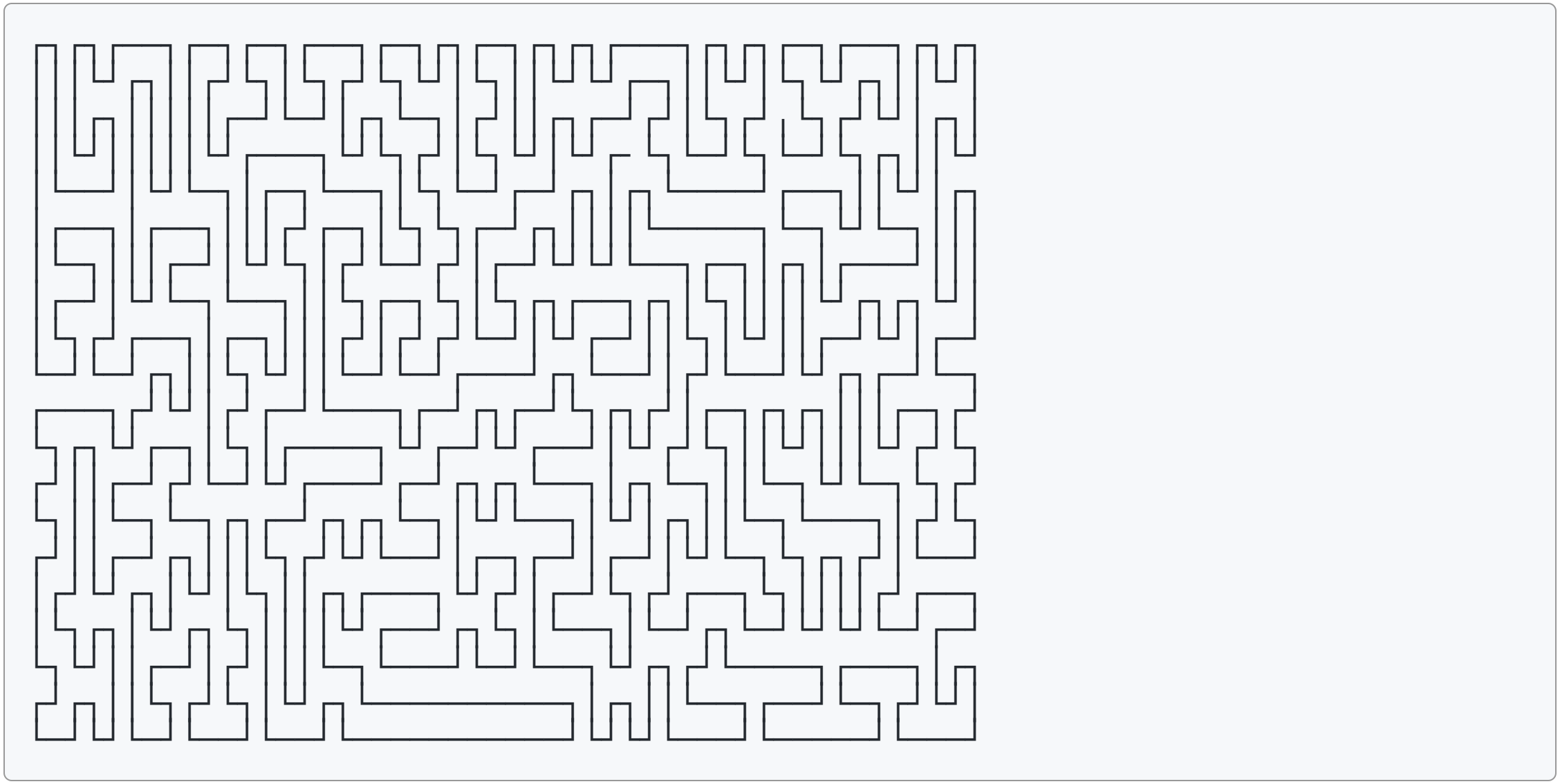
```
for r in range(number_of_rows+1):
    for c in range(number_of_columns+1):
        if in_range(r,c):
            if (array[r][c] & UP) == UP :
                wall_array[r][c] &= ~RIGHT
            if (array[r][c] & LEFT) == LEFT :
                wall_array[r][c] &= ~DOWN
        if in_range(r-1,c-1):
            if (array[r-1][c-1] & DOWN) == DOWN :
                wall_array[r][c] &= ~LEFT
            if (array[r-1][c-1] & RIGHT) == RIGHT :
                wall_array[r][c] &= ~UP
        ...
```

```
for r in range(number_of_rows+1):
    for c in range(number_of_columns+1):
        if in_range(r,c):
            if (array[r][c] & UP) == UP :
                wall_array[r][c] &= ~RIGHT
            if (array[r][c] & LEFT) == LEFT :
                wall_array[r][c] &= ~DOWN
        if in_range(r-1,c-1):
            if (array[r-1][c-1] & DOWN) == DOWN :
                wall_array[r][c] &= ~LEFT
            if (array[r-1][c-1] & RIGHT) == RIGHT :
                wall_array[r][c] &= ~UP
        if in_range(r-1,c):
            if (array[r-1][c] & DOWN) == DOWN :
                wall_array[r][c] &= ~RIGHT
            if (array[r-1][c] & LEFT) == LEFT :
                wall_array[r][c] &= ~UP
        if in_range(r,c-1):
            if (array[r][c-1] & UP) == UP :
                wall_array[r][c] &= ~LEFT
            if (array[r][c-1] & RIGHT) == RIGHT :
                wall_array[r][c] &= ~DOWN
```

