INFORMED SEARCH
Three foxes and three chickens wish to cross the river. They have a small boat that will carry up to two animals. The boat can’t cross unless it has at least one animal to drive it. If at any time the Foxes outnumber the Chickens on either bank of the river, they will eat the Chickens. Find the smallest number of crossings that will allow everyone to cross the river safely.

What is the “state” of this problem (it should capture all possible valid configurations)?
Three foxes and three chickens wish to cross the river. They have a small boat that will carry up to two animals. The boat can’t cross unless it has at least one animal to drive it. If at any time the Foxes outnumber the Chickens on either bank of the river, they will eat the Chickens. Find the smallest number of crossings that will allow everyone to cross the river safely.
Foxes and Chickens

Three foxes and three chickens wish to cross the river. They have a small boat that will carry up to two animals. The boat can’t cross unless it has at least one animal to drive it. If at any time the Foxes outnumber the Chickens on either bank of the river, they will eat the Chickens. Find the smallest number of crossings that will allow everyone to cross the river safely.

- CCCFFFF B
- CCFF
- CF
- ...
Searching for a solution

What states can we get to from here?
Searching for a solution

Next states?
### Foxes and Chickens Solution

<table>
<thead>
<tr>
<th>Step</th>
<th>Near side</th>
<th>Far side</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CCCFFFF</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>CCCF</td>
<td>B FF</td>
</tr>
<tr>
<td>2</td>
<td>CCCFFF</td>
<td>B F</td>
</tr>
<tr>
<td>3</td>
<td>CCC</td>
<td>B FFF</td>
</tr>
<tr>
<td>4</td>
<td>CCCF</td>
<td>B FF</td>
</tr>
<tr>
<td>5</td>
<td>CF</td>
<td>B CCFF</td>
</tr>
<tr>
<td>6</td>
<td>CCFFF</td>
<td>B CF</td>
</tr>
<tr>
<td>7</td>
<td>FF</td>
<td>B CCCF</td>
</tr>
<tr>
<td>8</td>
<td>FFF</td>
<td>B CCC</td>
</tr>
<tr>
<td>9</td>
<td>F</td>
<td>B CCCFF</td>
</tr>
<tr>
<td>10</td>
<td>FF</td>
<td>B CCCF</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>B CCCFFF</td>
</tr>
</tbody>
</table>

How is this solution different than the n-queens problem?
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Near side</th>
<th>Far side</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial setup:</td>
<td>CCCFFFF</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>Two foxes cross over:</td>
<td>CCCF</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>One comes back:</td>
<td>CCCFF</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>Two foxes go over again:</td>
<td>CCC</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>One comes back:</td>
<td>CCCF</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>Two chickens cross:</td>
<td>CF</td>
<td>B</td>
</tr>
<tr>
<td>6</td>
<td>A fox &amp; chicken return:</td>
<td>CCFF</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>Two chickens cross again:</td>
<td>FF</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>A fox returns:</td>
<td>FFF</td>
<td>B</td>
</tr>
<tr>
<td>9</td>
<td>Two foxes cross:</td>
<td>F</td>
<td>B</td>
</tr>
<tr>
<td>10</td>
<td>One returns:</td>
<td>FF</td>
<td>B</td>
</tr>
<tr>
<td>11</td>
<td>And brings over the third:</td>
<td>-</td>
<td>B</td>
</tr>
</tbody>
</table>

Solution is not a state, but a sequence of actions (or a sequence of states)!
One other problem

What would happen if we ran DFS here?
One other problem

If we always go left first, will continue forever!
One other problem

Does BFS have this problem? No!
DFS vs. BFS

Why do we use DFS then, and not BFS?
Consider a search problem where each state has two states you can reach.

Assume the goal state involves 20 actions, i.e. moving between ~20 states.

How big can the queue get for BFS?
Consider a search problem where each state has two states you can reach.

Assume the goal state involves 20 actions, i.e. moving between ~20 states.

At any point, need to remember roughly a “row”
Consider a search problem where each state has two states you can reach.

