1 True/False Questions on Search

1. [T/F] If all moves are unit cost, breadth-first search always finds the optimal path to the goal.

2. [T/F] Iterative-deepening search always finds the optimal path to the goal.

3. [T/F] In a graph with no loops, depth-first search always finds the goal quicker than breadth-first search.

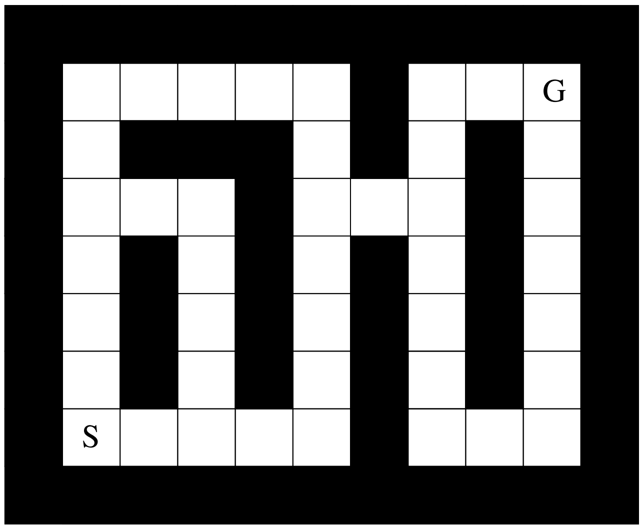
4. [T/F] All admissible heuristics are equal. That is, A* will search states in the same order for all admissible heuristics.

Solution:

1:T, 2:F, 3:F, 4:F

2 Search Trees

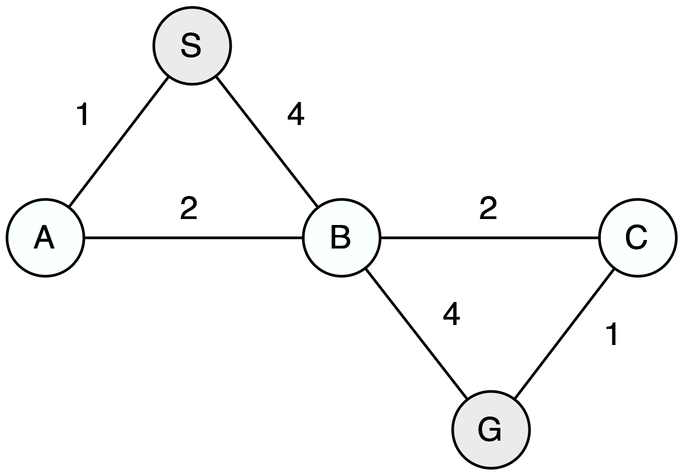
Imagine a problem with a robot trying to navigate in the following maze from the start position “S” to the goal position “G”. Draw the complete search tree associated with the problem, listing the cost of each path:



Solution:

Every junction in the maze can be represented by a node in a graph, and the cost of an edge is the distance between two junctions. This graph can then be turned into a search tree in the standard manner (slides 5.1.8-9).

3 More Searches



Consistent Heuristic

S:	5	**A:**	4
B:	3	**C:**	1
G:	0		

3.1 Best-First

Find the order that nodes are expanded in a best-first search with a visited list.

Solution:

Expanded Node	Resulting List
S	A:0+4, B:0+3
B	A:0+4, C:0+1, G:0+0
G	

3.2 Uniform Cost

Find the order that nodes are expanded in a uniform-cost search with a strict expanded list

Solution:

Expanded Node	Resulting List
S	A:1+0, B:4+0
A	B:4+0, B:3+0
B	C:5+0, G:7+0
C	G:6+0, G:7+0
G	

3.3 A*

Find the order that nodes are expanded in a A* search with a strict expanded list

Solution:

Expanded Node	Resulting List
S	A:1+4, B:4+3
A	B:4+3, B:3+3
B	C:5+1, G:7+0
C	G:6+0, G:7+0
G	

4 Code Overview

```
(define (SEARCH goal                     ; goal state
                successors               ; successor function
                pending                  ; pending list function (the Q)
                expanded                 ; expanded list function (or #f)
                visited                  ; visited list function (or #f)
                )
  (cond
   ((pending 'empty?)                            ; Failed, at least be cute...
    (display* "Cain't get thar from heah.")
    #f)
   (else
    (let ((current (pending 'next)))            ; get (and remove) node to expand

      ;; Show some status information
      (if *verbose*                      ; only in verbose mode.
          (search-node-display current "Current node: "))
      (set! *number-of-search-steps* (1+ *number-of-search-steps*))
      (cond ((= 0 (remainder *number-of-search-steps* 100))
             (pending 'summary)))
      (cond ((and expanded
                  (expanded 'member (search-node-state current)))
             (if *verbose* (display* "Already expanded."))
             (search goal successors pending expanded visited))
            ((if (procedure? goal)      ; are we there?
                 (goal (search-node-state current))
                 (equal? (search-node-state current) goal))
             (pending 'summary)         ; print a summary of pending list
             (search-node-display current " Final: ") ; display node
             (let ((path (search-node-path current)))
               (display* " Path length = " (length path))
               (display* " Path = " (map state-name path)))
             current)
            (else
             ;; pending still has entries in it and we haven't found goal
             ;; yet, so expand current node and merge results into pending
             ;; and update the expanded and visited lists.
             (search-update current (successors current) pending expanded visited)
             (search goal successors pending expanded visited))))
    )))
{
```