SEARCH ALGORITHMS

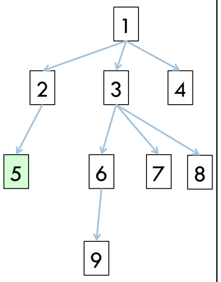
David Kauchak
CS30 – Spring 2015

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

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

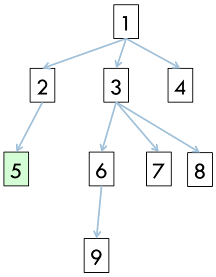


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

What order will BFS and DFS visit the states?

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

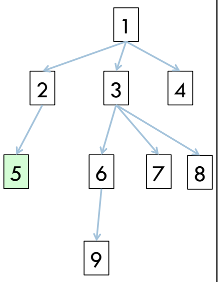
Why not 1, 2, 5?

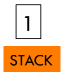


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

What order will BFS and DFS visit the states?

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

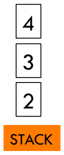


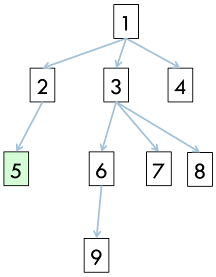


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

What order will BFS and DFS visit the states?

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

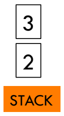


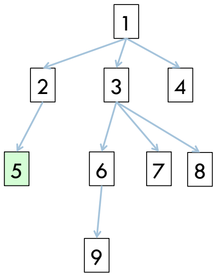


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

What order will BFS and DFS visit the states?

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



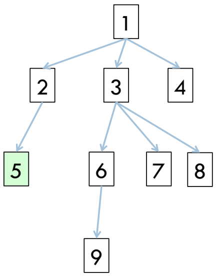


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

What order will BFS and DFS visit the states?

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

BFS: 1, 2, 3, 4, 5



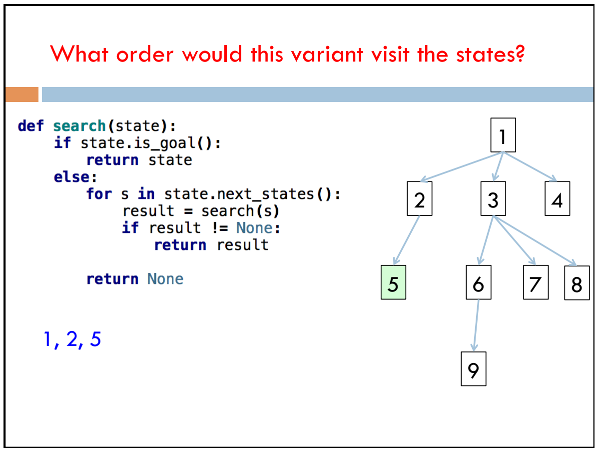
Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

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

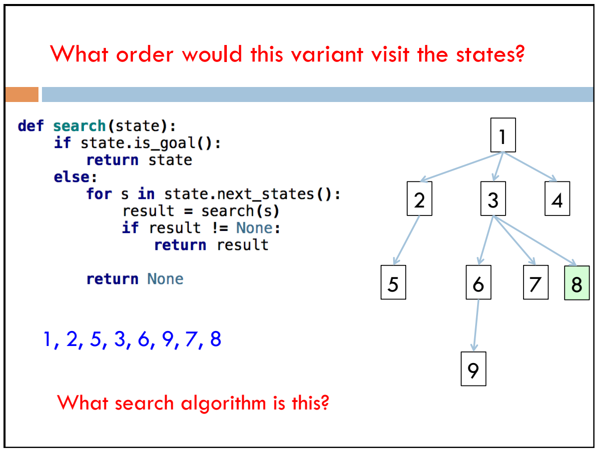
```
def dfs(start_state):
    s = Stack()
    return search(start_state, s)

def bfs(start_state):
    q = Queue()
    return search(start_state, q)

def search(start_state, to_visit):
    to_visit.add(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():
            result = search(s)
            if result != None:
                return result

        return None
```