Assume the goal state involves 20 actions, i.e. moving between ~20 states.

How big does this get?
Consider a search problem where each state has two states you can reach.

Assume the goal state involves 20 actions, i.e. moving between ~20 states.

Doubles every level we have to go deeper. For 20 actions that is $2^{20} = \sim 1$ million states!
Consider a search problem where each state has two states you can reach.

Assume the goal state involves 20 actions, i.e. moving between ~20 states.

How many states would DFS keep on the stack?
DFS vs. BFS

Consider a search problem where each state has two states you can reach.

Assume the goal state involves 20 actions, i.e. moving between ~20 states.

Only one path through the tree, roughly 20 states.
One other problem

If we always go left first, will continue forever!

Solution?
DFS avoiding repeats

```python
def dfs(state, visited):
    # note that we've visited this state
    visited[str(state)] = True

    if state.is_goal():
        return [state]
    else:
        result = []

        for s in state.next_states():
            # check if we've visited a state already
            if not str(s) in visited:
                result += dfs(s, visited)

        return result
```
Other search problems

What problems have you seen that could be posed as search problems?

What is the state?

Start state

Goal state

State-space/transition between states
8-puzzle

Start State

Goal State
8-puzzle

goal

state representation?

start state?

state-space/transitions?
8-puzzle

**state:**
- all 3 x 3 configurations of the tiles on the board

**transitions between states:**
- Move Blank Square Left, Right, Up or Down.
- This is a more efficient encoding than moving each of the 8 distinct tiles
Cryptarithmetic

Find an assignment of digits (0, ..., 9) to letters so that a given arithmetic expression is true. examples:

\[
\begin{array}{c}
\text{SEND} + \text{MORE} = \text{MONEY}\\
\text{FORTY} + \text{TEN} + \text{TEN} + \text{TEN} = \text{SIXTY}\\
\text{F}=2, \text{ O}=9, \text{ R}=7, \text{ etc.}
\end{array}
\]
Given the following configuration of sticks, remove exactly 5 sticks in such a way that the remaining configuration forms exactly 3 squares.
Given a full 5-gallon jug and a full 2-gallon jug, fill the 2-gallon jug with exactly one gallon of water.
Water Jug Problem

State = (x, y), where x is the number of gallons of water in the 5-gallon jug and y is # of gallons in the 2-gallon jug

Initial State = (5, 2)

Goal State = (*, 1), where * means any amount

Operator table

<table>
<thead>
<tr>
<th>Name</th>
<th>Cond.</th>
<th>Transition</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty5</td>
<td>–</td>
<td>(x, y) → (0, y)</td>
<td>Empty 5-gal. jug</td>
</tr>
<tr>
<td>Empty2</td>
<td>–</td>
<td>(x, y) → (x, 0)</td>
<td>Empty 2-gal. jug</td>
</tr>
<tr>
<td>2to5</td>
<td>x ≤ 3</td>
<td>(x, 2) → (x+2, 0)</td>
<td>Pour 2-gal. into 5-gal.</td>
</tr>
<tr>
<td>5to2</td>
<td>x ≥ 2</td>
<td>(x, 0) → (x-2, 2)</td>
<td>Pour 5-gal. into 2-gal.</td>
</tr>
<tr>
<td>5to2part</td>
<td>y &lt; 2</td>
<td>(1, y) → (0, y+1)</td>
<td>Pour partial 5-gal. into 2-gal.</td>
</tr>
</tbody>
</table>
8-puzzle revisited

How hard is this problem?

```
1 3 8
4 7
6 5 2
```
8-puzzle revisited

The average depth of a solution for an 8-puzzle is 22 moves.

An exhaustive search requires searching \( \sim 3^{22} = 3.1 \times 10^{10} \) states:
- BFS: 10 terabytes of memory
- DFS: 8 hours (assuming one million nodes/second)

Can we do better?

