Tree Search

Simon D. Levy

CSCI 111

18 March 2008

Two roads diverged in a wood, and I –

I took the one less traveled by,

And that has made all the difference.

Two roads diverged in a wood, and I –

I took the one less traveled by,

And that has made all the difference.

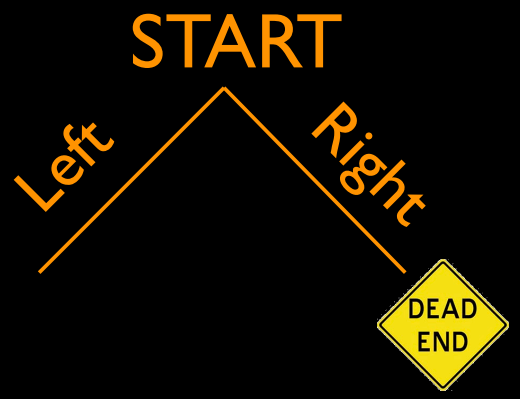
– Robert Frost (1915)

Choices Build Trees

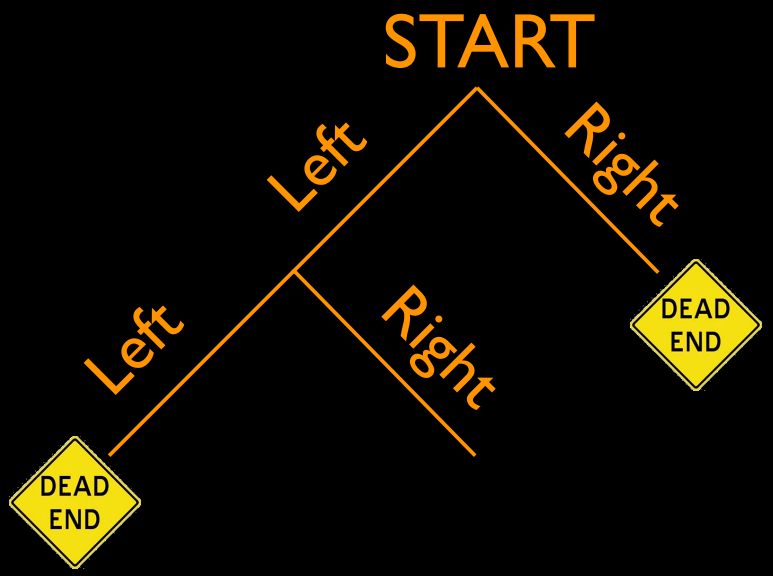
Choices Build Trees

START

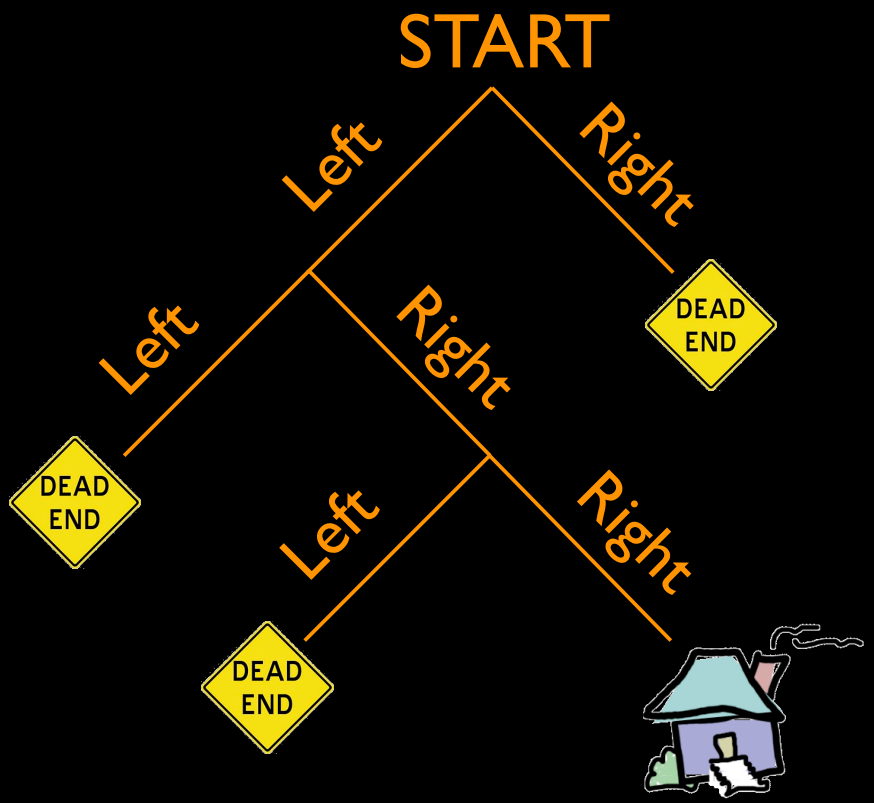
