Search algorithm

Keep track of a list of states that we could visit, we’ll call it “to_visit”

General idea:
- take a state off the to_visit list
- if it’s the goal state
  - we’re done!
- if it’s not the goal state
  - Add all of the next states to the to_visit list
- repeat

Search algorithms

add the start state to to_visit

Repeat
- take a state off the to_visit list
- if it’s the goal state
  - we’re done!
- if it’s not the goal state
  - Add all of the next states to the to_visit list

Two variants: breadth first search (BFS) and depth first search (DFS) depending on whether we use a stack or a queue for to_visit. Which is which?
Search algorithms

add the start state to `to_visit`

Repeat
- take a state off the `to_visit` list
- if it's the goal state
  - we're done!
- if it's not the goal state
  - Add all of the next states to the `to_visit` list

Depth first search (DFS): `to_visit` is a stack
Breadth first search (BFS): `to_visit` is a queue

Implementing the state space

What the “world” (in this case a maze) looks like
- We’ll define the world as a collection of discrete states
- States are connected if we can get from one state to another by taking a particular action
- This is called the “state space”

State:
- Is this the goal state? (is_goal)
- What states are connected to this state? (next_states)

Implementing state space

Search variants implemented

add the start state to `to_visit`

Repeat
- take a state off the `to_visit` list
- if it's the goal state
  - we're done!
- if it's not the goal state
  - Add all of the next states to the `to_visit` list

Definition:
- `dfs(start_state):`
  - `s = Stack()`
  - return search(start_state, s)
- `bfs(start_state):`
  - `q = Queue()`
  - return search(start_state, q)
- `search(start_state, to_visit):`
  - `to_visit.push(start_state)`
  - while `not to_visit.is_empty()`:
    - `current = to_visit.remove()`
    - if `current.is_goal()`:
      - return current
    - else:
      - for `s in current.next_states()`:
        - `to_visit.add(s)`
  - return None
What order would this variant visit the states?

```
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            if result != None:
                return result
        return None
```

1, 2, 5

DFS!

What search algorithm is this?

```
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            result = search(s)
            if result != None:
                return result
        return None
```

1, 2, 5, 3, 6, 9, 7, 8

DFS with a stack

Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue
DFS with a stack

Depth first search (DFS): \( \text{to\_visit} \) is a stack
Breadth first search (BFS): \( \text{to\_visit} \) is a queue

One last DFS variant

How is this different?

Matrices!
N-queens problem
Place N queens on an N by N chess board such that none of the N queens are attacking any other queen.

Solution(s)?

N-queens problem
Place N queens on an N by N chess board such that none of the N queens are attacking any other queen.

Solution(s)?

N-queens problem
Place N queens on an N by N chess board such that none of the N queens are attacking any other queen.

Solution(s)?

N-queens problem
Place N queens on an N by N chess board such that none of the N queens are attacking any other queen.

How do we solve this with search:

What is a state?
What is the start state?
What is the goal?
How do we transition from one state to the next?
Search algorithm

add the start state to `to_visit`

Repeat
- take a state off the `to_visit` list
- if it’s the goal state
  - we’re done!
- if it’s not the goal state
  - What states can I get to from the current state?
  - Add all of the next states to the `to_visit` list

Any problem that we can define these three things can be plugged into the search algorithm!

N queens problem