What order will BFS and DFS visit the states assuming states are added to to_visit left to right?

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

What order will BFS and DFS visit the states?

DFS: 1, 4, 3, 8, 7, 6, 9, 2, 5
Why not 1, 2, 5?

Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue
What order will BFS and DFS visit the states?

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

DFS: 1, 4, 3, 8, 7, 6, 9, 2, 5

BFS: 1, 2, 3, 4, 5

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 successive states to the to_visit list
- if to_visit is empty():
  - return current
else:
  - for s in current.next_states():
    - to_visit.add(s)
return None
What order would this variant visit the states?

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

1, 2, 5

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

What search algorithm is this?

DFS!

Where's the stack?

One last DFS variant

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

DFS!

Where's the stack?

How is this different?
One last DFS variant

```python
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            result = search(s)
            if result is not None:
                return result
    return None

def dfs(state):
    if state.is_goal():
        return (state)
    else:
        result = []
        for s in state.next_states():
            result += dfs(s)
        return result

Returns ALL solutions found, not just one
```

Missionaries and Cannibals

Three missionaries and three cannibals wish to cross the river. They have a small boat that will carry up to two people. Everyone can navigate the boat. If at any time the Cannibals outnumber the Missionaries on either bank of the river, they will eat the Missionaries. 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)?

Missionaries and Cannibals

Three missionaries and three cannibals wish to cross the river. They have a small boat that will carry up to two people. Everyone can navigate the boat. If at any time the Cannibals outnumber the Missionaries on either bank of the river, they will eat the Missionaries. Find the smallest number of crossings that will allow everyone to cross the river safely.

<table>
<thead>
<tr>
<th>Missionary1</th>
<th>Missionary2</th>
<th>Missionary3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cannibal1</td>
<td>Cannibal2</td>
<td>Cannibal3</td>
</tr>
</tbody>
</table>

MMMCCC B

MMCC B MC

MC B MMCC

...
Searching for a solution

MMMCCC B ~~

What states can we get to from here?

Searching for a solution

MMMCCC B ~~

MMMCC ~ B C

MMCC ~ B MC

MMC ~ B CC

Next states?

Code!

http://www.cs.pomona.edu/~dkauchak/classes/cs30/examples/cannibals.txt

Talk about copy.deepcopy
Missionaries and Cannibals Solution

<table>
<thead>
<tr>
<th>Near side</th>
<th>Far side</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Initial setup:</td>
<td>MMMCCC B -</td>
</tr>
<tr>
<td>1 Two cannibals cross over:</td>
<td>MMMC B CC</td>
</tr>
<tr>
<td>2 One comes back:</td>
<td>MMMCC B C</td>
</tr>
<tr>
<td>3 Two cannibals go over again:</td>
<td>MMM C CCC</td>
</tr>
<tr>
<td>4 One comes back:</td>
<td>MMMB B CC</td>
</tr>
<tr>
<td>5 Two missionaries cross:</td>
<td>MNC B MMCC</td>
</tr>
<tr>
<td>6 A missionary &amp; cannibal return:</td>
<td>MMCC B MC</td>
</tr>
<tr>
<td>7 Two missionaries cross again:</td>
<td>CCCC B MMM</td>
</tr>
<tr>
<td>8 A cannibal returns:</td>
<td>CCCC B MMM</td>
</tr>
<tr>
<td>9 Two cannibals cross:</td>
<td>C B MMMCC</td>
</tr>
<tr>
<td>10 One returns:</td>
<td>CC B MMMC</td>
</tr>
<tr>
<td>11 And brings over the third:</td>
<td>- B MMMCCC</td>
</tr>
</tbody>
</table>

How is this solution different than the n-queens problem?

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?

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?

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?

Doubles every level we have to go deeper. For 20 actions that is $2^{20} = ~1$ million states!

How many states would DFS keep on the stack?

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
```