

NAME: \_\_\_\_\_

login: \_\_\_\_\_

Signature: \_\_\_\_\_

Computer Science and Engineering 150  
Programming Languages for Artificial Intelligence  
Tuesday, November 5, 2002: DON'T FORGET TO VOTE!!!

FIRST MIDTERM EXAM

DO NOT TURN THIS PAGE UNTIL YOU ARE TOLD TO START!!!!

- Please DO NOT put your name at the top of each page:
  - This should prevent residual sexism, racism, favoritism lurking in the unconscious of your professor & TA from biasing grading!!
- THE EXAM IS CLOSED BOOK. IN FACT, PLEASE LEAVE YOUR BOOKS AT THE FRONT OF THE CLASS!
- Once the exam has started, SORRY, NO TALKING!!!
- No, you can't even say "later, dude!"
- There are 6 problems: Make sure you have all of them - AFTER I TELL YOU TO START!
- Read each question carefully.
- Remain calm at all times!

Problem	Type	Points	Score
1	True/False	15	
2	Fill in the blanks	17	
3	Multiple Choice	18	
4	Lisp: Threat, or menace?	10	
5	Recursive functions	20	
6	State space search	20	
	Total	100	

**True/False**

1. (15 pts: +1 for correct, -1 for incorrect, 0 for no answer (Hence guessing is dangerous!))

If you would like to justify an answer, feel free.

- \_\_\_\_\_ Uniform-cost search is guaranteed to find the cheapest path to a solution, as long as link costs are positive.
- \_\_\_\_\_ Beam search is guaranteed to find the optimal path to a solution, as long as  $k = d$ , the depth to the solution.
- \_\_\_\_\_ The result of calling `(let ((x 4)) x)` after doing `(setf x 2)` is 2.
- \_\_\_\_\_ Writing a macro such as:  
`(defmacro (x) (let (result (+ x 2)) result))`  
could cause variable capture.
- \_\_\_\_\_ Recursive best first search may change its mind several times, pruning away nodes and then searching them again.
- \_\_\_\_\_ The phrase “artificial intelligence” was coined in 1966.
- \_\_\_\_\_ Russell & Norvig (Chapter 1) make the distinction between acting rationally and thinking logically, and focuses upon thinking logically.
- \_\_\_\_\_ Chapter 2 discusses Agents that suck, among other things.
- \_\_\_\_\_ The difference between a reflex agent and a model-based agent is that the model-based agent has an internal state.
- \_\_\_\_\_ IDA\* would not work well in a situation where f-costs increase by very small increments.
- \_\_\_\_\_ If both are admissible, a heuristic  $h_1(n)$  is *more informed* than  $h_2(n)$  if  $\forall n, h_1(n) \geq h_2(n)$
- \_\_\_\_\_ Uniform cost search with all costs equal to 5 is the same as Breadth first search.
- \_\_\_\_\_ Lexical closures are a good way to make generators in lisp.
- \_\_\_\_\_ A lexical closure is a pair: A lisp expression and an environment in which that expression is evaluated.
- \_\_\_\_\_ The resource that is most often used up in search problems is space, not time.
- \_\_\_\_\_ A *documentation string* is usually a short string that is shipped with the lisp interpreter. It is used to tie around your finger to remind you to document your code.

**Fill in the blanks/short answer (17pts)**

2. (a) (2 pts) Why do most of the search algorithms stop only when a goal node is removed from the open list, instead of terminating when the goal node is first encountered (as an expansion from another node)?

(b) (1 pt) Suppose we used the heuristic estimating function  $h(n) = -g(n)$  for greedy best-first search algorithm. What kind of behavior would this produce?

(c) (2 pts) Describe the difference between planning efficiency and execution efficiency as those terms apply to state-space search algorithms. Is the A\* algorithm optimal with respect to both planning and execution efficiency?

(d) (2 pts) You know that A\* search is admissible if and only if your heuristic is admissible. You get the following bright idea: why not define a heuristic function that always returns a 1 (where 1 is the minimum cost for any move). Is this heuristic function admissible? Is this a good idea? What algorithm will A\* search behave like if we use this heuristic (Hint: It's one that we have studied).

(1 pt each blank):

(e) The value of the \_\_\_\_\_ functions is that they do not create new cons cells; however, they should be used with great caution, since they may produce unexpected \_\_\_\_\_.

2(f). Consider the following table, adapted from the draft of Chapter 3:

Criterion	Time	Space	Is it Optimal?	Is it Complete?
Breadth First	$b^{d+1}$	$b^{d+1}$	Yes	Yes
Uniform Cost	$b^{\lceil C^*/\epsilon \rceil}$	$b^{\lceil C^*/\epsilon \rceil}$	Yes	Yes
Depth First	$b^m$	$bm$	No	No
Depth-limited( $l$ )	$b^l$	$bl$	No	No
Iter. Deepening	$b^d$	$bd$	Yes	Yes
Bidirectional	$b^{d/2}$	$b^{d/2}$	Yes	Yes

$b$  = branching factor,  $d$  = depth of shallowest solution,  $m$  = max depth of search tree,  $l$  = depth limit

Please explain:

(i) (1 pt) The qualification on the claim of completeness of Iterative Deepening and Breadth First Search. (Hint: It is a property of  $d$  (not  $b$  as the Russell & Norvig book draft would have it)).

