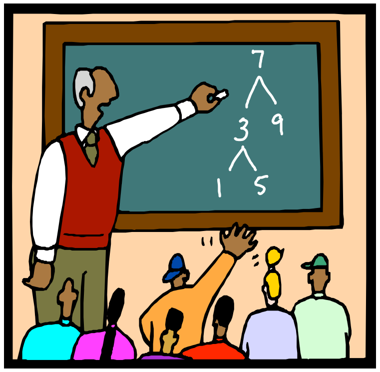
6.001 recitation 13 4/04/07

- 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 n (or nodes); aka finding a path to a goal node.

search

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

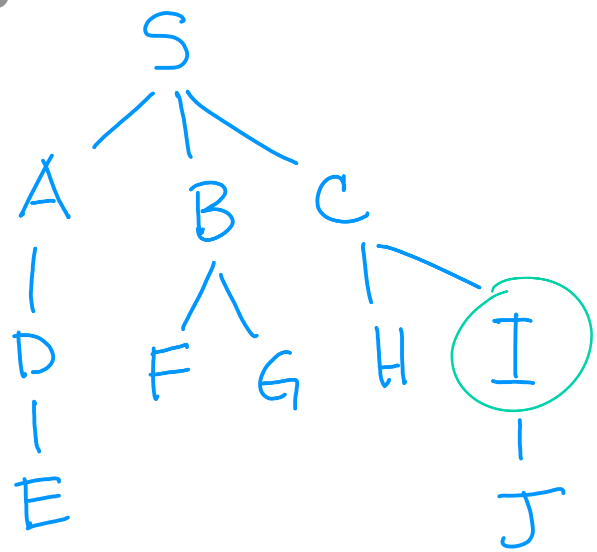
depth-first : remove from front, add node's children to the front

breadth first : remove from front, add to back

best-first : either remove from front, + maintain a sorted list

or remove best from anywhere

search trees

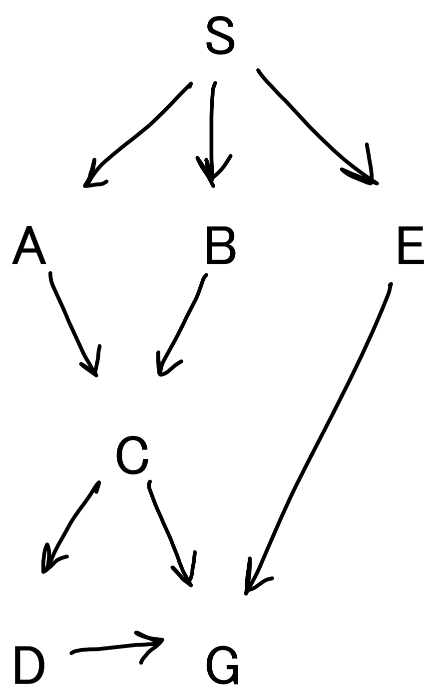


node I
aka partial path
S–C–I

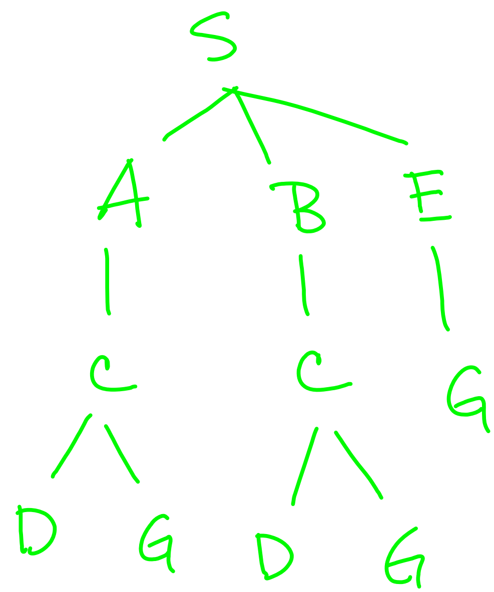
terminology:

- can think of each node as a partial path through the tree; e.g. I represents S–C–I
- "expanding a node" means removing a n… from the queue and adding its childr… (if it's not the goal node); aka "expanding a partial path"

