Search

Fall Semester

Topic Outline

- Statement of the Problem
  - Input
  - Output
- Examples
- Graph Traversals
  - Depth-first
  - Recursion
  - Breadth-first
  - FIFO queue
- Tracing Step-by-Step

- Test cases
- Improvements
  - Iterative deepening
  - Bidirectional
- Connected Components
  - Big-Oh Analysis
  - Flood fill
  - Floyd-Warshall
  - Dynamic programming
- Up Next: Romania

Statement of the Problem

- Input. A dictionary of six-letter words, the starting word and the ending word.

- Output. A sequence of words such that:
  - The first word is the starting word.
  - The last word is the ending word.
  - All the words are unique.
  - Consecutive words differ by exactly one letter.

- If no such sequence exists then output NO MATCH.

- Note. It's not enough to just find the ending word because all the words *in the entire path* must be displayed.
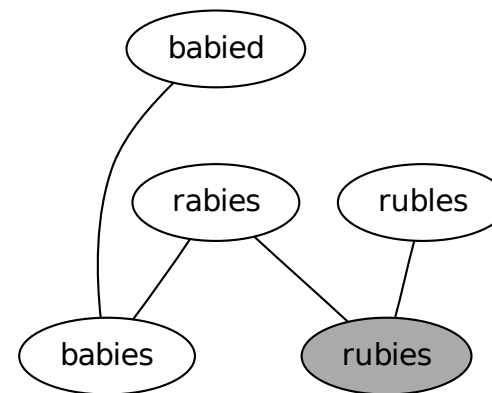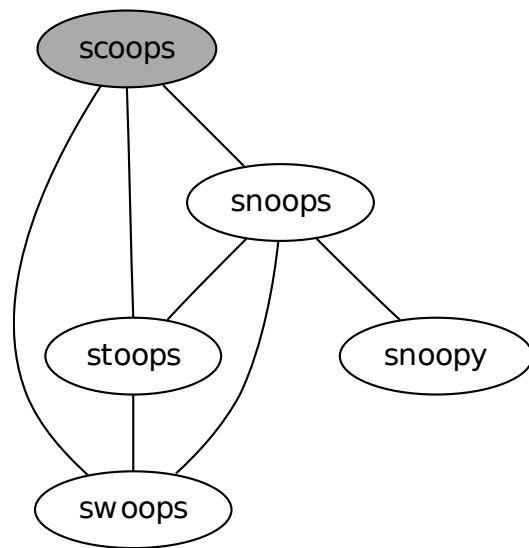
Example: FROM battle TO castle
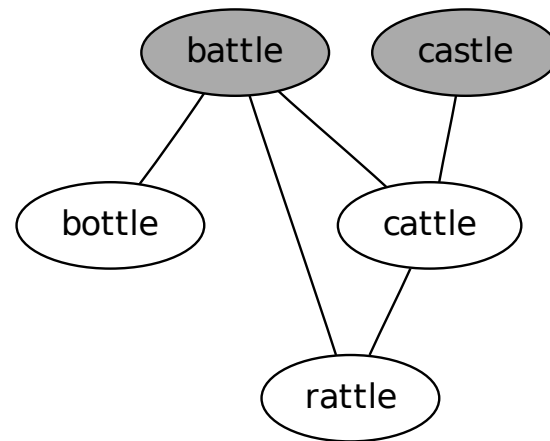
- PATH: battle, cattle, castle

- PATH: battle, rattle, cattle, castle

Example: FROM radius TO motion
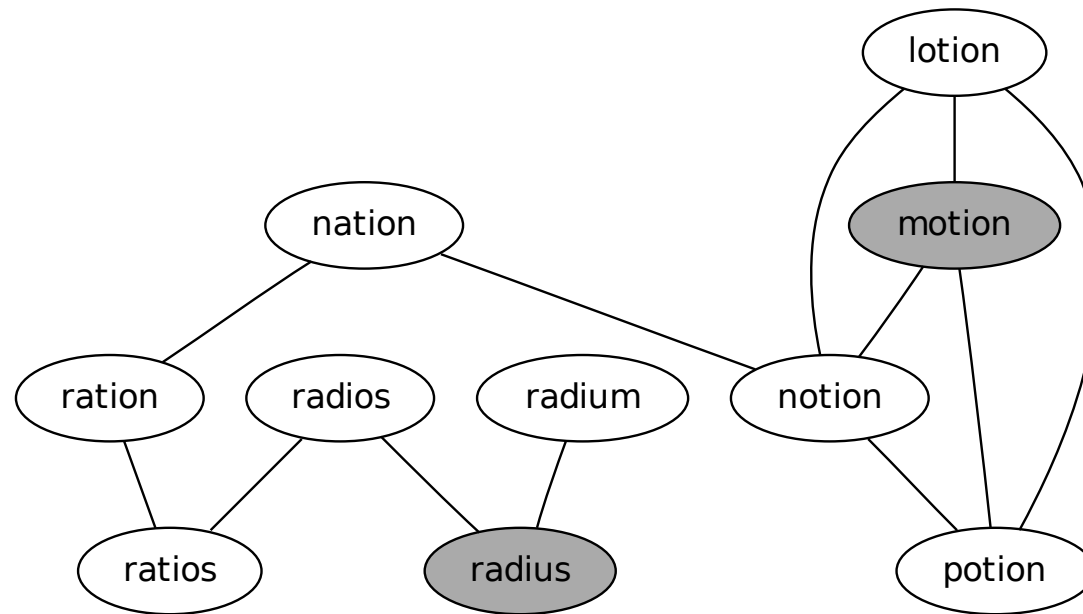
- PATH: radius, radios, ratios, ration, nation, notion, motion