Choices Build Trees



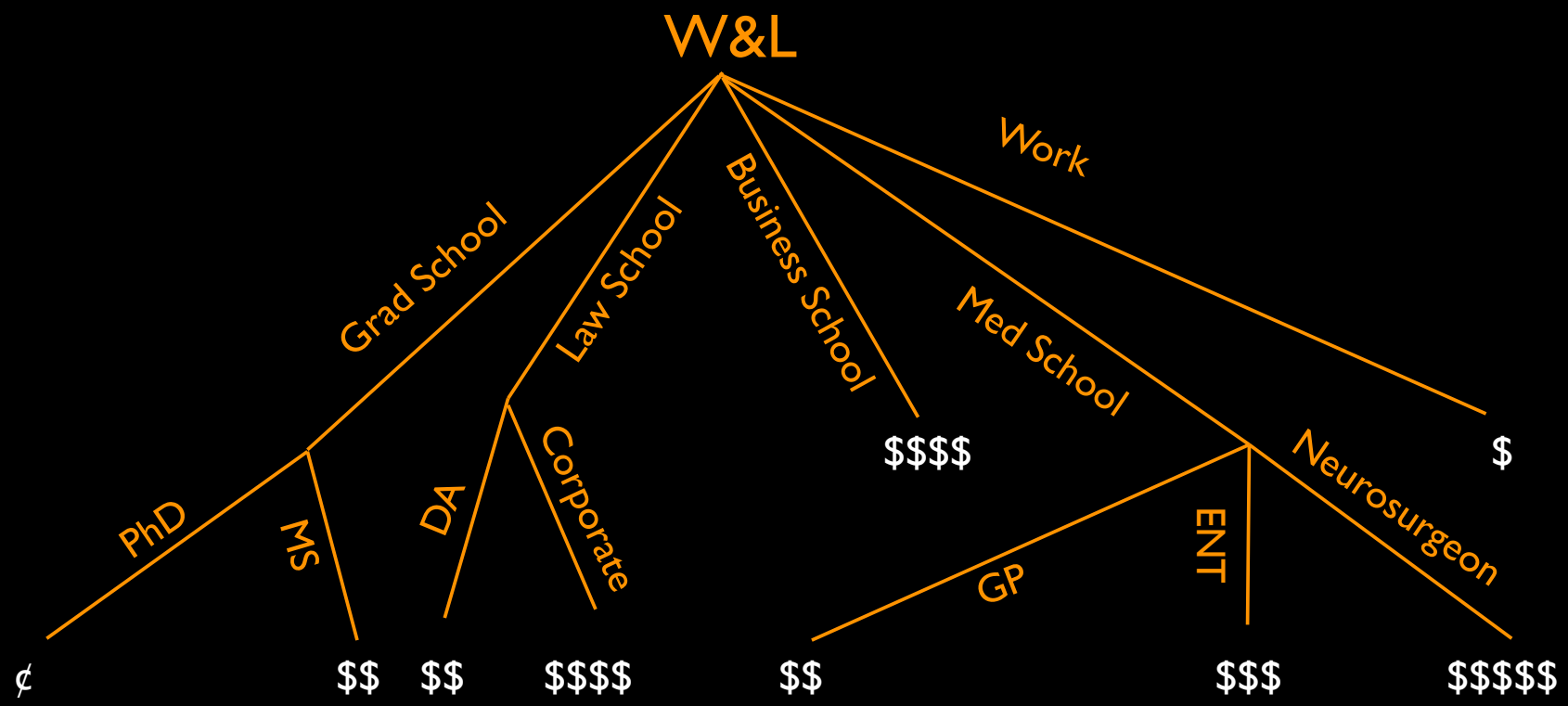
Choices Build Trees



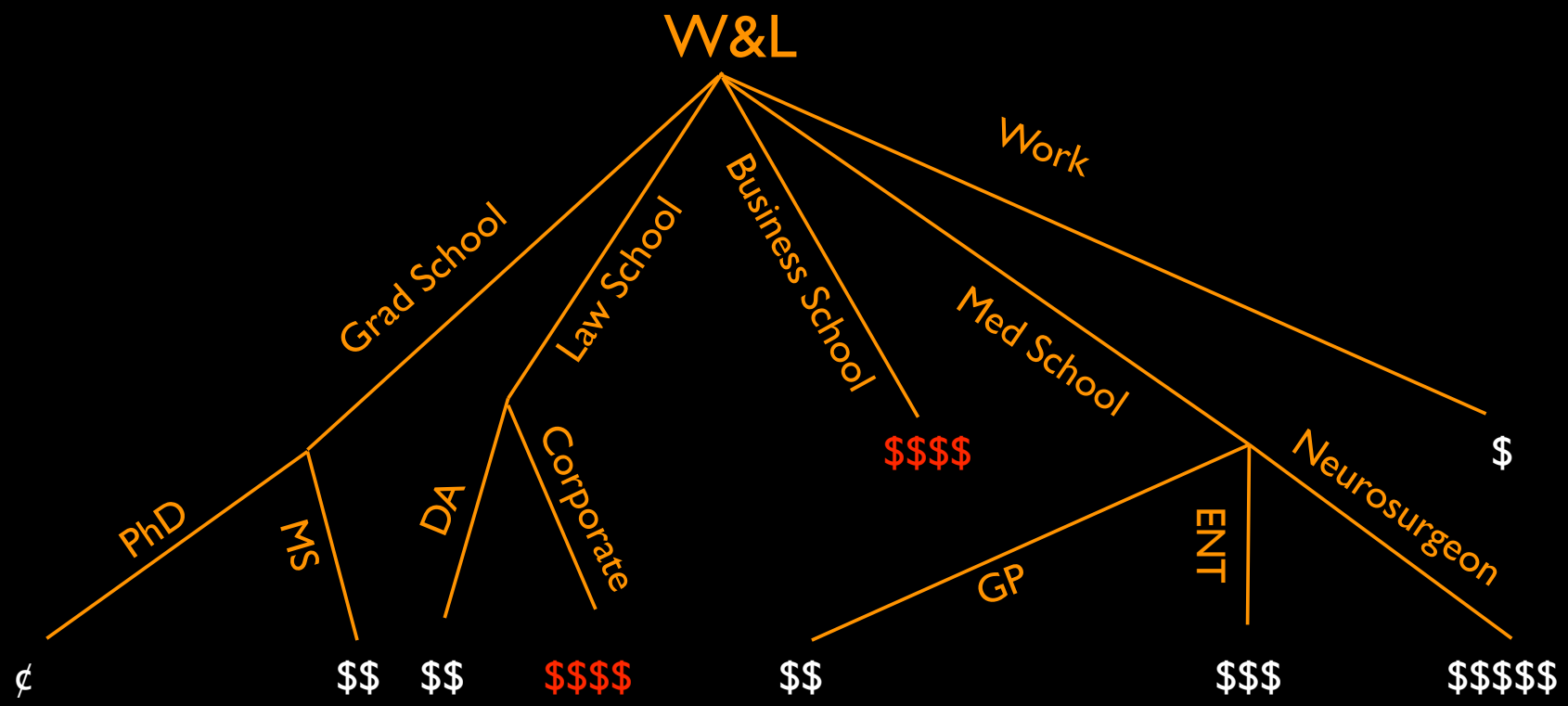
Choices Build Trees



Not Always Two Branches



Tree Search:
"My Goal Is $$$$"

