Search Algorithms

[Peter Mawhorter]

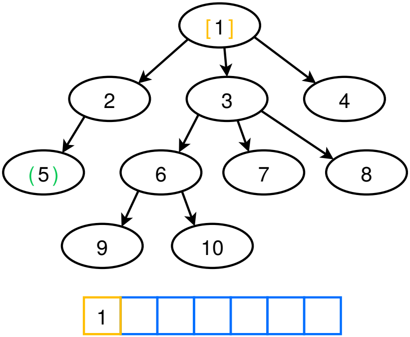
University of California Santa Cruz

March 31st, 2016

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're 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 not:
    - Add all successor states to to_visit.



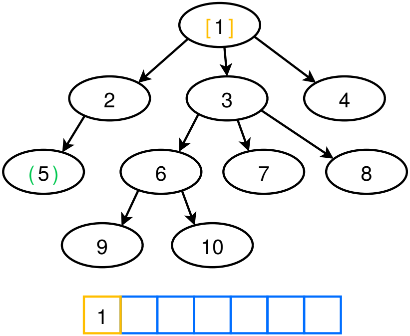
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1

BFS:



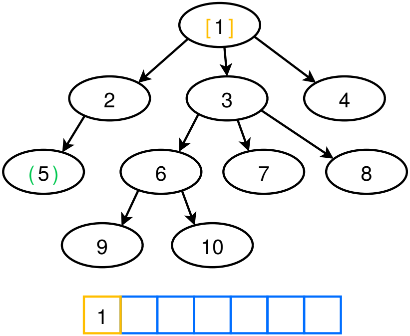
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 2, 5?

BFS:



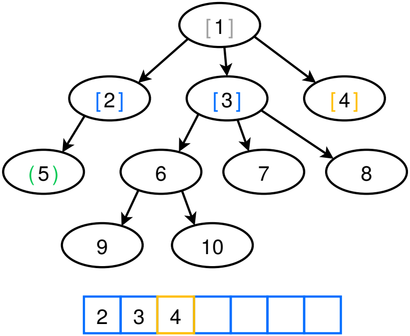
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4

BFS:



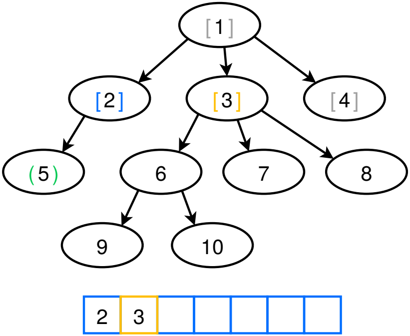
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3

BFS:



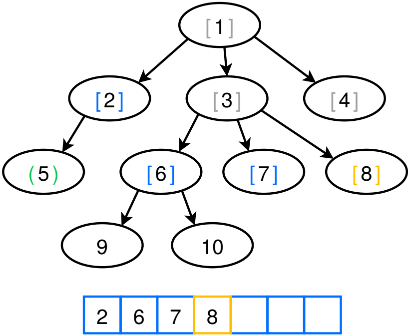
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8

BFS:



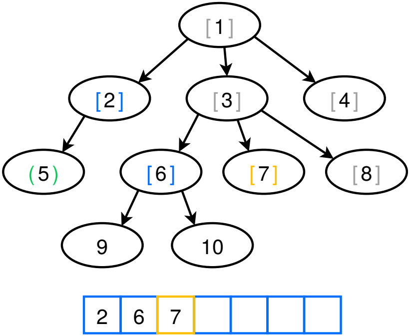
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7

BFS:



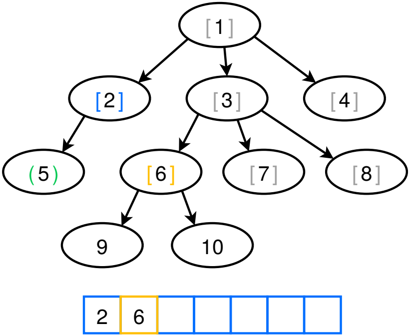
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

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

BFS:



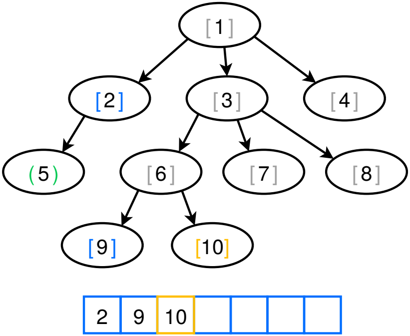
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7, 6, 10

BFS:



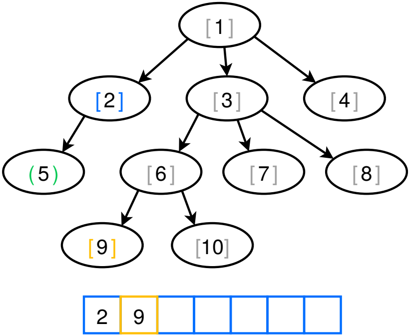
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7, 6, 10, 9

BFS:



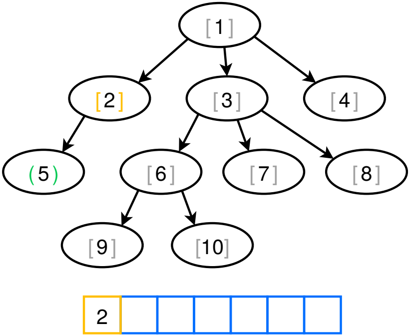
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7, 6, 10, 9, 2

BFS:



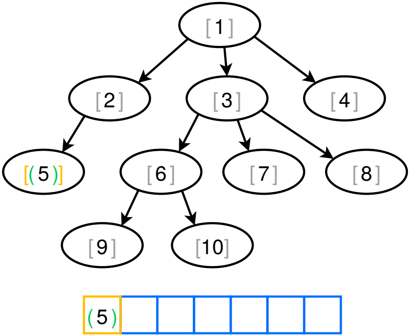
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7, 6, 10, 9, 2, 5

BFS:



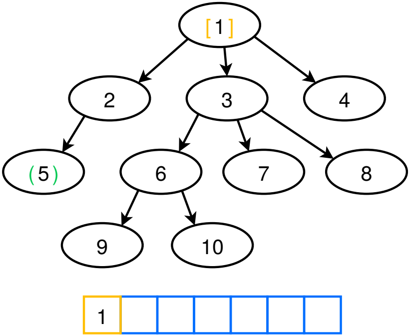
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7, 6, 10, 9, 2, 5

BFS:
1



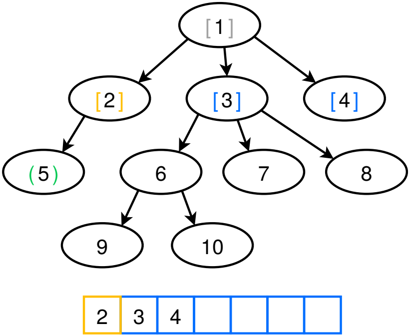
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7, 6, 10, 9, 2, 5

BFS:
1, 2



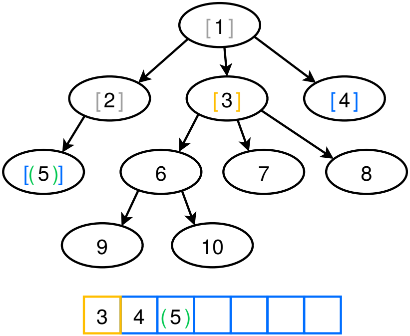
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7, 6, 10, 9, 2, 5

BFS:
1, 2, 3



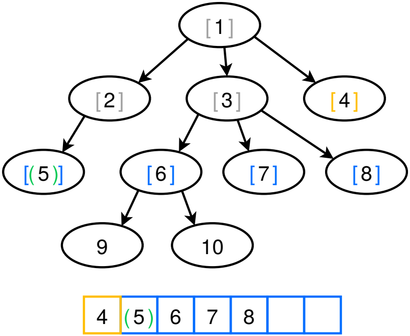
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7, 6, 10, 9, 2, 5