Example: FROM scoops TO rubies

- NO MATCH

Depth-First Search (1)

- Recursion. Recursion. Recursion.

- At each step check the *deepest* node that has been seen but not yet checked; same as when you first learned trees.

- By "seen" we mean that some neighbor of the node has already been checked and by "checked" we mean that a node has been tested for equality with the target (e.g., the ending word).

- Leads to minimax for game playing.

- Backtracking has very low memory requirements but may run for a long, long time and still produce a suboptimal result.

- See also, Russell and Norvig page 75.

Depth-First Search (2)
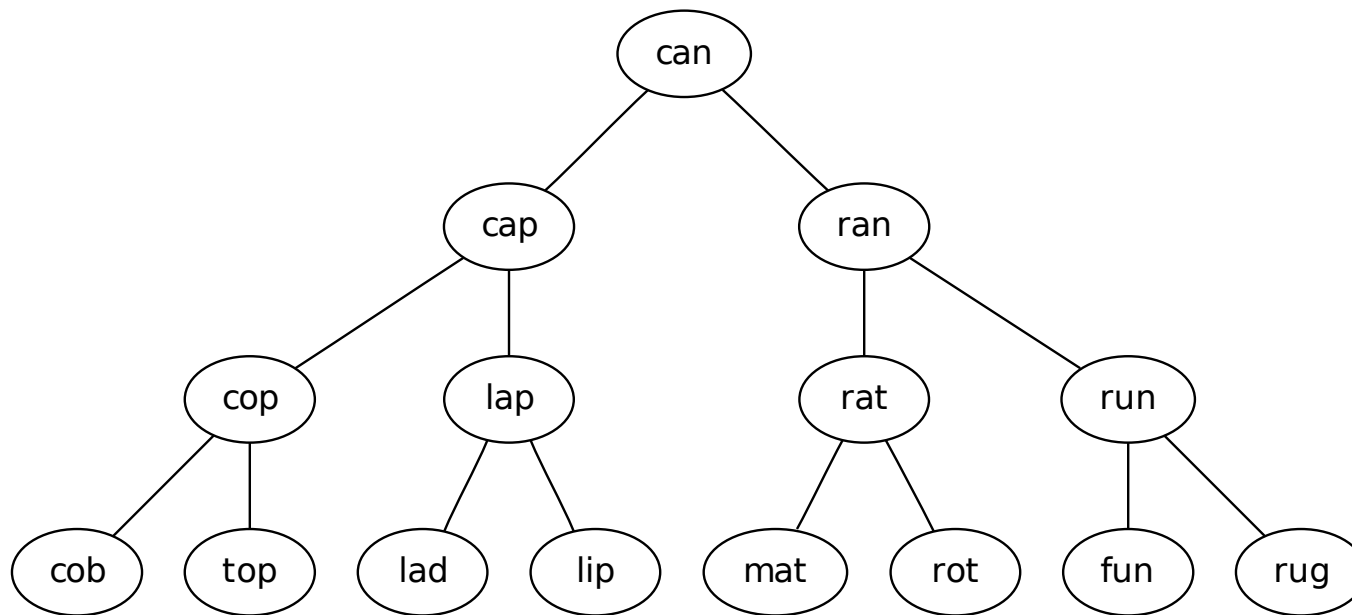
- can cap cop cob top lap lad lip ran rat mat rot run fun rug

DFS Pseudocode

```
*** BINARY TREE ***        *** GENERAL CASE ***
IF word is target          IF word is target
   THEN success               THEN success
ELSE IF dead end           ELSE IF dead end
   THEN failure               THEN failure
TRY left child             LOOP over neighbors
   STOP if succeeds           TRY neighbor
TRY right child                  STOP if succeeds
   STOP if succeeds        END LOOP
OTHERWISE failure          OTHERWISE failure
```

Breadth-First Search (1)

- No recursion. Rather, a queue and a loop.

- At each step check the *shallowest* node that has been seen but not yet checked; same as the print-by-level tree problem.

- Finds the optimal (i.e., shortest) path on an unweighted graph.

- Proof. We consider shorter paths before longer paths. Thus, when first we find a solution, if there had been a shorter solution then we would have already found it!

- Queue of paths may require an excessive amount of memory so keep track of nodes already seen and don't look at them again.

- See also, Russell and Norvig page 73.

