The missionaries and the cannibals problem
for CSCI 4150 Intro to AI, September 17, 2001
Copyright (c) 2000, 2001 Wesley H. Huang. All rights reserved.

State representation for the missionaries and cannibals problem:
(bside lm lc rm rc) where: bside = boat side (either 'right or 'left)
lm, lc = # of missionaries and cannibals on left
rm, rc = # of missionaries and cannibals on right

Node structure:
(state parent) where: parent = this node's parent node

(define mc-start '((left 3 3 0 0) ()))

(define (mc-goal? node)
  (equal? (car node) '(right 0 0 3 3)));

(define (mc-children node)
  (let ((state (car node)))
    (map (lambda (x) (list x node))
      (mc-child-states state))))

(define (mc-child-states state)
  (if (equal? (car state) 'left)
    (map (lambda (x) (cons 'right x))
      (mc-boat-trip (cdr state)))
    (map (lambda (x) (cons 'left (switch-sides x)))
      (mc-boat-trip (switch-sides (cdr state))))))

(define (mc-boat-trip persons)
  (mc-valid-filter (list (map + '(-1 0 1 0) persons)
    (map + '(-2 0 2 0) persons)
    (map + '(0 -1 0 1) persons)
    (map + '(0 -2 0 2) persons)
    (map + '(-1 -1 1 1) persons))))
Breadth first search

Copyright (c) 2000 Wesley H. Huang. All rights reserved.

given:
- the root node of the search tree,
- a predicate that determines whether a node is the goal node
- a procedure which, given a node, returns a list of the children of
  that node

assume that a node is a list where:
- the first element is the state of the problem
- the second element is the parent node (parent of the root node is '())
- the node representation could be augmented by a third element of the
  list which describes the action used to go from the parent to the
  child node.

bfs sets up a call to the bfs-helper procedure, which will return
the goal node (or #f if the goal node is not found. bfs-list-path
then traces back to the root node of the search tree in order to
print out the sequence of states taken to reach the goal.

(define (bfs root-node at-goal? get-children)
  (newline)
  (bfs-list-path (bfs-helper (list root-node) at-goal? get-children 0)))

(define (bfs-helper node-list at-goal? get-children nodes-searched)
  (if (null? node-list)
      (begin (display "No solution found after searching ")
              (display nodes-searched)
              (display " nodes")
              (newline)
              #f)
      (let ((node (car node-list))
        ; debugging/progress information
        (if (= (modulo nodes-searched 1000) 0)
            (begin (display nodes-searched)
                    (display " nodes")
                    (newline)
                    #f))
        ; make the recursive call, appending the children to the
        ; end of the list of nodes to be explored
        (bfs-helper (append (cdr node-list) (get-children node)) at-goal?
                    get-children (+ nodes-searched 1))))))

(define (bfs-list-path node)
  (if (not (null? node))
      (begin (bfs-list-path (cadr node))
              (display (car node))
              (newline))))