1. Describe the difference between “acting rationally” and “acting humanly.”

2. What are the four measures of an AI agent, abbreviated COTS, and what do they measure?

3. Describe the difference between the Greedy Search and the A* search.

4. One type of game, in the realm of AI, is deterministic, imperfect information. Name the other three and provide examples.

5. Give two characteristics about the LISP language that make it a good choice for programming AI.

6. Explain the term “relaxed problem” and why it is useful in finding an admissible heuristic for the A* search.

7. Name the four components of an intelligent agent, abbreviated by the word PEAS and briefly describe each.

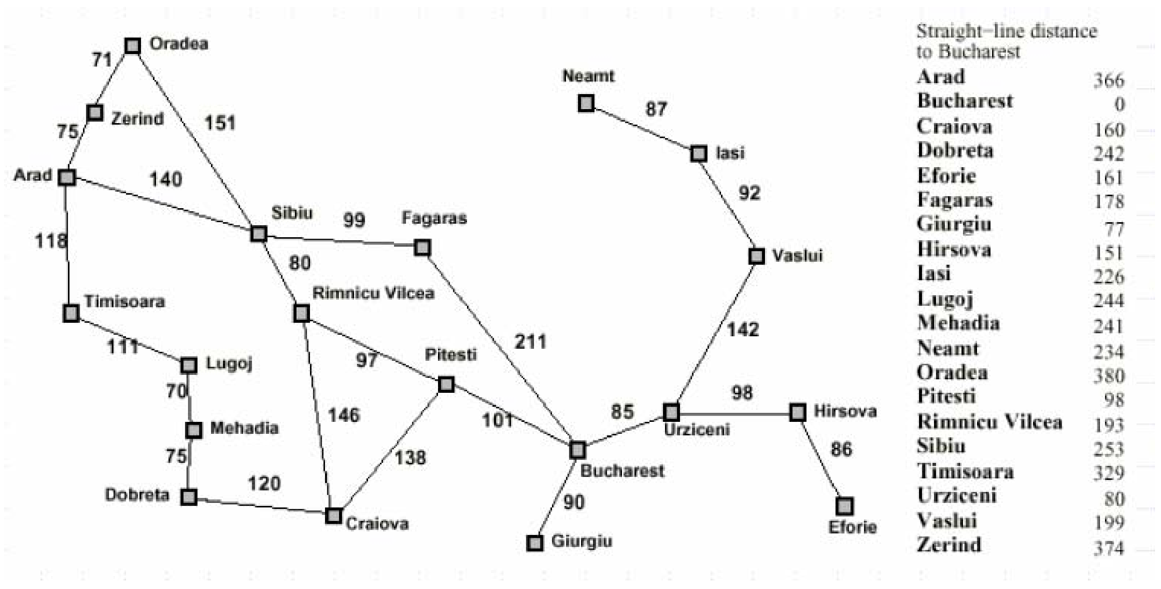
8. Match each of the approaches to artificial intelligence on the left to the field or thing that best represents it on the right.

___ Thinking Humanly	A. Turing Test
___ Thinking Rationally	B. Cognitive Science
___ Acting Humanly	C. Rational Agent
___ Acting Rationally	D. Laws of Thought

9. What do the letters of acronym PEAS stand for, in relation to describing a task environment? Choose any agent, and name at least one item that would apply to each letter of PEAS for it.

10. What properties must a heuristic have to:
    - Ensure that the optimal solution is reached in an A* search?
    - Be considered “better” than another admissible heuristic?
    - Be considered “ideal”?

11. See the map below. Using a greedy search with the straight-line distance to Bucharest as the heuristic, show the node expansion that would occur going from an initial state of Oradea to the goal state of Bucharest.



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

12. Using function notation, show the difference between the A* and greedy search.

13. Write a function in Lisp, named “tripleplusone”. It should accept one input number “x” and return the value of “3x+1”. Also, write the command line instruction to run this function for a number of your choice.