Breadth-First Search (2)
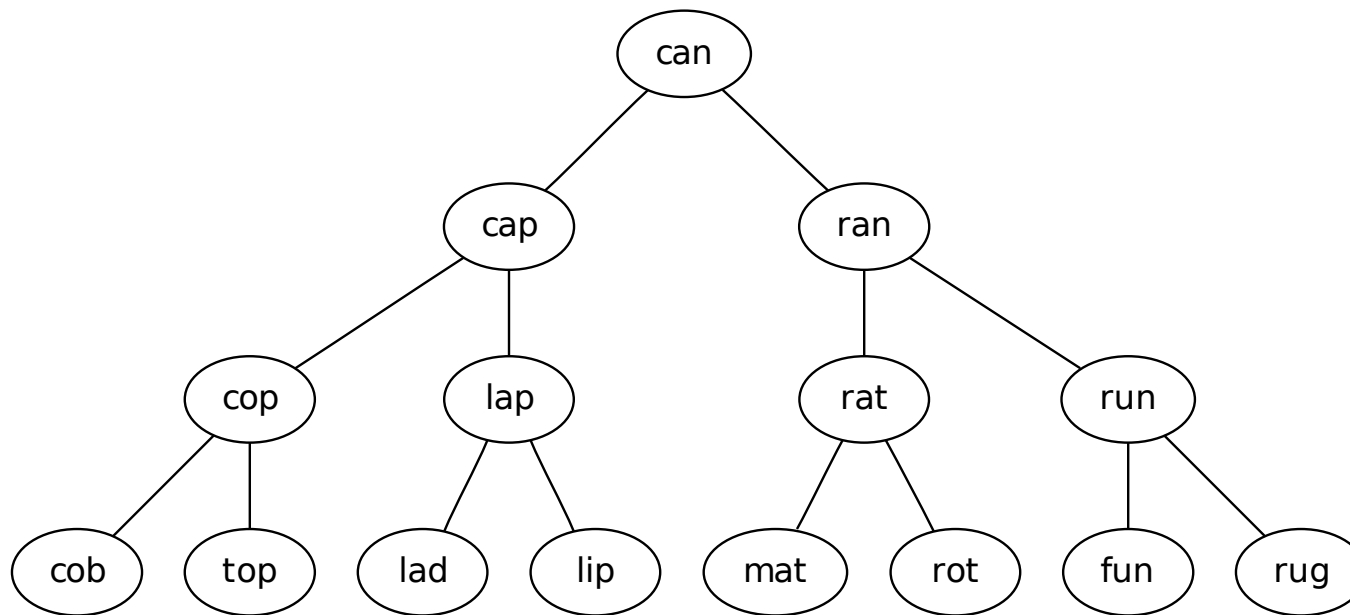
- can cap ran cop lap rat run cob top lad lip mat rot fun rug