BFS:
1, 2, 3, 4



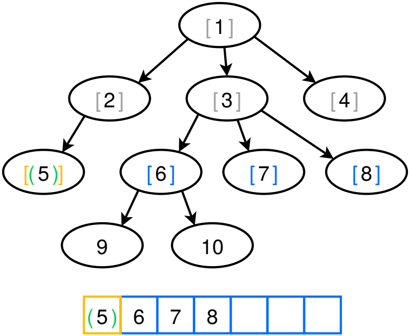
Depth-first search (DFS): to_visit is a stack
Breadth-first search (BFS): to_visit is a queue

Review: Ordering

In what order will BFS and DFS visit these states?
(assuming they're added to to_visit left-to-right)

DFS:
1, 4, 3, 8, 7, 6, 10, 9, 2, 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

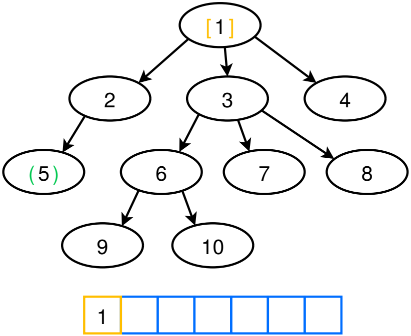
Search Implementations

- 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 not:
    - Add all successor states to to_visit.

```
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
```

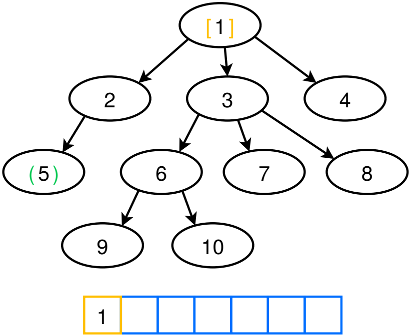
Search Implementations

```
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
```



Search Implementations

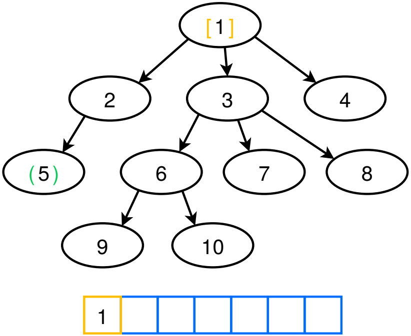
```
def search(state):
  if state.is_goal():
    return state
  else:
    for s in state.next_states():
      result = search(s)
      if result != None:
        return result

    return None
```



Ordering?

Search Implementations

```
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
```

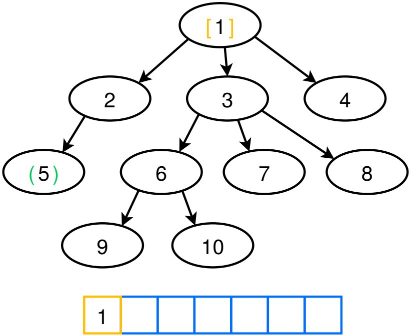


Ordering?

1, 2, 5

Search Implementations

```
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
```



Ordering?

1, 2, 5

What algorithm is this?

Search Implementations

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

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

    return result
```

What is the difference?

Search Implementations

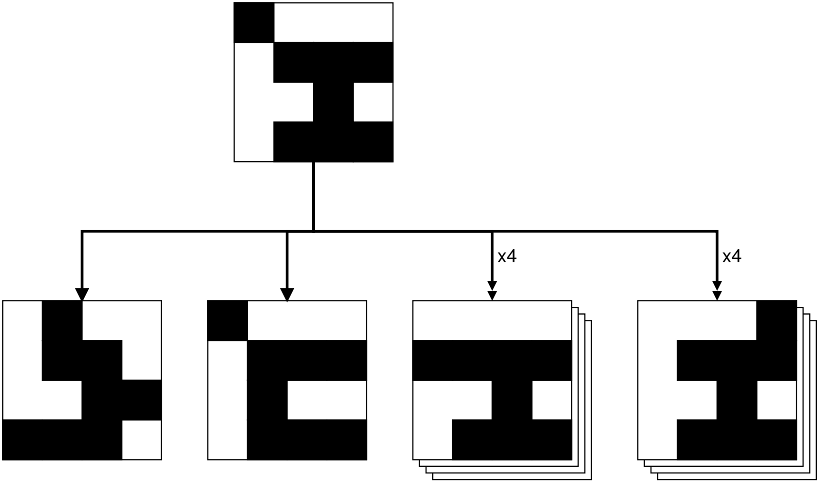
```python
def search(state):
  if state.is_goal():
    return state
  else:
    for s in state.next_states():
      result = search(s)
      if result != None:
        return result

    return None
```

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

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

    return result
```

