SEARCH ALGORITHMS

David Kauchak
CS51A – Spring 2019

Admin

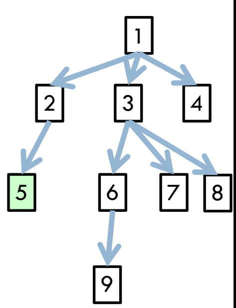
Assignment 9

No office hours today for Dr. Dave

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

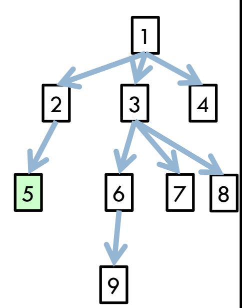


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

What order will BFS and DFS visit the states?

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

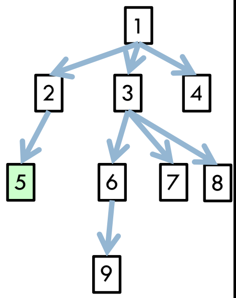
Why not 1, 2, 5?

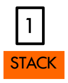


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

What order will BFS and DFS visit the states?

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

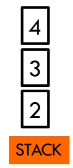


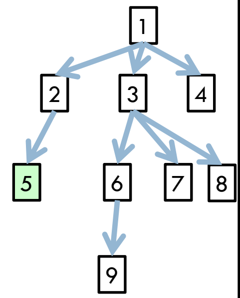


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

What order will BFS and DFS visit the states?

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

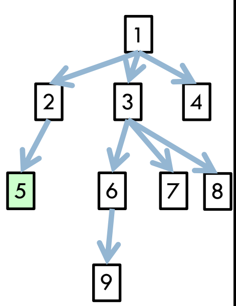


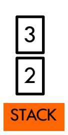


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

What order will BFS and DFS visit the states?

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



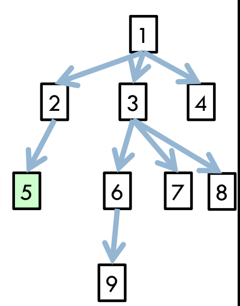


Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue

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

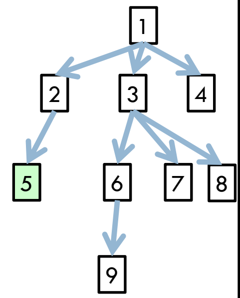
add the start state to to_visit

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?

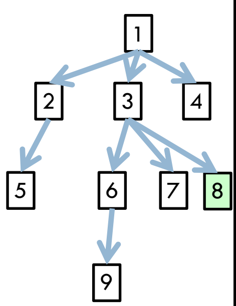
```
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            result = search(s)
            if result != None:
                return result

        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
```

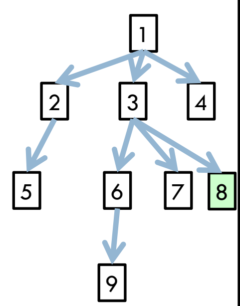


1, 2, 5, 3, 6, 9, 7, 8

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.