BFS Pseudocode

```
CREATE a path containing only the starting word
CREATE a queue containing only this path
WHILE the queue is not empty
    GET word from end of path at front of queue
    IF word is target THEN success
    LOOP over neighbors
        CREATE a copy of word's path
        ADD neighbor to this copy
        ADD copy to end of queue
OTHERWISE failure
```
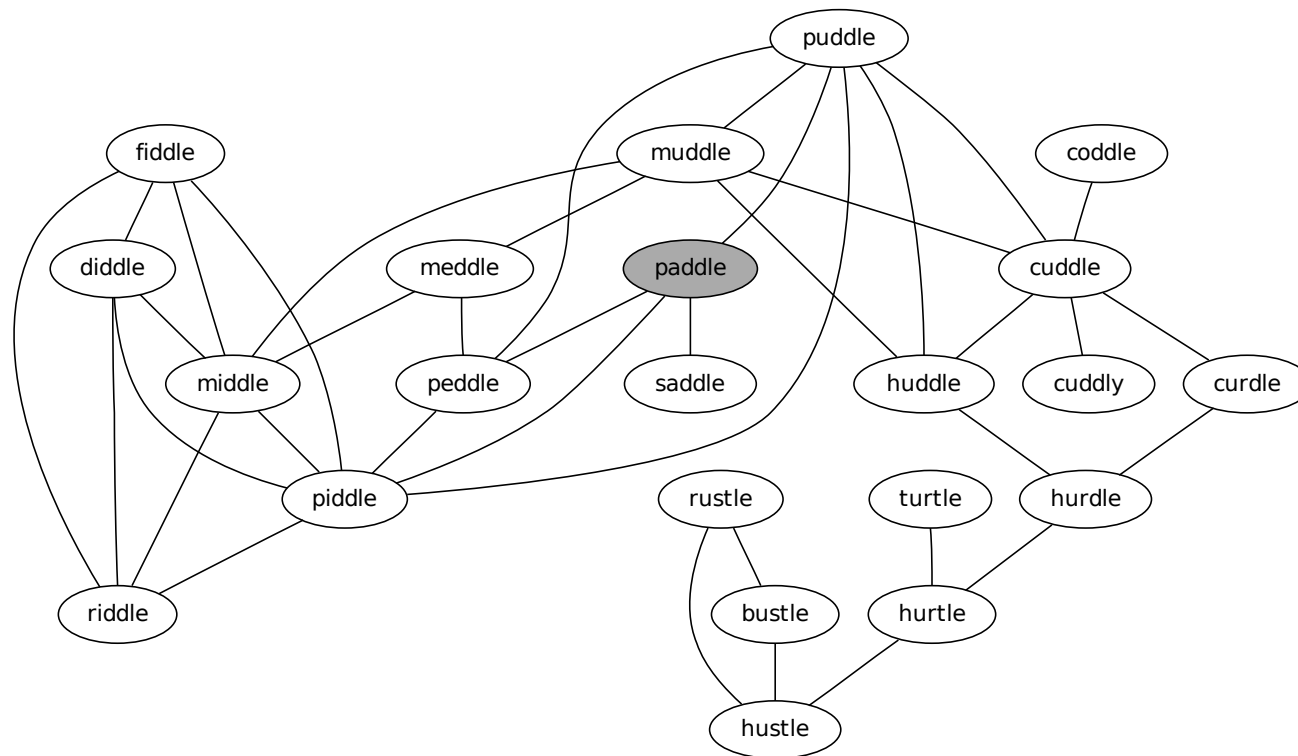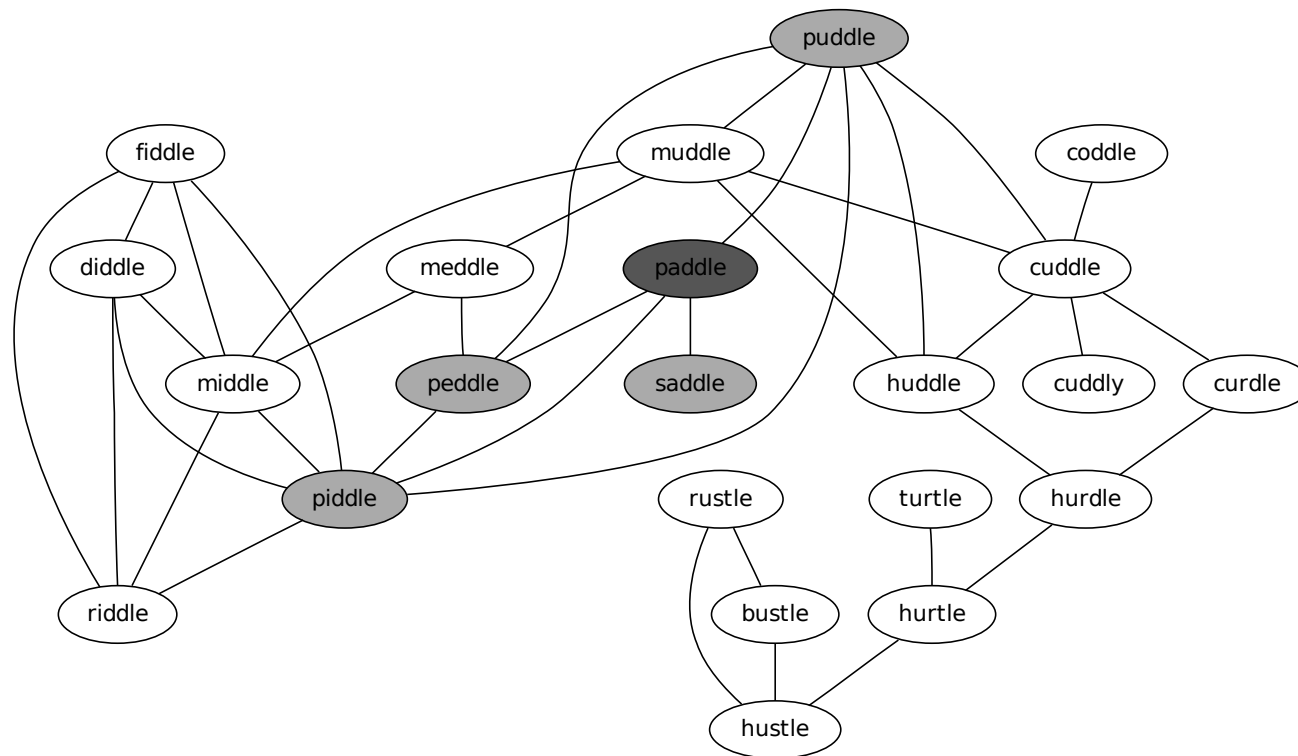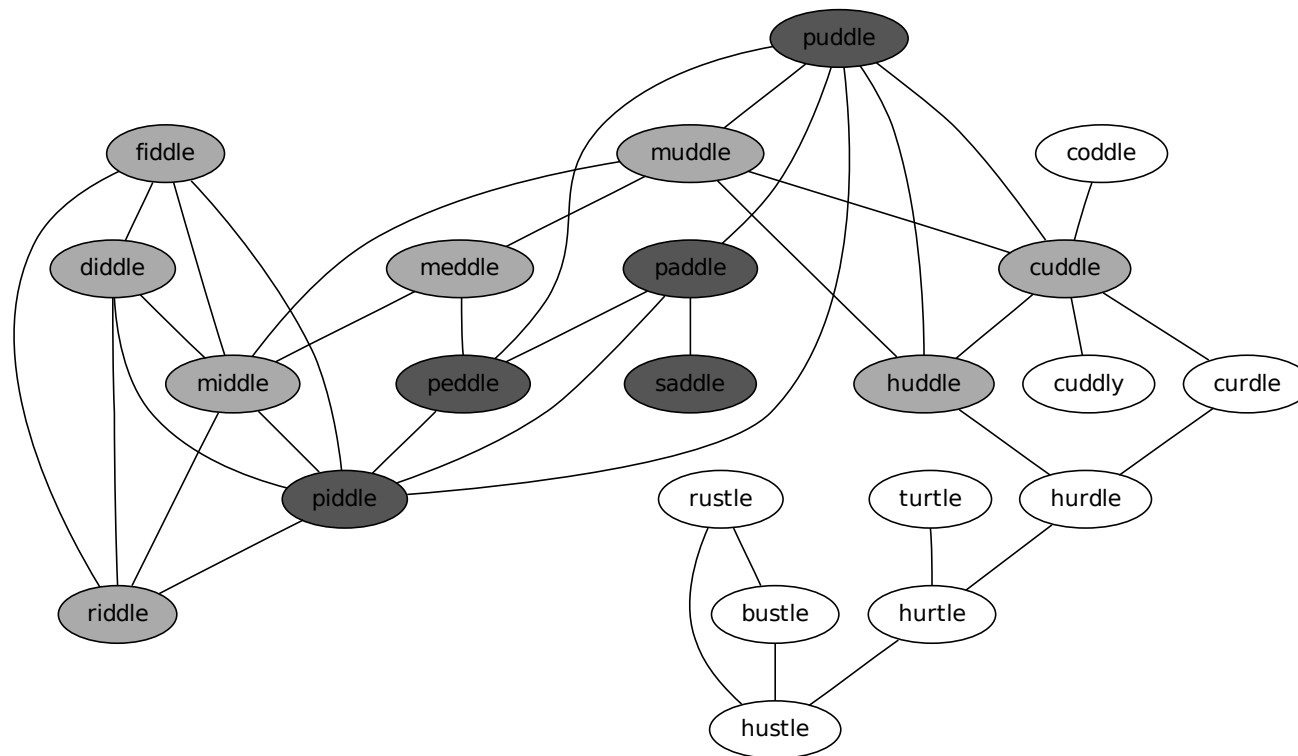
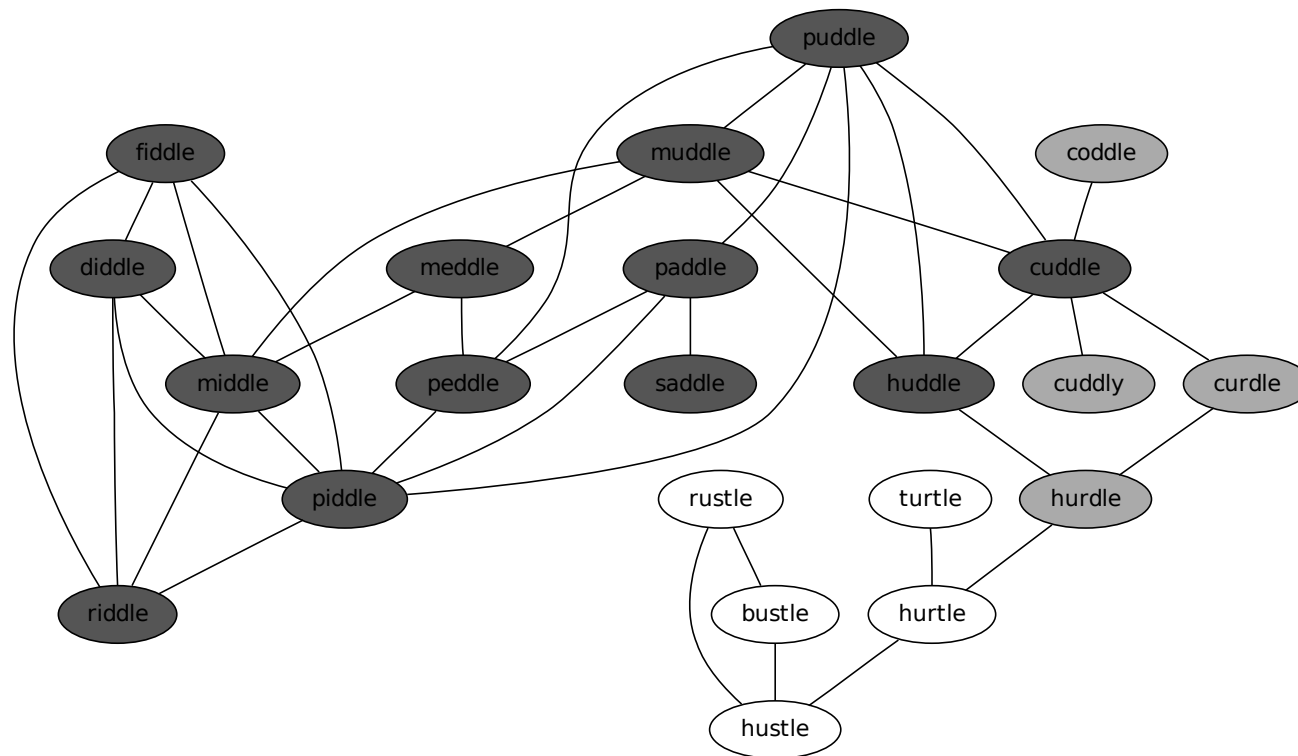# Example: FROM paddle TO turtle (1)