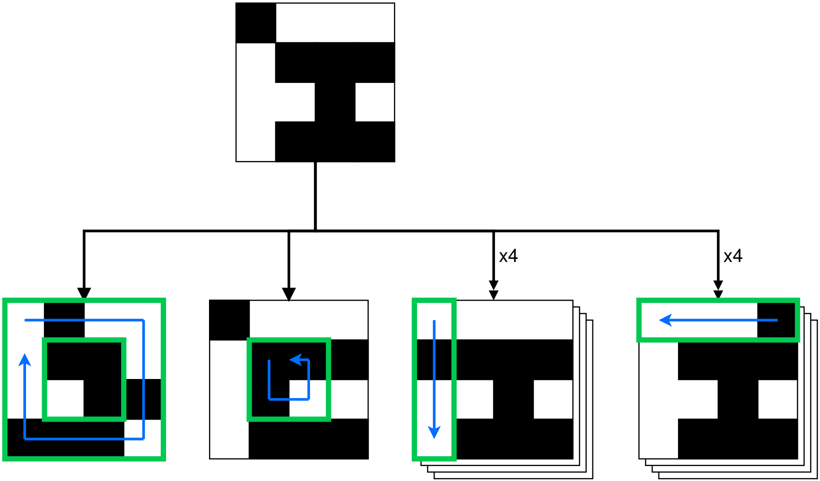
What is the difference?

Returns **all** solutions, not just one.

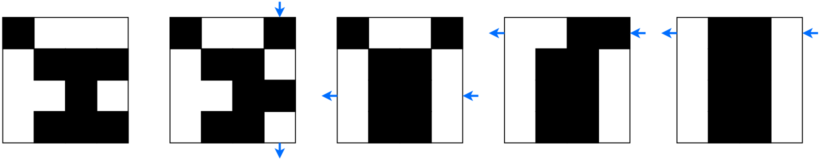
Square Puzzle



Square Puzzle



Square Puzzle



Square Puzzle

- How can we represent a state?
- How do we know if we're at a solution?
- How many next states does each state have?
- How can we get the next states?

Square Puzzle

- How can we represent a state?

Square Puzzle

- How can we represent a state?

```
( (0, 1, 1, 1),
  (1, 0, 0, 0),
  (1, 1, 0, 1),
  (1, 0, 0, 0) )
```

Square Puzzle

- How do we know if we're at a solution?

Square Puzzle

- How do we know if we're at a solution?

```
def is_vert_solution(state):
  for x in range(len(state)):
    for y in range(len(state[0])):
      first = state[x][0]
      if state[x][y] != first:
        return False
  return True

def is_horiz_solution(state):
  for y in range(len(state[0])):
    for x in range(len(state)):
      first = state[0][y]
      if state[x][y] != first:
        return False
  return True
```

Square Puzzle

- How many next states does each state have?
- How can we get the next states?

Square Puzzle

- How many next states does each state have?
- How can we get the next states?

```
def swizzle(state):
  lstate = as__list(state)
  save = lstate[1][1]
  lstate[1][1] = lstate[1][2]
  lstate[1][2] = lstate[2][2]
  lstate[2][2] = lstate[2][1]
  lstate[2][1] = save
  return as__tuple(lstate)

...

def next_states(state):
  return [ pull_column(state, x) for x in range(len(state)) ]
       + [ pull_row(state, y) for y in range(len(state[0])) ]
       + [ swizzle(state), swozzle(state) ]
```