1, 2, 5

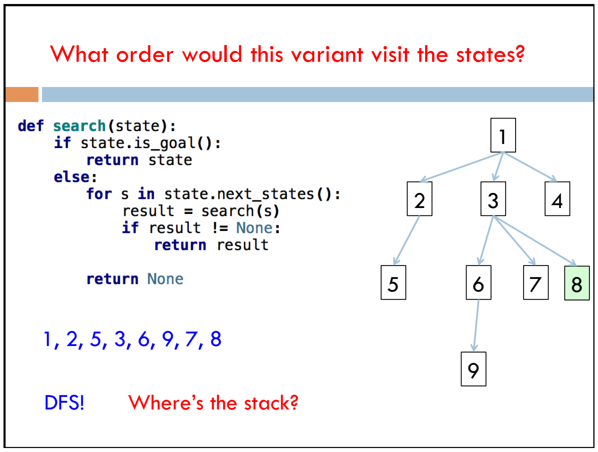


What order would this variant visit the states?

```
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

What search algorithm is this?

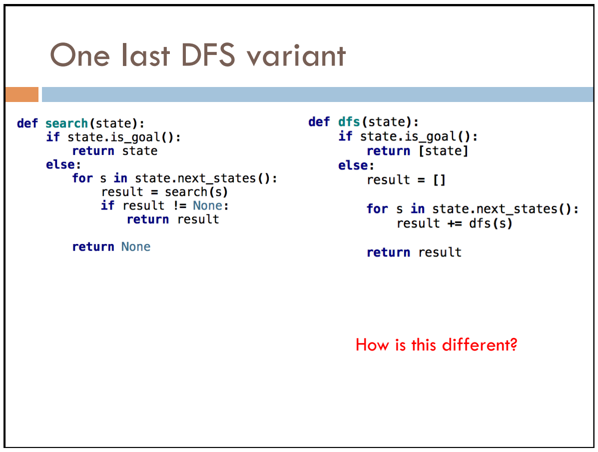


What order would this variant visit the states?

```
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! Where's the stack?



One last DFS variant

```
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
```

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

        for s in state.next_states():
            result += dfs(s)

        return result
```

How is this different?

One last DFS variant

```
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
```

```
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

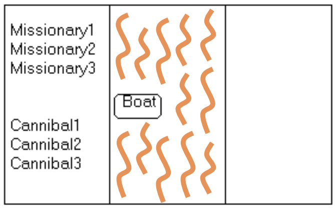
Missionaries and Cannibals

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.



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.

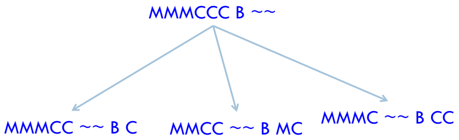
MMMCCC B	
MMCC	B MC
MC	B MMCC
…	

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

MMMC ~~ B CC

Next states?

Code!

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

Talk about copy.deepcopy

Missionaries and Cannibals Solution

```
                                     Near side      Far side
0 Initial setup:                     MMMCCC  B         -
1 Two cannibals cross over:          MMMC          B  CC
2 One comes back:                    MMMCC   B        C
3 Two cannibals go over again:       MMM           B  CCC
4 One comes back:                    MMMC    B        CC
5 Two missionaries cross:            MC            B  MMCC
6 A missionary & cannibal return:    MMCC    B        MC
7 Two missionaries cross again:      CC            B  MMMC
8 A cannibal returns:                CCC     B        MMM
9 Two cannibals cross:               C             B  MMMCC
10 One returns:                      CC      B        MMMC
11 And brings over the third:        -             B  MMMCCC
```

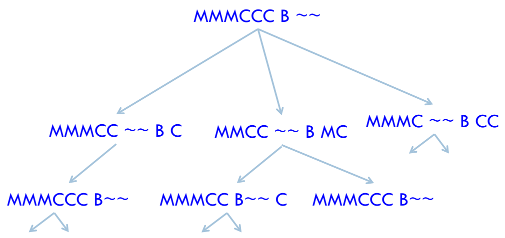
How is this solution different than the n-queens problem?

