6.034 Artificial Intelligence
Final Examination

Fall 2000

Name	
E-mail	
TA	
Recitation Instructor	

Problem Number	Max	Score	Grader
Problem 1	20		B D L P R U D L-P W
Problem 2	19		B D L P R U D L-P W
Problem 3	12		B D L P R U D L-P W
Problem 4	18		B D L P R U D L-P W
Problem 5	17		B D L P R U D L-P W
Problem 6	14		B D L P R U D L-P W
Total	100		

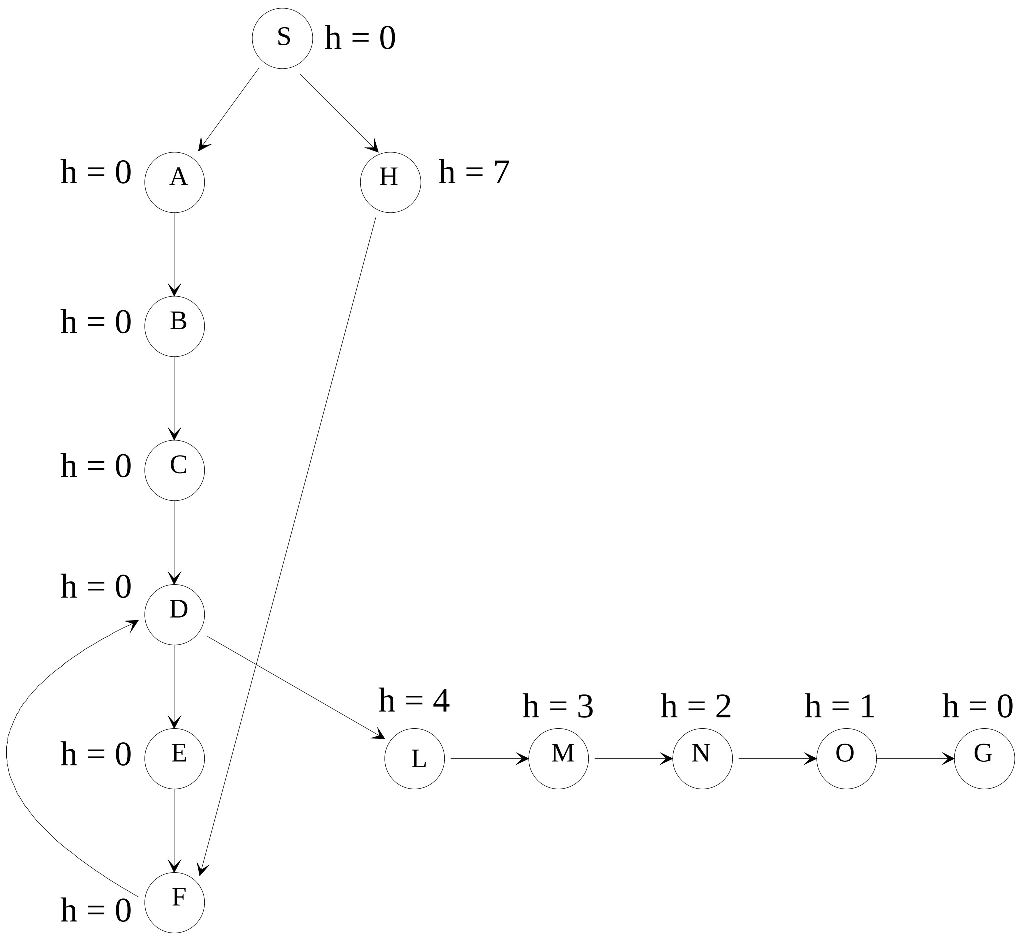
Problem 1, Search (19 points)

Part 1 (11 points)

This problem is similar to a problem given on the 1999 final, but different in important details. You are trying to find a path from the state **S** to state **G** in the following directed graph using various search algorithms.