Maze (working example)

https://replit.com/@lemire/DemandingLowFinance#main.py

Space-filling curve (Hamiltonian path)



```
def generate_row(direction):
    r = [ LEFT | RIGHT for col in range(number_of_columns) ]
    if direction == RIGHT:
        r[0] = DOWN | RIGHT
        r[number_of_columns -1] = LEFT | UP
    elif direction == LEFT:
        r[0] = RIGHT | UP
        r[number_of_columns - 1] = LEFT | DOWN
    return r
```

```
array = []
for row in range(number_of_rows):
    if (row & 1) == 0:
        array.append(generate_row(LEFT))
    else:
        array.append(generate_row(RIGHT))
```

```
array[0][0] = RIGHT

dangling = [(0,0)]
```

```
if (number_of_rows & 1) == 0:
    array[number_of_rows-1][0] = RIGHT
    dangling.append((number_of_rows-1,0))
else:
    array[number_of_rows-1][number_of_columns-1] = LEFT
    dangling.append((number_of_rows-1,number_of_columns-1))
```

```
def modify():
    pick = random.choice([0,1])
```

```
def modify():
    pick = random.choice([0,1])
    x,y =  dangling[pick]
    value = array[x][y]
```

```
    possibilities = []
    if x < number_of_rows - 1 and value != DOWN:
        possibilities.append(DOWN)
    if x > 0 and value != UP:
        possibilities.append(UP)
    if y < number_of_columns - 1 and value != RIGHT:
        possibilities.append(RIGHT)
    if y > 0 and value != LEFT:
        possibilities.append(LEFT)
    newdirection = random.choice(possibilities)
```

```
    newdirection = random.choice(possibilities)
    initial_direction = array[x][y]
    array[x][y] = newdirection | initial_direction
```

```
    m = move[newdirection]
    newpointer = (x+m[0],y+m[1])
    array[x+m[0]][y+m[1]] |= opposite[newdirection]
```

```
dangling[pick] = disconnect((x,y), newpointer, initial_direction)
```

```
def disconnect(startpoint, endpoint, direction):
    p = startpoint
    m = move[direction]
    p = (p[0]+m[0], p[1]+m[1])
    while p != endpoint:
        direction = array[p[0]][p[1]] ^ opposite[direction]
        m = move[direction]
        p = (p[0]+m[0], p[1]+m[1])
    array[p[0]][p[1]] ^= opposite[direction]
    array[p[0]-m[0]][p[1]-m[1]] ^= direction
    return (p[0]-m[0], p[1]-m[1])
```

