Recursive Backtracking, Round 2

(And recursive backtracking with no “backtracking”)
Remember this?
Recursive Backtracking

or

or

or...

or

or...

or

or...

or...
The pattern

You have some state.

Chessboard
Sudoku board
Boggle board
(and non-board things)
The pattern

You have some state.

If you’re in a winning state, hooray!

For each action you can take:
- Take that action.
- Recurse.
- See what happened.

Base Case
## mazeEscape

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>.</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Can you escape from \((r, c)\) in this maze in *exactly* \(n\) steps?

```java
public boolean mazeEscape(char[][] map, int row, int col, int steps) {
```
Can you escape from \((r, c)\) in this maze in exactly \(n\) steps?

\((1, 1, 3) \Rightarrow \text{true}\)
\((1, 1, 4) \Rightarrow \text{true}\)
\((1, 1, _) \Rightarrow \text{false}\)

```
public boolean mazeEscape(char[][] map, int row, int col, int steps) {
    Hint: your state is all four of these parameters!
```
A variation

1

0 1 2 3 4
0 X X . X X
1 X . . .
2 X X X X X

2

0 1 2 3 4
0 X X . X X
1 X X . .
2 X X X X X

3

0 1 2 3 4
0 X X . X X
1 X X X .
2 X X X X X

4

0 1 2 3 4
0 X X . X X
1 X X . .
2 X X X X X

Monday, November 5, 12
A variation

Original Recursive Backtracking:
- Check your base case
- For each move:
  - Copy the world
  - Modify the copy
  - Recurse
  - See what happened

New Recursive Backtracking:
- Check your base case
- For each move:
  - Modify the world
  - Recurse
  - See what happened
  - If you didn’t win, *undo the modification*!

*I always forget to do this part.*
What if you don’t backtrack?

0 1 2 3 4
0 X . . X X
1 . . . X .
2 . X X X .
3 . X . . .
4 . X X X X
5 . . . . .

Remind you of an APT? This should remind you of an APT...
What if you don’t backtrack?

```
0  1  2  3  4
0  X  .  .  X  X
1  .  .  .  X  .
2  .  X  X  X  .
3  .  X  .  .  .
4  .  X  X  X  X
5  .  .  .  .  .
```

Replace every ‘.’ on the board with a number. Every 1 should only be adjacent to other 1s, each 2 only adjacent to other 2s, and so on.
What if you don’t backtrack?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>X</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Replace every ‘.’ on the board with a number. Every 1 should only be adjacent to other 1s, each 2 only adjacent to other 2s, and so on.

```java
public void reachability(char[][][] map) {
```

Remind you of an APT? This should remind you of an APT...
Multi-Heap Nim

public void multiHeapNim(ArrayList<Integer> piles) {

Remind you of an APT? This should remind you of an APT...
public void multiHeapNim(ArrayList<Integer> piles) {

Remind you of an APT? This should remind you of an APT...
Democracy is a binary tree!

With nine states undecided:

- Obama has 431 ways to win (84.2% of paths)
- Romney has 76 ways to win (14.8% of paths)

If Obama wins Florida...

- Florida
- Ohio
- North Carolina
- Virginia
- Wisconsin
- Colorado
- Iowa
- Nevada
- New Hampshire

If Romney wins Florida...

- Florida