Demo

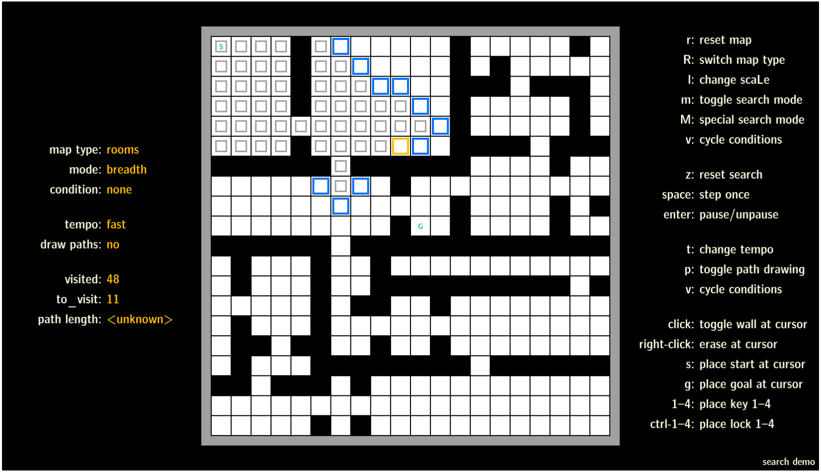

119
120
121
122
search mode: depth
memory: on
colors: 2
tempo: medium
visited: 124
to_visit: 1215
path length: <unknown>
q: quit
r: reset search
R: randomize puzzle
x: toggle search mode
m: toggle memory
c: toggle complexity
space: step once
enter: pause/unpause
t: change tempo
a: auto-cursor
z: reset cursor
←↑↓→: move cursor
search demo

Search With Memory

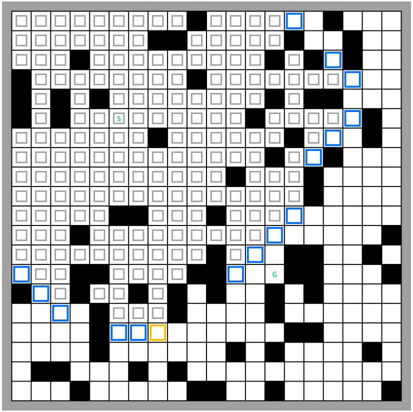
```
def search(state):
  if state.is_goal():
    return [state]
  else:
    result = []

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

    return result
```

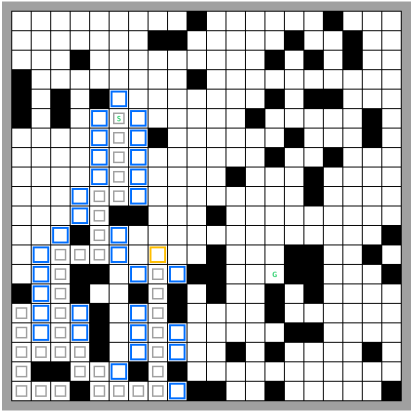
```
def search(state, visited):
  # remember this state
  visited[state] = True

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

    for s in state.next_states():
      # check if it’s already visited
      if not(s in visited):
        result += search(s, visited)

    return result
```

More Demo


S
G
map type: rooms
mode: breadth
condition: none
tempo: fast
draw paths: no
visited: 48
to_visit: 11
path length: <unknown>
r: reset map
R: switch map type
l: change scaLe
m: toggle search mode
M: special search mode
v: cycle conditions
z: reset search
space: step once
enter: pause/unpause
t: change tempo
p: toggle path drawing
v: cycle conditions
click: toggle wall at cursor
right-click: erase at cursor
s: place start at cursor
g: place goal at cursor
1–4: place key 1–4
ctrl-1–4: place lock 1–4
search demo

Breadth or Depth?





Breadth or Depth?

- What is the best case?
- What is the worst case?
- Time? Memory?

Breadth or Depth?

- What is the best case?
- What is the worst case?
- Time? Memory?
- How do these depend on the search space?

Breadth or Depth?

- What is the best case?
- What is the worst case?
- Time? Memory?
- How do these depend on the search space?
- Quality of solutions?

Breadth or Depth?

For a solution at depth d in a space with branching factor B and max depth M:

BFS

- Best case:
  - Consider $\sim B^d$ nodes
  - Remember $\sim B^d$ nodes
- Worst case:
  - Same as the best case
- Features:
  - Consistent (but expensive)
  - Finds shortest paths

DFS

- Best case:
  - Consider d nodes
  - Remember d nodes
- Worst case:
  - Consider $\sim B^M$ nodes
  - Remember $\sim B^M$ nodes
- Features:
  - Inconsistent
  - Can save memory if there aren't cycles

Something More?