<u>**You are NOT to use a visited list or expanded list.**</u>

<u>**The diagram is repeated on a tear-off page at the end of this examination for your convenience.**</u>



Make the following assumptions:

- All the links are of length 1.
- The heuristic distance estimates (h) from each node to the goal are shown in the figure next to each node. Each distance estimate is known to be less than or equal to the true distance.
- None of the search algorithms generate paths with loops.
- The depth-first and breadth-first search algorithms explore branches in alphabetical order.
- The search algorithms use a search queue as in the problem sets.
- **You are NOT to use a visited list or expanded list.**

Write the sequence of nodes **expanded** by the specified search methods.

Note: In the searches you are asked to perform, intermediate nodes may appear more than once in your node sequences.

Show the depth-first expansion sequence (we have started it for you):

```
S-A-B-
```

Show the A* sequence (we have started it for you). **Write above each node** in your expansion sequence **the number used by A*** to pick the next node to expand. **Hint: work out the search tree on a piece of scrap paper. Remember, you are not to use a visited or expanded list (so no pruning of multiple paths to intermediate nodes), but you are to use the heuristic estimates of distance remaining. (**This is another way of saying you are to use branch and bound with heuristic estimates of distance remaining.)

```
0 1 2
S-A-B-
```

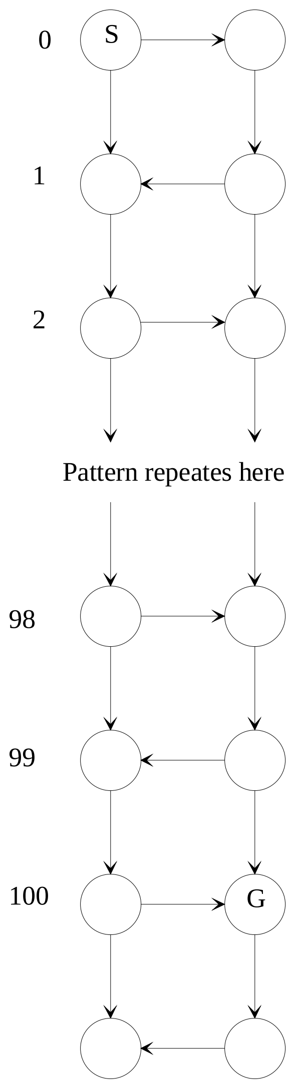
For A*, indicate the final path found…

```
S-
```

… and its length

Part 2 (8 points)

Consider the following directed graph, in which all links are of length one.



Suppose you search with the British Museum algorithm, with **no** use of any mechanism to eliminate multiple paths to the same intermediate node, how many nodes are there at the shallowest level at which the goal node**, G** is found**. <u>Hint: work out part of the search tree .</u>**

Given A* search, with dynamic programming, using a heuristic estimate of remaining path length of 0, how many paths are extended all the way to the goal node**, G.**

Problem 2, Rules (20 Points)

Note that this problem uses some of the same basic concepts as an on-line problem set, *but it is different in important ways.*

Imagine that we have a robot that moves around within a room in which there are a number of other objects. Each of these objects, and the robot, has some location characterized by a pair of numbers (x and y on some grid). The robot is also able to climb on objects. We describe the state of the world by a set of assertions of the following types:

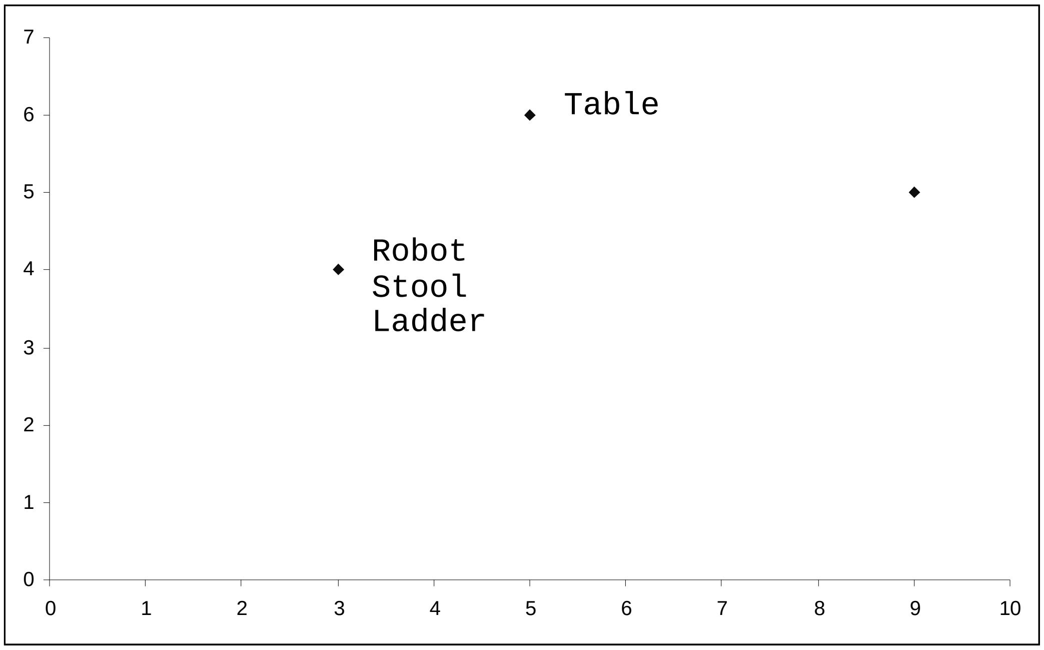
```
(robot at <location>)     ;--  this indicates where the robot is.
                          ;--  note that this is different from the
                          ;--  problem set.
(robot on <object>)       ;--  also different from the problem set.
(<object> at <location>)
(loc <location>)
```

The `(robot at <location>)` and `(robot on <object>)` assertions describe the state of the robot. The `(<object> at <location>)` assertions describe each object's location. There is a special object, `floor`, with no specified location. The existence of a location is indicated by `(loc <location>)`. There are two actions that the robot can perform:

- WALK - The robot must be on the floor as a precondition for walking and it will be at the new location afterwards.

- CLIMB - There are two cases. Climbing down from an object: the robot can do this at any time that it is not already on the floor. After the climb-down, the robot is on the floor. Climbing onto an object: the robot must be on the floor at the same location as that object. After the climb-up, the robot is on the relevant object.

Sample arrangement:



Here are some rules that we could imagine using to model this situation. *These rules are different in small but important ways from those in the online problem set, in order to take account of the representation differences pointed out above.*

```
(climb-down
 IF
 (robot on ?support)
 AND-IF (not (equal? '?support 'floor))
 SAYING "CLIMB down to floor"
 ADD    (robot on floor)
 DELETE (robot on ?support))


(walk
 IF
 (robot on floor)
 (robot at ?old-loc)
 (loc ?new-loc)
 AND-IF (not (equal? '?old-loc '?new-loc))
 SAYING "WALK to " ?new-loc
 ADD    (robot at ?new-loc)
 DELETE (robot at ?old-loc))

(end  ;-- rule just terminates chaining when the goal is accomplished
 IF
 (goal ?g)
 ?g
 SAYING "Stop"
 ADD (stop))
```

Here are some assertions for a simple situation, the one pictured on the previous page.

```
(loc (9 5))
(loc (3 4))
(loc (5 6))
(robot at (3 4))
(robot on floor)
(table at (5 6))
(stool at (3 4))
(ladder at (3 4))
(goal (on table))
```

Part 1 (10 points)

A) (6 points) Indicate the results of three cycles of forward chaining, assuming that

- you start with exactly the situation described by the assertions above,
- you have the three rules given above (`end`, `climb-down`, and `walk`),
- the system uses a forward chainer that works just like the one you have seen on problem sets, and in particular, relies on rule order for conflict resolution,
- new assertions are added at the top of the assertion list

Do this by indicating which rule instance fires on each cycle, and by listing the assertions that hold after the rule has fired. (To save you some writing, we have reproduced the initial assertion list. Edit the assertions to show what is present after each rule is fired.)