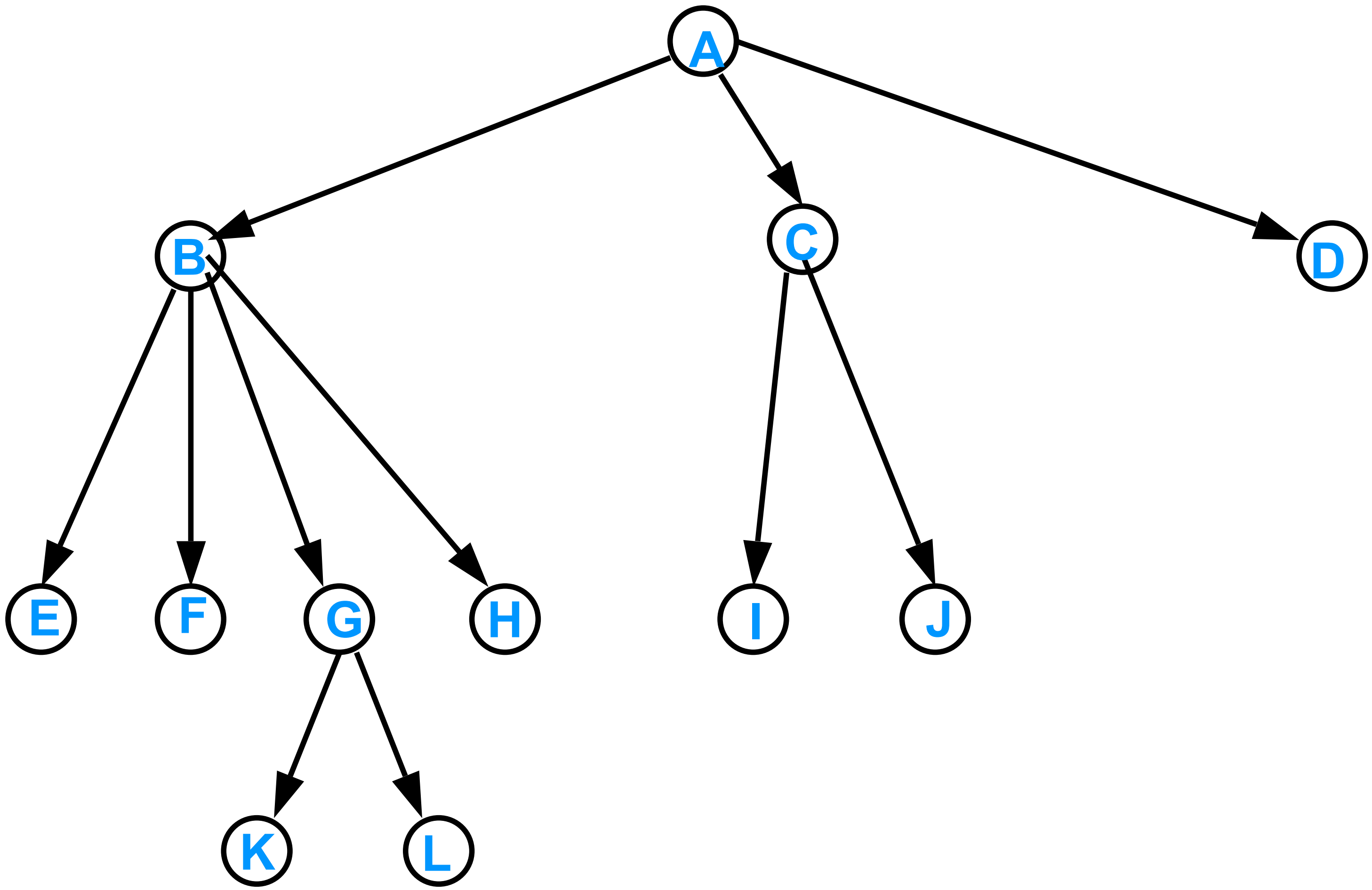
Search problems are often represented by graphs