Example: FROM paddle TO turtle (2)
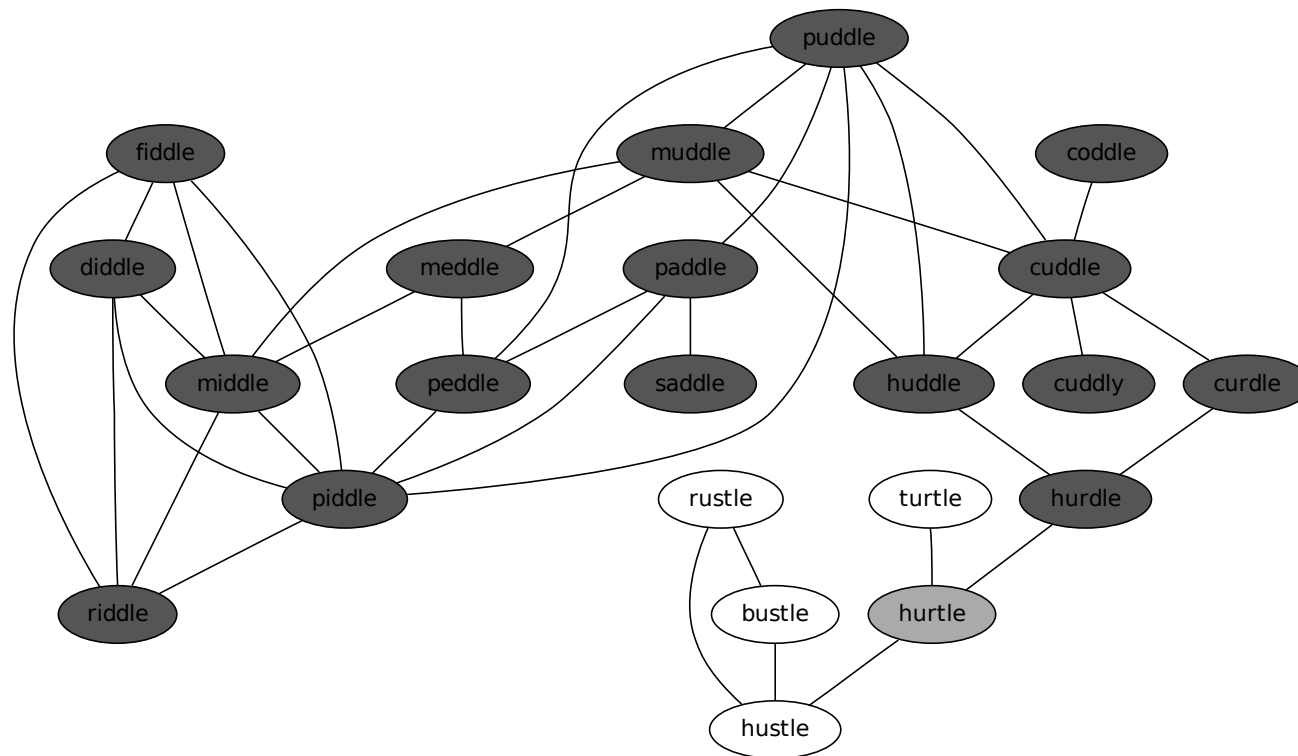
Example: FROM paddle TO turtle (3)