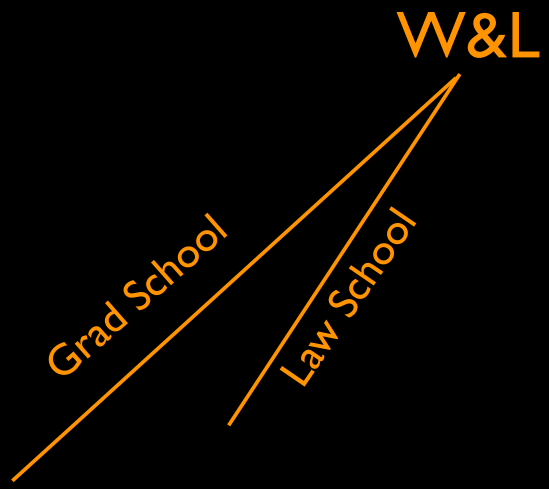


Breadth-First Search

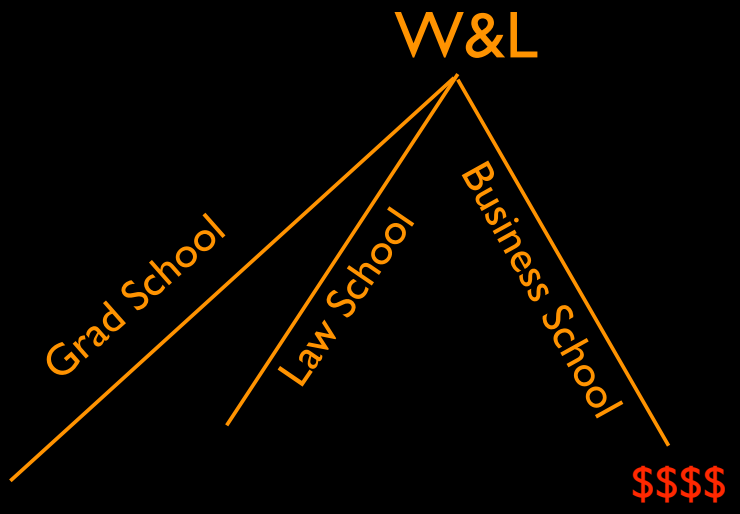
W&L

Breadth-First Search

Breadth-First Search



Breadth-First Search

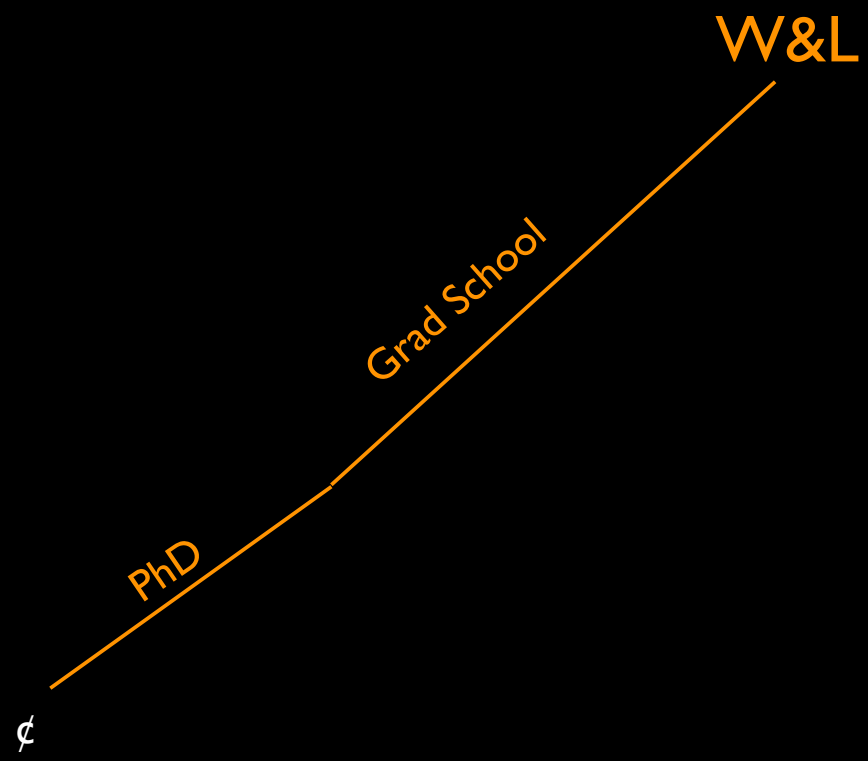


Depth-First Search

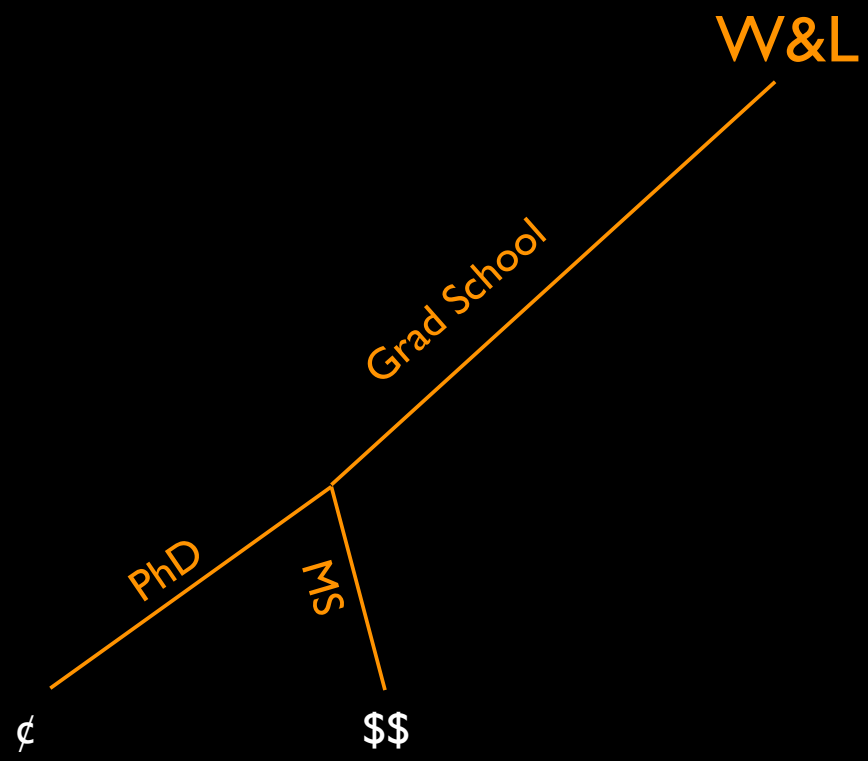
W&L

Depth-First Search

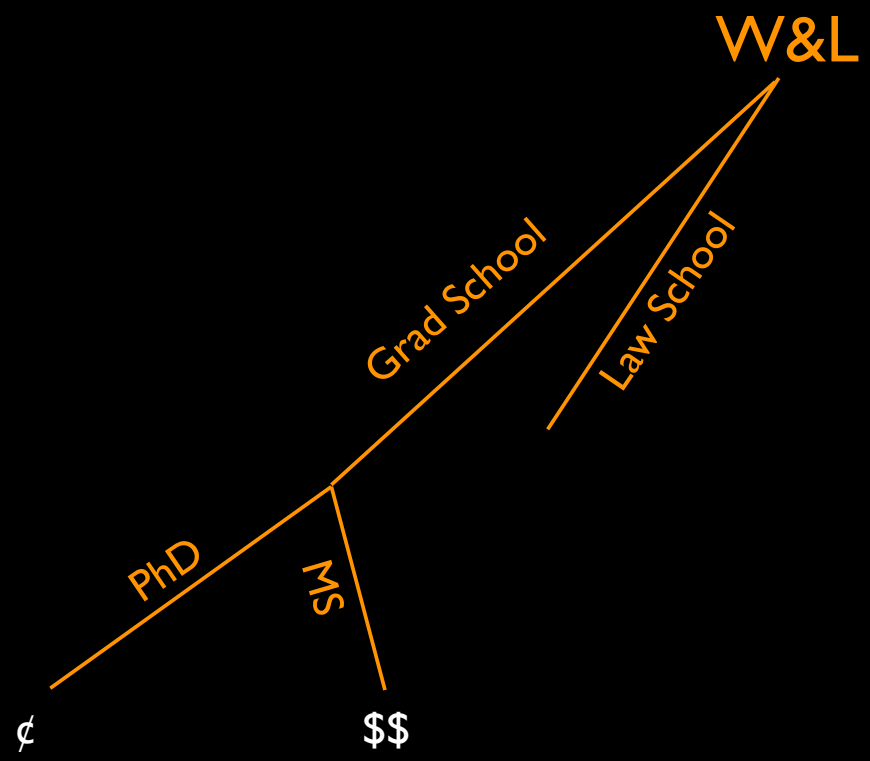
Depth-First Search



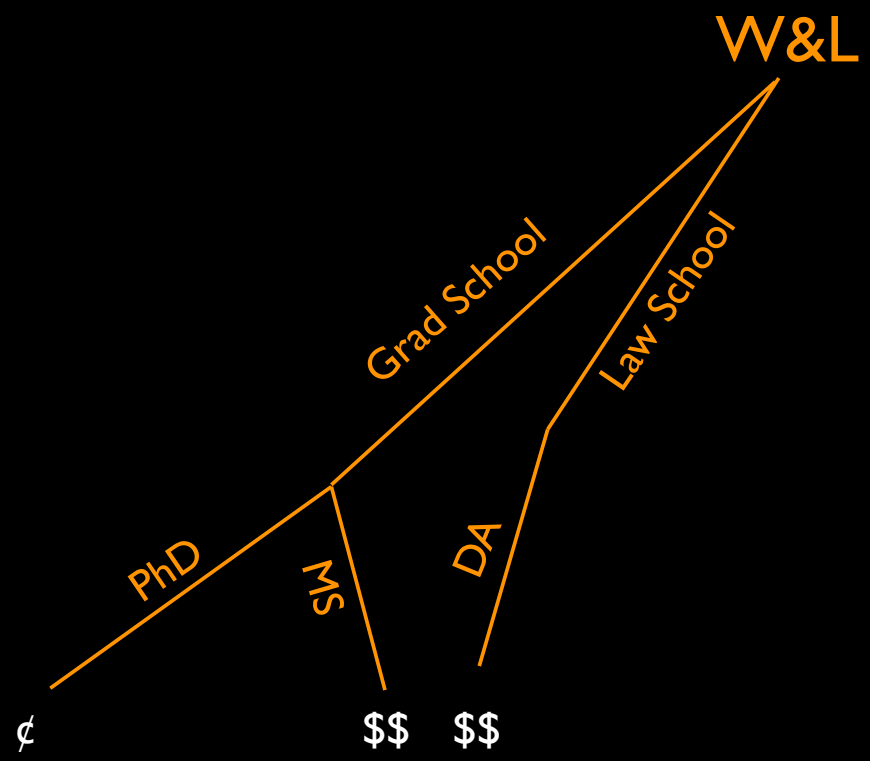
Depth-First Search



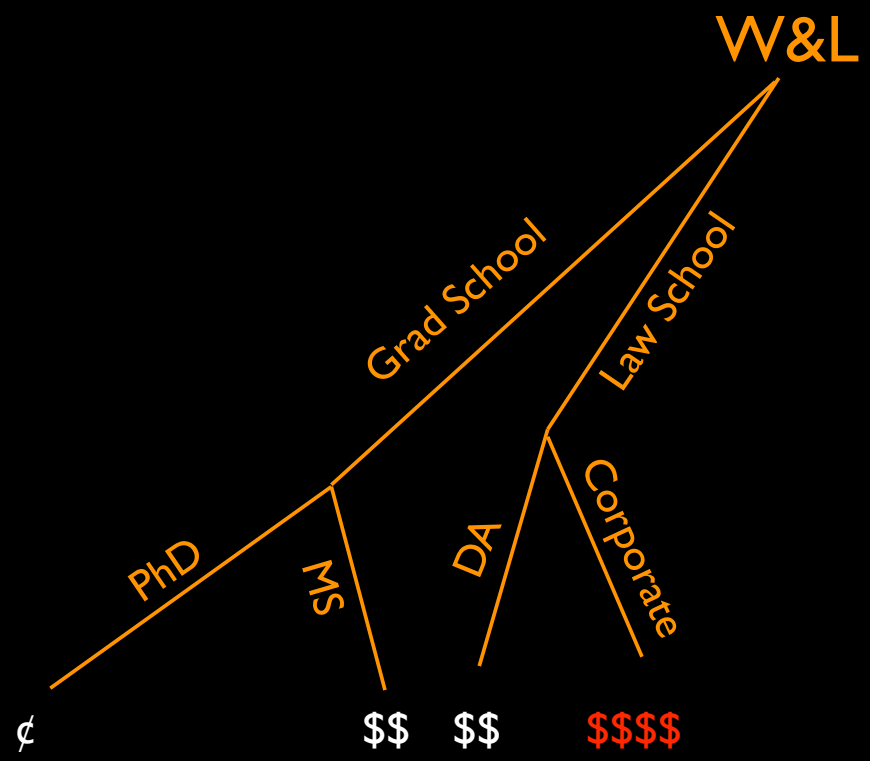
Depth-First Search



Depth-First Search



Depth-First Search



Tree = Function

- Function SUCCESSORS takes an element and returns what can come right after it.
- SUCCESSORS(W&L) =

  [gradschool, lawschool, bschool, medschool, work]
- SUCCESSORS(gradschool) = [ms, phd]
- *etc.*

DFS via Successors

- Get initial state (top of tree) from problem to be solved
- Put initial state into a list
- Repeat until success, or list empty:
  - Remove first state from list
  - If state is a goal, succeed
  - Otherwise, put state's successors at front of list

DFS via Successors

- Put initial state into a list

  [W&L]

DFS via Successors

- Remove first state from list

  [ ] W&L

DFS via Successors

- Put state's successors at front of list

  [gradschool, lawschool, bschool, medschool, work]

DFS via Successors

- Remove first state from list

  [lawschool, bschool, medschool, work] gradschool

DFS via Successors

- Put state's successors at front of list

  [phd, ms, lawschool, bschool, medschool, work]

DFS via Successors

- Remove first state from list

  [ms, lawschool, bschool, medschool, work] phd:¢

DFS via Successors

- Put state's successors at front of list

  [ms, lawschool, bschool, medschool, work]