1. Draw a tree that corresponds to this graph.



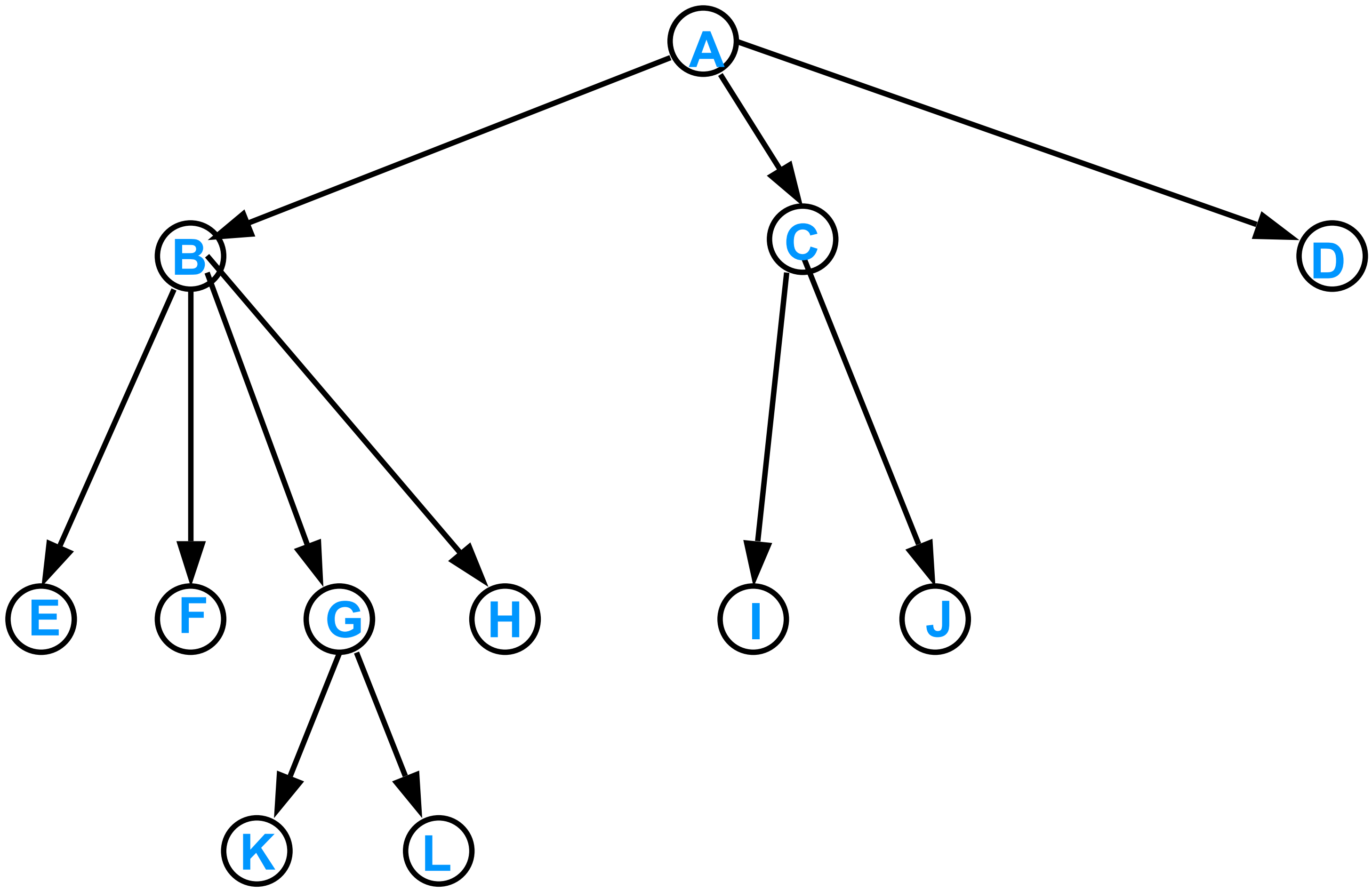
depth first search



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

breadth first search



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

search

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.)

```
(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)))
```

depth-first: childrens nodes put on front of queue

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 (qeue))) (cdr queue))
beam: (list-head n (merge (sort (children (car queue))) (cdr queue)))

← or sort entire q (less efficient)

What's the new method?

search

What factors influence the outcome?

- size + shape of tree
- order in which search (e.g. left to right)

What search algorithm works best with

- a wide, shallow tree? depth-1st 
- a narrow, deep tree? bread-1st (could get stuck down very long branch)

How could the search algorithm for the robot problem be more clever?

- keep track of which nodes it had expanded
- use A* search: heuristic of "best so far" + "best estimate to the goal"