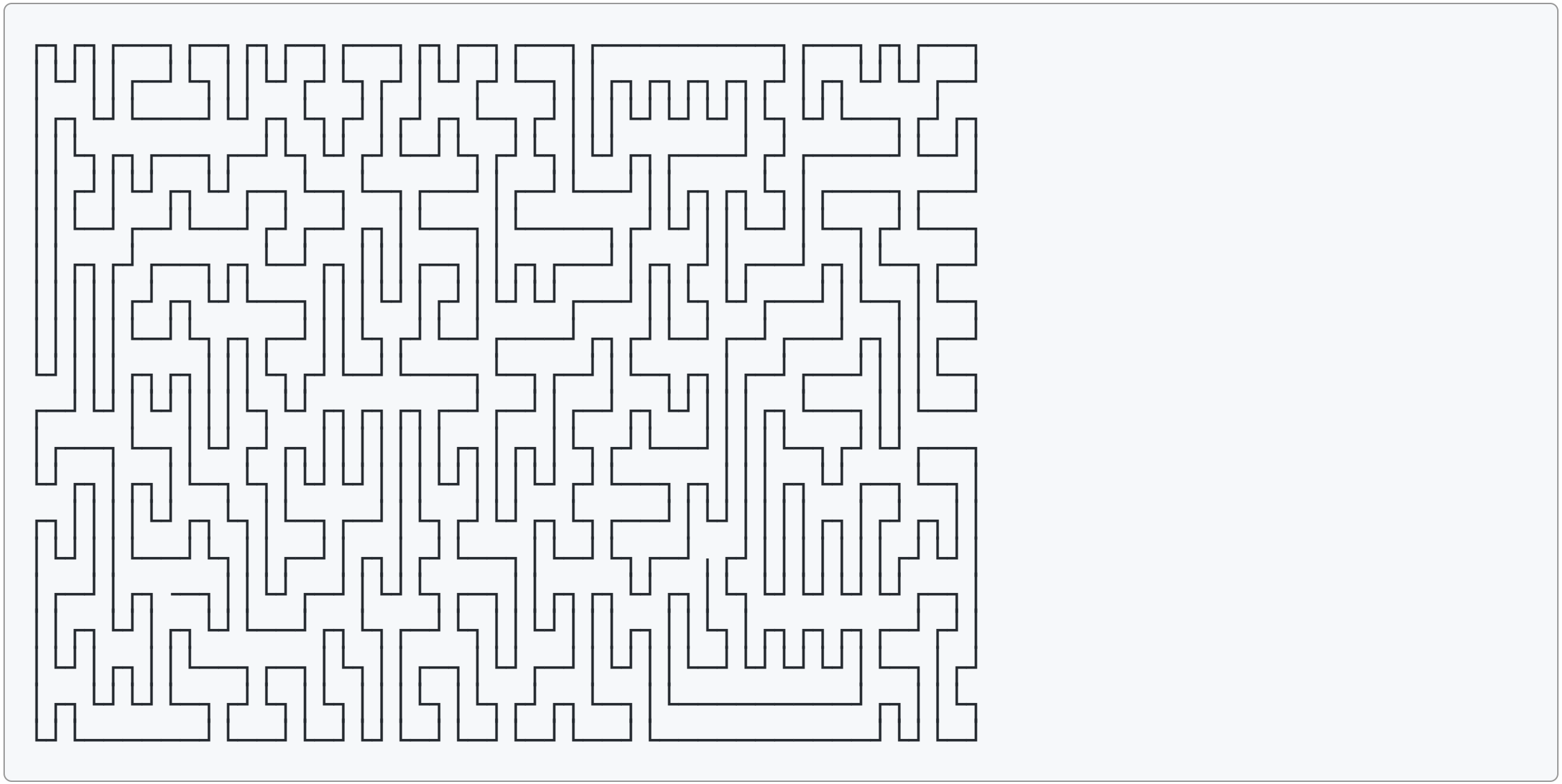
```
def disconnect(startpoint, endpoint, direction):
    p = startpoint
    m = move[direction]
    p = (p[0]+m[0], p[1]+m[1])
    while p != endpoint:
        direction = array[p[0]][p[1]] ^ opposite[direction]
        m = move[direction]
        p = (p[0]+m[0], p[1]+m[1])
    array[p[0]][p[1]] ^= opposite[direction]
    array[p[0]-m[0]][p[1]-m[1]] ^= direction
    return (p[0]-m[0], p[1]-m[1])
```

```
def disconnect(startpoint, endpoint, direction):
    p = startpoint
    m = move[direction]
    p = (p[0]+m[0], p[1]+m[1])
    ...
```

```
def disconnect(startpoint, endpoint, direction):
    p = startpoint
    m = move[direction]
    p = (p[0]+m[0], p[1]+m[1])
    while p != endpoint:
        direction = array[p[0]][p[1]] ^ opposite[direction]
        m = move[direction]
        p = (p[0]+m[0], p[1]+m[1])
```

```
def disconnect(startpoint, endpoint, direction):
    p = startpoint
    m = move[direction]
    p = (p[0]+m[0], p[1]+m[1])
    while p != endpoint:
        direction = array[p[0]][p[1]] ^ opposite[direction]
        m = move[direction]
        p = (p[0]+m[0], p[1]+m[1])
    array[p[0]][p[1]] ^= opposite[direction]
    array[p[0]-m[0]][p[1]-m[1]] ^= direction
    return (p[0]-m[0], p[1]-m[1])
```

```
total = number_of_rows*number_of_columns*100
for i in range(total):
    modify()
```



https://replit.com/@lemire/UnselfishWonderfulBackups#main.py

Homework

Use Python to generate a pretty output