DFS via Successors

- Remove first state from list

  [lawschool, bschool, medschool, work] ms:$$

DFS via Successors

- Put state's successors at front of list

  [lawschool, bschool, medschool, work]

DFS via Successors

- Remove first state from list

  [bschool, medschool, work] lawschool

DFS via Successors

- Put state's successors at front of list

  [da, corporate, bschool, medschool, work]

DFS via Successors

- Remove first state from list

  [corporate, bschool, medschool, work] da:$$

DFS via Successors

- Replace state wits its successors

  [corporate, bschool, medschool, work]

DFS via Successors

- Remove first state from list

  [bschool, medschool, work] corporate:$$$$

BFS via Successors

- Get initial state (top of tree) from problem to be solved
- Put initial state into a list
- Repeat until success, or list empty:
  - Remove first state from list
  - If state is a goal, succeed
  - Otherwise, put state's successors at back of list

BFS via Successors

- Put initial state into a list

  [W&L]

BFS via Successors

- Remove first state from list

  [ ] W&L

BFS via Successors

- Put state's successors at back of list

  [gradschool, lawschool, bschool, medschool, work]

BFS via Successors

- Remove first state from list

  [lawschool, bschool, medschool, work] gradschool

BFS via Successors

- Put state's successors at back of list

  [lawschool, bschool, medschool, work, phd, ms]

BFS via Successors

- Remove first state from list

  [bschool, medschool, work, phd, ms] lawschool

BFS via Successors

- Put state's successors at back of list

  [bschool, medschool, work, phd, ms, da, corporate]

BFS via Successors

- Remove first state from list

  [medschool, work, phd, ms, da, corporate]
  bschool:$$$$

DFS *vs.* BFS:

- DFS can get stuck going down a "rabbit hole", esp. when states repeat (so run out of time)
- Consider a tree with three states A, B, C
  - Goal at C
  - SUCCESSORS(A) = [B, C]
  - SUCCESSORS(B) = [A]

DFS *vs.* BFS:

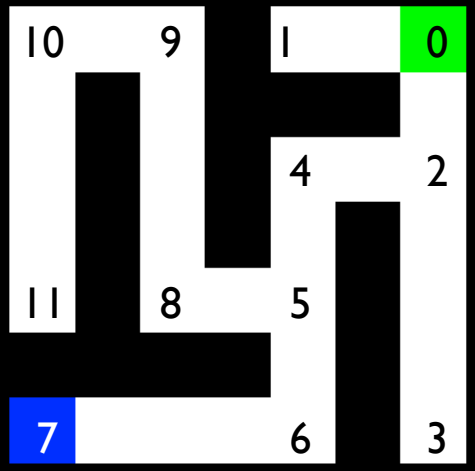
- BFS will always find a goal, but can make the list grow explosively (so run out of memory)
- Consider a very large tree with the goal at the bottom
  - SUCCESSORS(A) = [B, C]
  - SUCCESSORS(B) = [D, E]
  - SUCCESSORS(C) = [F, G]
  - *etc.*

DFS *vs.* BFS: Conclusion

- Use DFS with some kind of "safety" check
  - Limit the depth
  - Check for repetitions