Example: FROM paddle TO turtle (4)

Example: FROM paddle TO turtle (5)

Example: FROM paddle TO turtle (6)
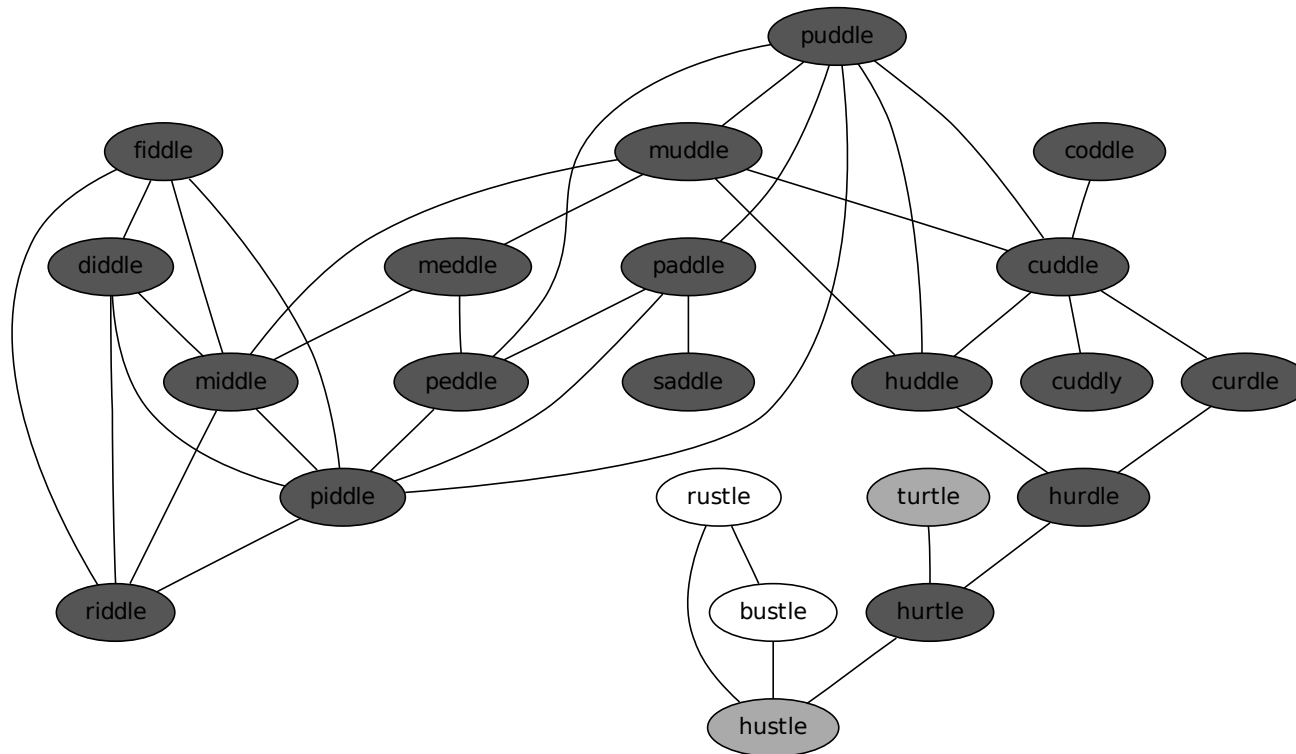
Test Cases

```
battle castle              chorus singer

scoops rubies              winter storms

hooked relics              better grades

relics hooked              crafty wolves

spider clucks              sliver silver
```

- To test backtracking consider how neighbors are orderd:
  - Alphabetical is predictable but random is more interesting.
  - Or by number of neighbors or letters matching target or...
- Have a good test case? Let us know!

DFS with Iterative Deepening (1)

- Recursion. No queue. A loop.

- Memory like DFS. Shortest path like BFS.

- Trick. Limit the depth of the recursion.

  - Start with a limit of zero.
  - Run a depth-limited search.
  - If this search fails then *increase the limit.*
  - Repeat.

- See also, Russell and Norvig page 78.

- For depth-limited search, Russell and Norvig page 77.
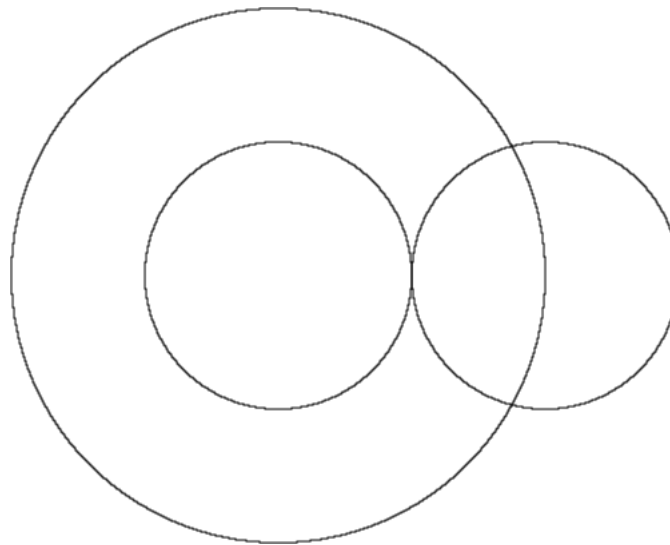
DFS with Iterative Deepening (2)
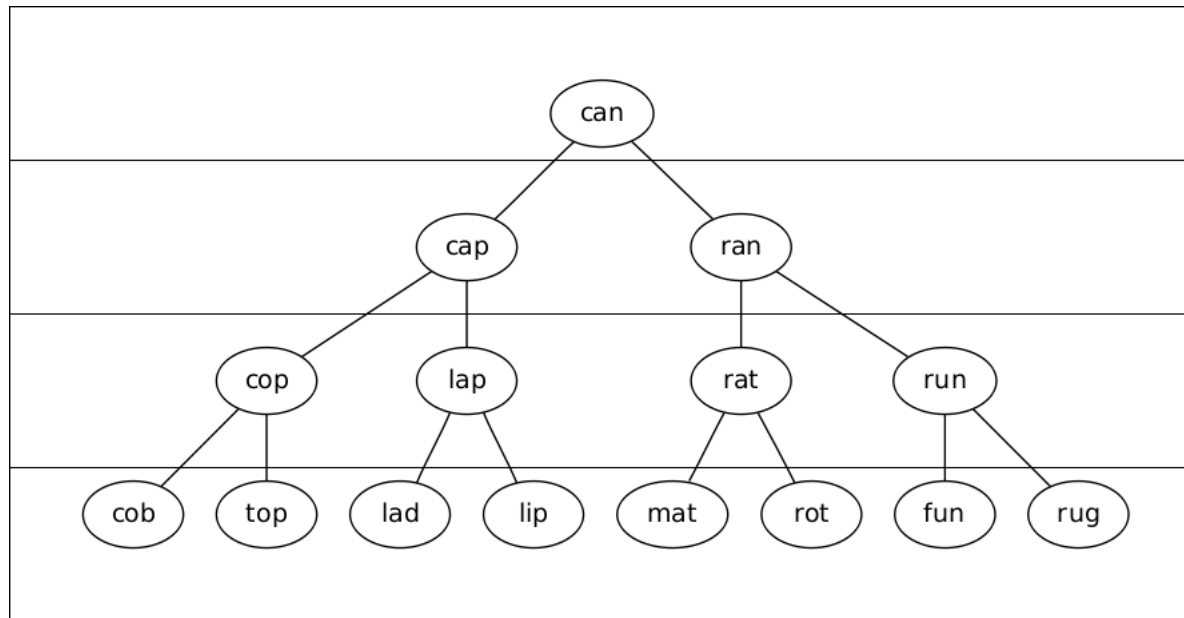
- can, can cap ran, can cap cop lap ran rat run, can cap cop cob top lap lad lip ran rat mat rot run fun rug