Missionaries and Cannibals Solution

```
                                     Near side      Far side
0 Initial setup:                     MMMCCC  B         -
1 Two cannibals cross over:          MMMC          B  CC
2 One comes back:                    MMMCC   B        C
3 Two cannibals go over again:       MMM           B  CCC
4 One comes back:                    MMMC    B        CC
5 Two missionaries cross:            MC            B  MMCC
6 A missionary & cannibal return:    MMCC    B        MC
7 Two missionaries cross again:      CC            B  MMMC
8 A cannibal returns:                CCC     B        MMM
9 Two cannibals cross:               C             B  MMMCC
10 One returns:                      CC      B        MMMC
11 And brings over the third:        -             B  MMMCCC
```

Solution is not a state, but a sequence of actions (or a sequence of states)

One other problem



MMMCCC B ~~

MMMCC ~~ B C

MMCC ~~ B MC

MMMC ~~ B CC

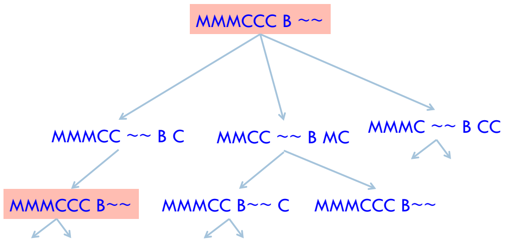
MMMCCC B~~

MMMCC B~~ C

MMMCCC B~~

What would happen if we ran DFS here?

One other problem



MMMCCC B ~~

MMMCC ~~ B C

MMCC ~~ B MC

MMMC ~~ B CC

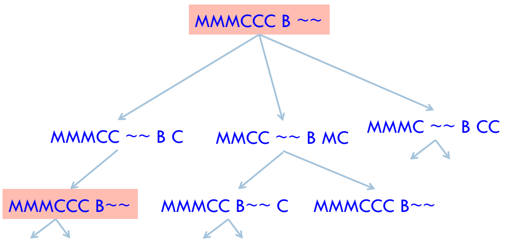
MMMCCC B~~

MMMCC B~~ C

MMMCCC B~~

If we always go left first, will continue forever!

One other problem



MMMCCC B ~~

MMMCC ~~ B C

MMCC ~~ B MC

MMMC ~~ B CC

MMMCCC B~~

MMMCC B~~ C

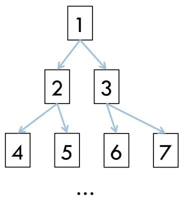
MMMCCC B~~

Does BFS have this problem? No!

DFS vs. BFS

Why do we use DFS then, and not BFS?

DFS vs. BFS

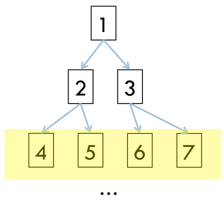


Consider a search problem where each state has two states you can reach

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

How big can the queue get for BFS?

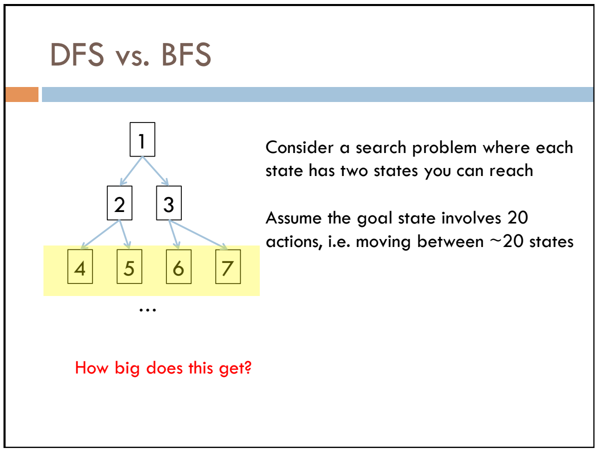
DFS vs. BFS



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

At any point, need to remember roughly a “row”

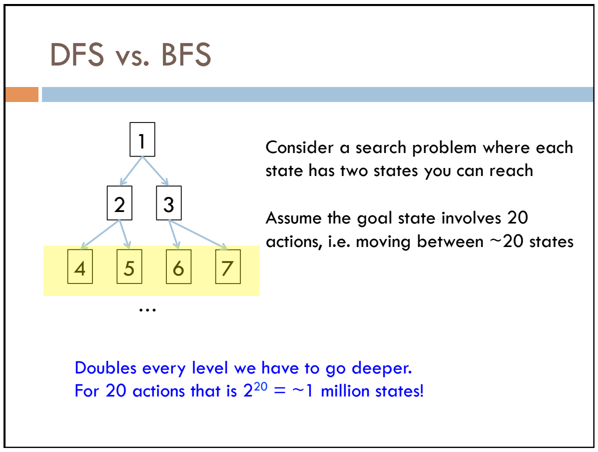


DFS vs. BFS

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

How big does this get?