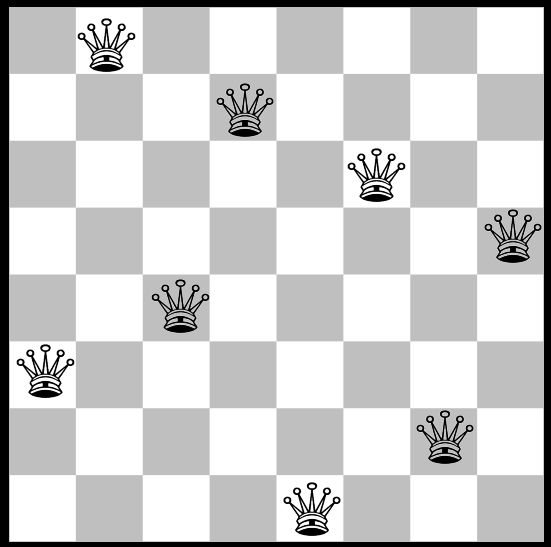
Trees for Games: Maze Navigation

- SUCCESSORS(0) = `[1,2]`
- SUCCESSORS(2) = `[0,3,4]`
- *etc.*



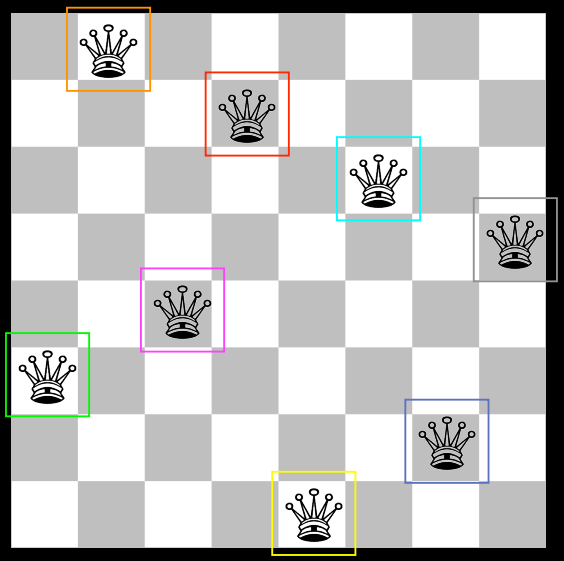
Trees for Games: Eight Queens Puzzle

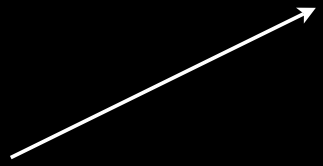
- Put one Queen in each column
- No Queen may attack any other
- Represent board as list of filled positions



Trees for Games: Eight Queens Puzzle

- Put one Queen in each column
- No Queen may attack any other
- Represent board as list of filled positions





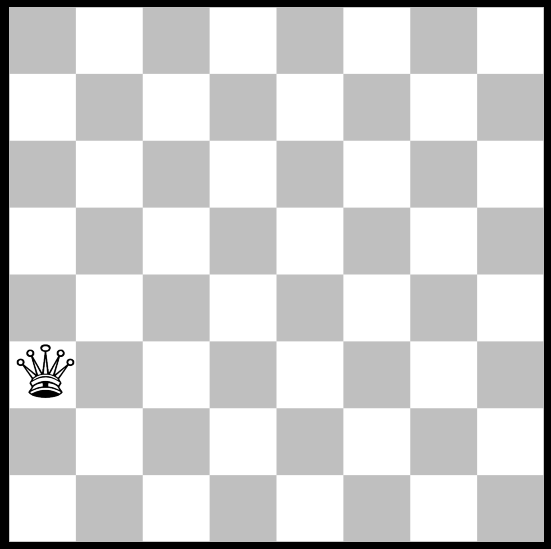
```
[ (0,2), (1,7), (2,3), (3,6), (4,0), (5,5), (6,1), (7,4) ]
```

Trees for Games: Eight Queens Puzzle

- Successor Function #1: Put a Queen in any open position
- SUCCESSORS(`[0,2]`) =

  `[0,2,0,0], [0,2,0,1],`

  `[0,2,0,3], ..., [0,2, 7,7]`
- Whaddya think ?



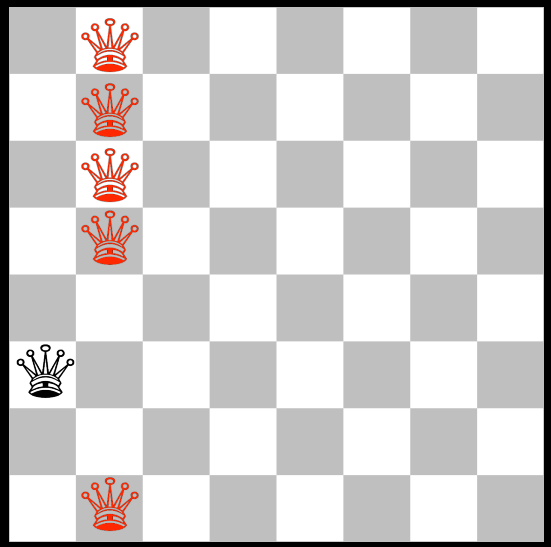
Trees for Games: Eight Queens Puzzle

- Successor Function #2: Put a Queen in a leftmost unattacked position
- SUCCESSORS([0,2]) =

  [0,2,1,0], [0,2,1,4],

  [0,2,1,5], [0,2,1,6],

  [0,2, 1,7]



Let's Write "Smart" DFS in Python

```
def DFS(problem):
  # Get initial state from problem
  # Put initial state into a list
  # Repeat:
      # If list is empty, return failure
      # Remove first state from list
      # If state is a goal, return success
      # Get successors of state
      # XXX Re-order successors with new ones at front
      # Insert successors at front of list
```