Is DFS and BFS intelligent?
from: Claremont to: Rowland Heights

What would the search algorithms do?
from: Claremont to: Rowland Heights

DFS
from: Claremont to: Rowland Heights

BFS
from: Claremont to: Rowland Heights

Ideas?
from: Claremont to: Rowland Heights

We’d like to bias search towards the actual solution
Informed search

Order to_visit based on some knowledge of the world that estimates how “good” a state is

- $h(n)$ is called an evaluation function

Best-first search

- rank to_visit based on $h(n)$
- take the most desirable state in to_visit first
- different approaches depending on how we define $h(n)$
Heuristic

Merriam-Webster's Online Dictionary
Heuristic (pron. \hyu-’ris-tik\): adj. [from Greek heuriskoín to discover.] involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods

The Free On-line Dictionary of Computing (2/19/13)
heuristic 1. Of or relating to a usually speculative formulation serving as a guide in the investigation or solution of a problem: "The historian discovers the past by the judicious use of such a heuristic device as the 'ideal type'" (Karl J. Weintraub).
Heuristic function: $h(n)$

An estimate of how close the node is to a goal

Uses domain-specific knowledge!

Examples

- Map path finding?
- 8-puzzle?
- Foxes and chickens?
Heuristic function: $h(n)$

An estimate of how close the node is to a goal

Uses domain-specific knowledge!

Examples

- Map path finding?
  - straight-line distance from the node to the goal (“as the crow flies”)
- 8-puzzle?
  - how many tiles are out of place
  - sum of the “distances” of the out of place tiles
- Foxes and chickens?
  - number of animals on the starting bank
### Two heuristics

#### Which state is better?

<table>
<thead>
<tr>
<th>2</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

GOAL
Two heuristics

How many tiles are out of place?

How many tiles are out of place?
Two heuristics

Goal

5
Two heuristics

What is the “distance” of the tiles that are out of place?

Goal
Two heuristics

Goal

6
Two heuristics

Tiles out of place

Sum of distances for out of place tiles

5

6

GOAL
Two heuristics

Tiles out of place | Sum of distances for out of place tiles

<table>
<thead>
<tr>
<th>2</th>
<th>8</th>
<th>3</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GOAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>
Two heuristics

Tiles out of place

Sum of distances for out of place tiles

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Which heuristic is better (if either)?

GOAL

1 2 3
8 4
7 6 5
Two heuristics

Tiles out of place

Sum of distances for out of place tiles

More closely approximates “real” number of steps remaining?
<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Next states?
Which would you do?
Which would DFS choose

Completely depends on how next states are generated. Not an “intelligent” decision!
Best first search: out of place tiles?
Best first search: distance of tiles?
Next states?
Which next for best first search?
GOAL
Informed search algorithms

Best first search is called an “informed” search algorithm

Why wouldn’t we always use an informed algorithm?

• Coming up with good heuristics can be hard for some problems
• There is computational overhead (both in calculating the heuristic and in keeping track of the next “best” state)
Informed search algorithms

Any other problems/concerns about best first search?
Informed search algorithms

Any other problems/concerns about best first search?

- Only as good as the heuristic function

Best first search using distance as the crow flies as heuristic

What would the search do?
Informed search algorithms

Any other problems/concerns about best first search?

- Only as good as the heuristic function

Best first search using distance as the crow flies as heuristic

What is the problem?
Informed search algorithms

Any other problems/concerns about best first search?

• Only as good as the heuristic function

Best first search using distance as the crow flies as heuristic

Doesn’t take into account how far it has come.
Best first search is a “greedy” algorithm
Informed search algorithms

Best first search is called an “informed” search algorithm

There are many other informed search algorithms:

- A* search (and variants)
- Theta*
- Beam search
Sudoku

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)
# Sudoku

Fill in the grid with the numbers 1-9

- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Sudoku

Fill in the grid with the numbers 1-9

- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

How can we pose this as a search problem?