14. What will be the overall result of executing the following Lisp code? Show the value of “result” after each iteration.

```
(let ((result 3))
  (dotimes (n 4 result)
    (setq result (+ result n))))
```

15. Remembering member - Lisp is a language which uses recursion for almost everything. Show that you have a general understanding of recursion by writing a function which determines if symbol a exists in the list lst. Here is the first line of the definition, since this is a predicate function it should return T or NIL.

```
(defun member? (a lst)
```

16. Who needs Java? - Using and expanding upon the following snippets of code, make our my-box class work with arguments which can be passed at run-time and write the code to find the volume of a box with sides of 10 and with sides of 20.

```
(defclass my-box () ((length :initform 10
                             _____________ )
```

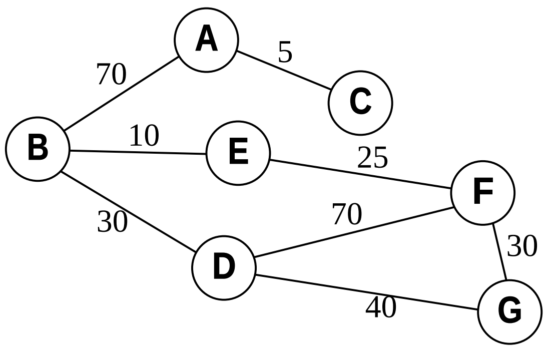
```
                         (width  :initform 10
                                 ______________ )
                         (height :initform 10
                                 ______________ )))

(defmethod volume ((self my-box))
  (* (slot-value self 'length)
     (slot-value self 'width)
     (slot-value self 'height)))
```

17. It's all C to me - Below are some pieces of simple Lisp code, briefly explain what their function is and give an example input and the corresponding output. Note: `expr` is short for expression, `length=1` checks if the list has exactly one element, and `list*` builds a list recursively.

```
(defun op (exp)
    (if (listp exp)   ; listp returns 'T if exp is a list, 'NIL
otherwise
        (first exp)
        exp))

(defun args (exp)
    (if (listp exp)
        (rest exp)
        nil))

(defun convert (expr)
  (cond ((atom expr) expr)  ; atom can be defined as (not (listp
expr))
        ((length=1 (args expr)) expr)
        (t (put-in (op expr) (mapcar #'convert (args expr))))))

(defun put-in (item list)
  (if (or (null list) (length=1 list))
      list
    (list* (first list) item (put-in item (rest list)))))
```

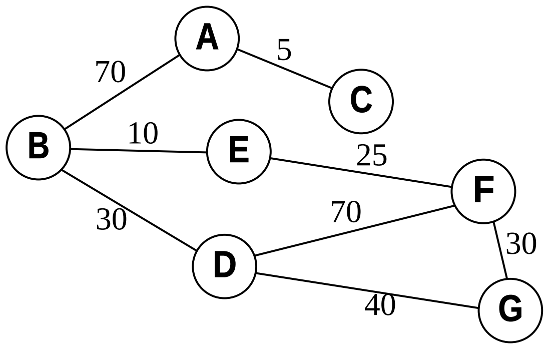
18. Defining the field - List the four main divisions of the Artificial Intelligence field. State the category under which most of AI falls and give examples of or explain two other fields.

19. Are you greedy… - Given the following diagram, distances between nodes, and using the heuristic on the right, perform a greedy search. Document each step and how you got to the next node. Write the greedy path and distance. Begin with node A and finish at the goal node G.



Heuristic h(node)	
h(A) = 100	
h(B) = 60	
h(C) = 75	
h(D) = 35	
h(E) = 50	
h(F) = 20	
h(G) = 0	

20. or are you A-star? - Using the same diagram, distances, and heuristic as before (repeated below), perform an A* search. Document each step and how you got to the next node. Make sure the answer is optimal. Write the final A* path and distance. Again, begin with node A and end at node G.



Heuristic h(node)
h(A) = 100
h(B) = 60
h(C) = 75
h(D) = 35
h(E) = 50
h(F) = 20
h(G) = 0

21. An optimal proof - In the previous problem you found (or should have found) the optimal answer by using an A* search, prove that in the general case an A* search is optimal. A formal proof is not needed, but at least give a clear and logical explanation as to why A* is optimal.

22. Find the output of the following Lisp expression.

```
 (let ((x 1) (y 2))
;; define a test function nested inside a let
statement:
(defun test (a b)
(let ((z (+ a b)))
;; define a helper function nested inside a
let/function/let:
(defun nested-function (a) (+ a a))
;; return a value for this inner let statement that
defines ‘z’:
(nested-function z)))
;; print a few blank lines, then test function ‘test’:
(format t "~%test result is ~A~%" (test x y)))
```

23. Write a Lisp expression to create the following list, sort it ascendingly, and make another list that has each element multiplied by 10.
(1 3 5 7 9 2 4 6 8)

24. Convert the following "if" expression into a "cond" expression
```
(defun our-member (elem list)
  (if (null list) nil (if (eql elem (first list)) list (our-member elem
(rest list)))))
```