(ii) (2 pts) What is  $C^*$ ? What is  $\epsilon$ ?

(iii) (2 pt) Under what condition is Breadth First Search optimal?

(iv) (1 pt) What is the extra qualification besides the one mentioned in (i) on the completeness of Uniform Cost Search?

**Multiple Choice (18 pts, 2 pts each) (circle the *BEST* answer)**

3(a). The kinds of agents Russell & Norvig discuss are:

- (a) simple reflex agents
- (b) model-based reflex agents
- (c) logical agents
- (d) goal-based agents
- (e) search agents
- (f) utility-based agents
- (g) (a), (b), (c) and (d).
- (h) (a), (b), (d) and (f).

3(b). A search algorithm is *complete* if it

- (a) includes both a base case and a recursive case.
- (b) In the lexical closure, any free variables are gensym'ed.
- (c) Finds a solution if there is one.
- (d) Finds the shortest solution path if there is one.
- (e) (c) and (d)

3(c). A heuristic is *admissible* if it

- (a) includes both a base case and a recursive case.
- (b) Never overestimates the distance to the goal.
- (c) Finds a solution if there is one.
- (d) Is consistent.
- (e) (b) and (d)

3(d). A heuristic is *consistent* if it

- (a) includes both a base case and a recursive case.
- (b) Never overestimates the distance to the goal.
- (c) Follows a form of the triangle inequality.
- (d) Is admissible.
- (e) (b) and (d)

3(e). **A\*** and **RBFS** suffer from

- a) using too much memory
- b) using too little memory
- c) non-optimality
- d) allergies to computer pollen.

3(f). One can invent new admissible heuristics by

- (a) using pattern databases
- (b) using weighted sums of inadmissible heuristics
- (c) describing the possible actions formally, and removing conditions on actions
- (d) thinking outside of the box.

CONTINUED NEXT PAGE

3(g). Dynamic variables

- (a) have behavior that is hard to understand
- (b) get their value from the most recent binding on the stack
- (c) are useful for dynamic programming problems
- (d) (a) and (b)
- (e) (b) and (c)
- (f) (a) and (c)

3(f). The garbage collector in Lisp

- (a) may have problems recovering storage from circular lists
- (b) collects cons cells from the stack that are no longer being used.
- (c) saves the programmer from worrying about processor allocation
- (d) also recycles other users' memory when possible.
- (e) (a) and (b).

3(g). A cons cell that prints out as (a.b) is

- (a) the result of an nconc operation on lists
- (b) the result of a setq
- (c) the result of a built-in function
- (d) the result of cons'ing two atoms.

3(h) I can implement cons using append and list as follows:

- (a) (defun MY-CONS (x y) (append (list x nil) (list y nil)))
- (b) (defun MY-CONS (x y) (append (list x) y))
- (c) (defun MY-CONS (x y) (cond ((null x) y) (t (list (car x) (my-cons (cdr x) y)))))

3(i). Consider admissible heuristics  $h_1(n), \dots, h_m(n)$ . Which of the following are also admissible?

[Circle ALL that are admissible.]

- (a)  $h(n) = \sum_{i=1}^m h_i(n)$
- (b)  $h(n) = \min(h_1(n), \dots, h_m(n))$
- (c)  $h(n) = \max(h_1(n), \dots, h_m(n))$
- (d)  $h(n) = \sum_{i=1}^m w_i h_i(n)$  where  $\sum_{i=1}^m w_i = 1, w_i > 0$ .
- (e)  $h(n) = \prod_{i=1}^m w_i h_i(n)$  where  $\sum_{i=1}^m w_i = 1, w_i > 0$ .

**Do *You* have a future as a Lisp Interpreter? (10 points, 1 point each)**

4. Suppose we type the following into the LISP interpreter:

```
(setf fred 'ethel)
(setf ethel '(fred george mabel))
```

Write the result of evaluating each of the following (as if we typed them into the LISP interpreter next). [HINT: These are ALL legal!] [HINT HINT: Just remember, every "eval" simply evaluates the value of its argument]

- a. fred
- b. ethel
- c. (car ethel)
- d. (eval (car ethel))
- e. (eval (eval (car ethel)))
- f. (let ((fred 5)) (eval (car ethel)))
- g. (let ((fred 5)) (declare (special fred)) (eval (car ethel)))
- h. (let ((fred 5)) (cons fred (cdr ethel)))
- i. (let ((fred 5) (ethel 7)) (cons fred ethel))
- j. (let ((fred 5) (ethel 7)) (declare (special fred) (special ethel)) (cons fred ethel))