Bidirectional Search

- Do a BFS from the starting word *and* from the ending word. Determine when the frontiers intersect. Smaller search space.

- See also, Russell and Norvig page 79.

Example Program: DFS on a Binary Tree

- Version 1.0

  - Node class.
  - Actually construct the tree.
  - Familiar recursive method.

- Version 1.1

  - More like the general case.
  - Do not construct the tree.
  - Build a neighbor list instead.
  - Use a hash table.
  - Assumes directed edges away from the root.

Lab Assignment: DFS, BFS, DFSID

- Implement the following search algorithms:

  – Depth-first search.

  – Breadth-first search.

  – Depth-first search with iterative deepining.

- Let the user choose which puzzle to solve and which search algorithm to use. Additional algorithms are also encouraged!

- Output:

  – The solution path found or NO MATCH.

  – For testing purposes also print a trace of DFS showing the backtracking and a trace of BFS showing the queue.

Connected Components (1)

- How many nodes are there in each component?

- How many edges?

- How do the diameters compare?

  – Find the shortest path distance between each pair of nodes.
  – The diameter is the longest such shortest path, the worst-case distance between any two nodes in the component.

- What is the average branching factor?

  – The branching factor tells how many neighbors a node has.
  – If this is not uniform then we consider the average instead.

Connected Components (2)

- Assume we label each component with an arbitrary number.
- The absolute worst case diameter is $D = 27$ in group 30.
  - From vaguer to drifts (or to crafty, or to drafty).
  - Even though $V = 1619$ in group 30.
- Likewise $V = 185$ in group 44 and $D = 14$ only.
- However $V = 127$ in group 43 but $D = 20$ still.
- Compare this to modern air travel where the worst-case trip from Dulles to Moscow takes maybe half a dozen hops.
- Once we know these upper bounds we can improve performance by using a depth-limited search in place of the straight DFS.

Connected Components (3)

- Time analysis from June 2008.

- It took me 7 seconds to find the diameter for all components except group 30. It took under 1/4 of a second if groups 43 and 44 were also excluded. For just group 30 it took 51 minutes.

- So groups 43 and 44 with $V \approx 100$ took $T \approx 3$ seconds.

- And group 30 with $V \approx 1000$ took $T \approx 3000$ seconds.

- Note the cubic behavior. As the number of nodes goes from 100 to 1000 the time goes from 3 seconds to 3000 seconds.

$$10 \cdot V \sim 10^3 \cdot T$$

Connected Components (4)

- The connected components were numbered using flood fill.

- The diameters were determined from the all-pairs-shortest-path output of the Floyd-Warshall algorithm; complexity $O\left(V^3\right)$.

- Big-Oh Analysis[1]. If b is the branching factor, m the maximum depth, and d the depth of the shortest path solution, then:

  - DFS is $O\left(b^m\right)$ in time and $O\left(bm\right)$ in space.
  - BFS is $O\left(b^{d+1}\right)$ in both time and space.
  - DFSID is $O\left(b^d\right)$ in time and $O\left(bd\right)$ in space.

---

[1]Russell, Stuart, and Peter Norvig. Artificial Intelligence: A Modern Approach. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2003. 81.