trees, cont'd  \((see\ handouts)\)

search

Dr. Kimberle Koile
graphs vs trees

graph: set of nodes (aka vertices) and links (aka edges); can be directed or undirected

tree: connected, acyclic graph; every finite tree has a root (top) node

the problem of search is finding a goal node (or nodes); aka finding a path to a goal node.
e.g. depth, breadth, best-first

Key difference between them:

**Queue** (queue keeps track of nodes to be checked)

1. Picking element from the queue
2. Adding new elements to the queue

**Depth-first**: remove from front, add node's children to the front

**Breadth-first**: remove from front, add to back

**Best-first**: either remove from front, or maintain a sorted list, or remove first from anywhere
search trees

- can think of each node as a partial path through the tree; e.g., I represents S-C-
- "expanding a node" means removing a node from the queue and adding its children (if it's not the goal node); aka "expanding a partial path"
Search problems are often represented by graphs.

1. Draw a tree that corresponds to this graph.
depth first search

What is the order in which nodes are explored (assuming a left to right algorithm)

A B E F G K L H C I J D
What is the order in which nodes are explored (assuming a left to right algorithm)?

A B C D E F G H I J K L
(define (search start-state done?)
  (define (search-help queue)
    (if (null? queue)
        #f
        (let ((current (car queue)))
          (if (done? current)
              current
              (search-help (append (children current) (cdr queue)))))))
  (search-help (list start-state)))

Mark the line of code that determines the search method.

Write a new line of code here that changes the search method.

breadth: (append (cdr queue) (children (car queue)))
best: (merge (sort (children (car queue))) (cdr queue))
beam: (list-head n (merge (sort (children (car queue))) (cdr queue)))

What's the new method?

NOTE: The beam search code below differs slightly from that in lecture. Beam search should look at all items in the list, then take the best n. (The lecture code just took the best n of the children.)
What factors influence the outcome?
- Size + shape of tree
- Order in which nodes are expanded (e.g., left to right)

What search algorithm works best with:
- A wide, shallow tree? *depth-first* (can get stuck in small branches)
- A narrow, deep tree? *breadth-first* (could get stuck in very long branches)

How could the search algorithm for the robot problem be more clever?
- Keep track of which nodes it has expanded
- Use A* search: heuristics of "best so far" + "best estimate to the goal"