**Recursive functions (20 points)**

5.(14 points) This problem involves writing Lisp code recursively (what else is there?).

a. Write a function called GETMAX that takes a list of numbers as a parameter and returns the largest number in that list. You may NOT use the built in function MAX. Here are some examples:

```
USER(20) (getmax '(1 23 6 7 8))  
23
```

```
USER(21) (getmax '(45 89 213 3))  
213
```

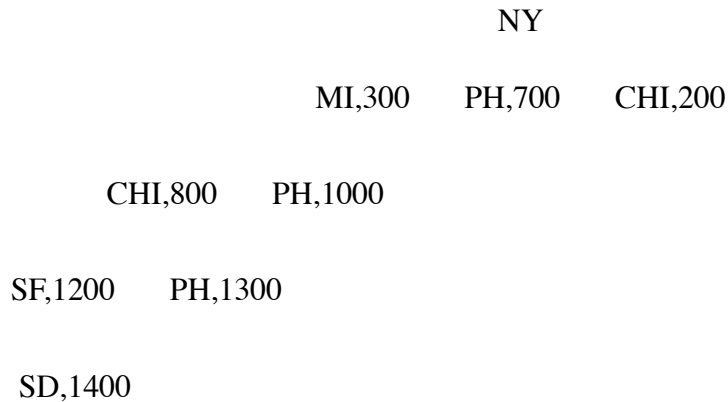
b. If you haven't already, re-write GETMAX so that it is tail recursive (HINT: Your recursive call should NOT always be on (cdr number-list)). If your GETMAX is already tail recursive, please write: "This is already tail recursive."

**State space search (20 points)**

6. Suppose your parents (who live in New York) decide to come visit you at school. They obtain the following price list for (ONE-WAY)<sup>1</sup> tickets for direct flights on their favorite airline:

From	To
New York	MIami is \$300; PHoenix is \$700; CHicago is \$200
MIami	CHicago is \$500; PHoenix is \$700.
PHoenix	San Diego is \$200; San Francisco is \$600
CHicago	San Francisco is \$400; PHoenix is \$500
San Francisco	San Diego is \$200.

Your parents aren't too good at planning, so they write a program which does a Depth First Search to find a flight path from New York to San Diego. Here is a tree showing the behavior of their program (with cumulative costs):



As you can see, this plan has them visiting five airports and costs \$1400!

(a) (6 pts) Knowing how much your parents hate layovers, you decide to take matters into your own hands and write a search program which minimizes the number of airports they have to go through. You write a Breadth first search. Draw a tree (like the one above) of your strategy's behavior. Don't forget that nodes aren't goal-tested until they come off the OPEN list! (true for *all* searches that follow).

How many airports do your parents have to visit now (COUNTING NY)? How much is the total ticket price?

---

<sup>1</sup>The important thing about the prices being ONE-WAY is that we assume the search tree is a *directed* graph – that is, you never travel BACK from where you came.

Costs from previous page for convenience:

From	To
New York	MIami is \$300; PHoenix is \$700; CHicago is \$200
MIami	CHicago is \$500; PHoenix is \$700.
PHoenix	San Diego is \$200; San Francisco is \$600
CHicago	San Francisco is \$400; PHoenix is \$500
San Francisco	San Diego is \$200.

(b) (6 pts) Your parents get your VISA bill, and realize they can't afford the trip you've planned for them. To minimize the total ticket price, you now write a Uniform Cost Search, using the actual costs of a trip on the links, and  $g(n)$  = sum of the link costs to node  $n$  (SHOW BOTH OF THESE). Finish this tree using this strategy:

NY  $g=0$

MI  $g=300$

PH  $g=700$

CHI  $g=200$

What are the dollar costs and number of airports visited now? CONTINUED NEXT PAGE

Costs from previous page for convenience:

From	To
New York	MIami is \$300; PHoenix is \$700; CHicago is \$200
MIami	CHicago is \$500; PHoenix is \$700.
PHoenix	San Diego is \$200; San Francisco is \$600
CHicago	San Francisco is \$400; PHoenix is \$500
San Francisco	San Diego is \$200.

(c) (8 pts) Now you say, "Gee, that was a lot of effort! Could I have done it with less searching?" You decide to try a heuristic search, using a coarse measure that the more westerly the city, the less cost to SD. So, you use  $f(n) = g(n) + h(n)$ , where  $g(n)$  = cost in graph so far (as in Uniform Cost Search) and  $h(n)$  given in the list below:

$$h(SF) = h(SD) = 100$$

$$h(PH) = 250$$

$$h(CHI) = 500$$

$$h(NY) = h(MI) = 1000$$

Finish the resulting search tree:

$$NY: g=0, h=1000, f=1000$$

$$MI: g=300$$

$$h=1000$$

$$f=1300$$

$$PH: g=700$$

$$h=250$$

$$f=950$$

$$CHI: g=200$$

$$h=500$$

$$f=700$$

What are the dollar costs and number of airports visited now?