- **State**
- **Start state**
- **Goal state**
- **State space/transitions**
Fill in the grid with the numbers 1-9

- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)
Fill in the grid with the numbers 1-9

- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sudoku
Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

How many next states? What are they?
Fill in the grid with the numbers 1-9
  • each row has 1-9 (without repetition)
  • each column has 1-9 (without repetition)
  • each quadrant has 1-9 (without repetition)

Generate next states:
  • pick an open entry
  • try all possible numbers that meet constraints

1, 6, 7, 9
Sudoku

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

1, 6, 7, 9
Sudoku

```
1    | 4    | 3    |         | 6    | 7    |
---   | ---   | ---   |         | ---   | ---   |
5    |       |       |         | 4    | 2    |
---   | ---   | ---   |         | ---   | ---   |
8    |       |       |         | 6    | 1    |
---   | ---   | ---   |         | ---   | ---   |
2    |       |       |         |       | 5    |
---   | ---   | ---   |         | ---   | ---   |
5    |       |       |         |       | 4    |
---   | ---   | ---   |         | ---   | ---   |
6    |       |       |         | 7    |     |
---   | ---   | ---   |         | ---   | ---   |
5    |       | 1     |         | 8    |     |
---   | ---   | ---   |         | ---   | ---   |
8    |       |       |         |     |     |
---   | ---   | ---   |
```

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

How many next states? What are they?

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)
Fill in the grid with the numbers 1-9

- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

2, 6, 7, 8, 9
## Sudoku

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th></th>
<th>6</th>
<th>7</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td></td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>6</td>
<td></td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2, 6, 7, 8, 9
Sudoku

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

What are the next states?
Sudoku

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

7, 8, 9
Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

7, 8, 9
Sudoku

Fill in the grid with the numbers 1-9

- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>6</td>
<td></td>
<td>1</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints
# Sudoku

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td>8</td>
<td>6</td>
<td>1</td>
<td></td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1</td>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints
Fill in the grid with the numbers 1-9

- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

Now what?

Try another branch, i.e. go back to a place where we had a decision and try a different one
Fill in the grid with the numbers 1-9

- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

7, 8, 9
Best first Sudoku search

DFS and BFS will choose entries (and numbers within those entries) randomly

Is that how people do it?

How do you do it?

Heuristics for best first search?

Generate next states:
• pick an open entry
• try all possible numbers that meet constraints
Best first Sudoku search

DFS and BFS will choose entries (and numbers within those entries) randomly

Pick the entry that is **MOST** constrained

People often try and find entries where only one option exists and only fill it in that way (very little search)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints
Board is a matrix (list of lists)

Each entry is either:

- a number (if we’ve filled in the space already, either during search or as part of the starting state)
- a list of numbers that are valid to put in that entry if it hasn’t been filled in yet

Which is the most constrained (of the ones above)?
Representing the Sudoku board

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1</td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

Which is the most constrained (of the ones above)?

- **Board is a matrix (list of lists)**
- **Each entry is either:**
  - a number (if we’ve filled in the space already, either during search or as part of the starting state)
  - a list of numbers that are valid to put in that entry if it hasn’t been filled in yet
Representing the Sudoku board

Board is a matrix (list of lists)

Each entry is either:
- a number (if we’ve filled in the space already, either during search or as part of the starting state)
- a list of numbers that are valid to put in that entry if it hasn’t been filled in yet

What would the state look like if we add pick 1?
Representing the Sudoku board

Board is a matrix (list of lists)

Each entry is either:

- a number (if we’ve filled in the space already, either during search or as part of the starting state)
- a list of numbers that are valid to put in that entry if it hasn’t been filled in yet

Remove 1 from all entries in the quadrant

What other parts of the board need to be updated?
Board is a matrix (list of lists)

Each entry is *either*:

- a number (if we’ve filled in the space already, either during search or as part of the starting state)
- a list of numbers that are valid to put in that entry if it hasn’t been filled in yet

Remove 1 from all entries in the quadrant

Remove 1 from all entries in the same column

Remove 1 from all entries in the same row

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td>4</td>
<td>2</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[6, 7, 9], [2, 6, 7, 8, 9], [2, 7, 8, 9], [9], 4, 3, 5, [6, 7, 9], [7, 9]