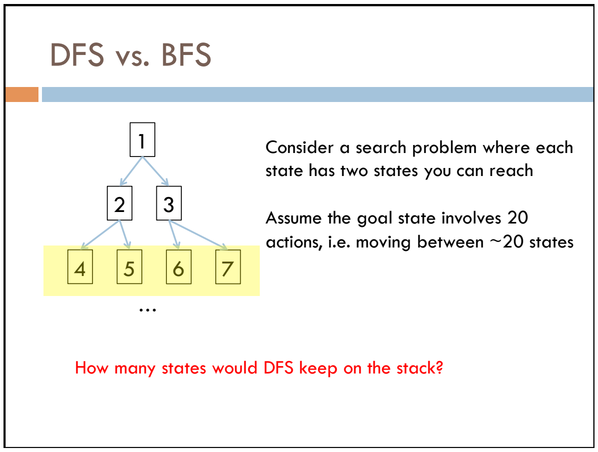


DFS vs. BFS

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

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

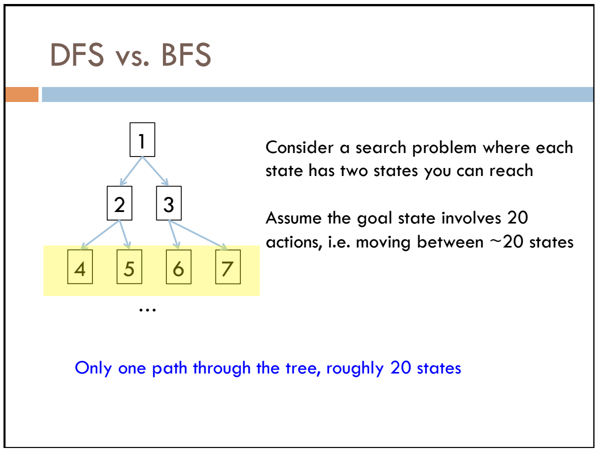


DFS vs. BFS

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

How many states would DFS keep on the stack?

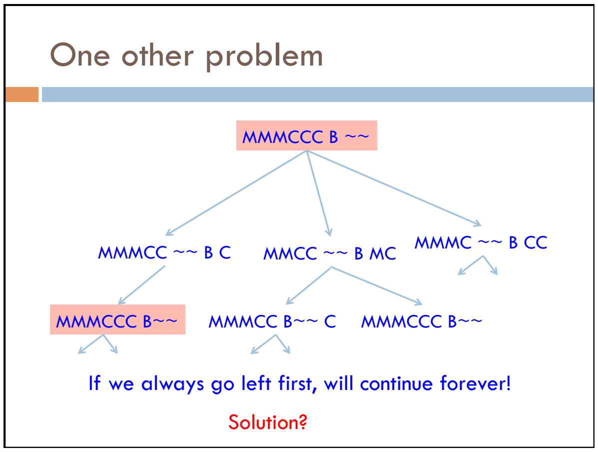


DFS vs. BFS

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

MMMCCC B ~~

MMMCC ~~ B C

MMCC ~~ B MC

MMMC ~~ B CC

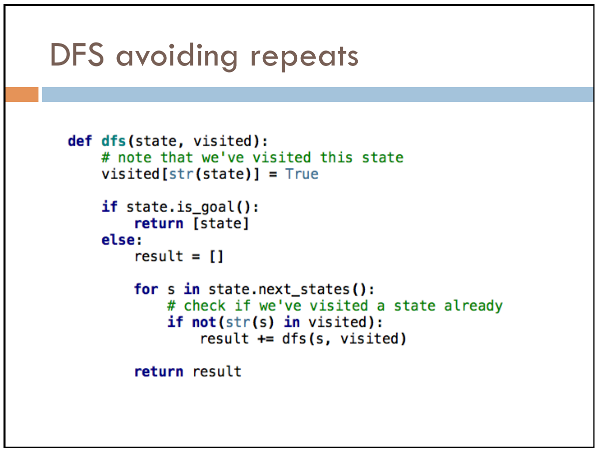
MMMCCC B~~

MMMCC B~~ C

MMMCCC B~~

If we always go left first, will continue forever!

Solution?



DFS avoiding repeats

```
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
```