Rule instance that fires (Rule name and variable bindings)	**Assertion set *after* the rule has fired. Cross out the ones that are not present, and add in new assertions.**

	`(loc (9 5))`
	`(loc (3 4))`
	`(loc (5 6))`
	`(robot at (3 4))`
	`(robot on floor)`
	`(table at (5 6))`
	`(stool at (3 4))`
	`(ladder at (3 4))`
	`(goal (on table))`

	`(loc (9 5))`
	`(loc (3 4))`
	`(loc (5 6))`
	`(robot at (3 4))`
	`(robot on floor)`
	`(table at (5 6))`
	`(stool at (3 4))`
	`(ladder at (3 4))`
	`(goal (on table))`

	`(loc (9 5))`
	`(loc (3 4))`
	`(loc (5 6))`
	`(robot at (3 4))`
	`(robot on floor)`
	`(table at (5 6))`
	`(stool at (3 4))`
	`(ladder at (3 4))`
	`(goal (on table))`

B) (4 points) The problem set asked you to write a climb-up rule . Here we give you one and ask you to use it. The rule is:

```
(climb-up
 IF
   (robot at ?location)
   (?support at ?location)
 SAYING "CLIMB onto " ?support
 ADD     (robot on ?support)
 DELETE  (robot on floor))
```

Add this rule at the front of the rule set, and think about happens when forward chaining with the initial set of assertions (repeated below). You should discover two bugs in the climb-up rule. Fix them by writing a better version of the rule, below. (Again to save you time, we have reproduced the rule and left room for any editing you wish to do on it.)
The initial assertions:

```
(loc (9 5))
(loc (3 4))
(loc (5 6))
(robot at (3 4))
(robot on floor)
(table at (5 6))
(stool at (3 4))
(ladder at (3 4))
(goal (on table))
```

Your revised rule:

```
(climb-up

   IF

     (robot at ?location)


     (?support at ?location)



   SAYING "CLIMB onto " ?support



   ADD     (robot on ?support)



   DELETE  (robot on floor))
```

Part 2 (10 points)

A set of simple deduction rules, without variables, is given below. You are to use them to backward chain from the goal, (Z**).**

To simplify the backward chaining, assume there are initially no facts in the database and that the user replies "YES" to every question. For example, if we had only the one rule

```
IF      (P)
        (Q)
THEN    (R)
```

and backward chained from there, the system would start by asking the user "P?" to which the user would reply YES, etc.

Assume that the system tries the rules in the order shown below.

```
R1
IF      (A)
        (B)
THEN    (Z)

R2
IF      (C)
        (D)
THEN    (Z)

R3
IF      (E)
THEN    (Z)

R4
IF      (F)
THEN    (D)

R5
IF      (G)
THEN    (E)

R6
IF      (H)
THEN    (F)

R7
IF      (J)
THEN    (A)

R8
IF      (K)
THEN    (J)
```

A) (2 points) Indicate the order in which questions are asked by listing their letters below. As usual we have given you more than enough room.

1.
2.
3.
4.
5.

B) (2 points) Now draw the entire search tree, backward chaining from (Z), assuming that all questions are answered NO:

C) (2 points) Ben Bitdiddle comes along and points out that the behavior of our system could be improved. Rather than the depth-first order traditionally used for backward chaining, Ben suggests an order you might call "shortest remaining path." That order makes the system seem smarter because it tries shorter paths, i.e., shorter inference chains, before longer ones.
To implement this, Ben starts out by listing all of the paths from leaf nodes to the goal for the tree. Show that list of paths for the tree you created in Part B above (as usual you have more than enough room; we have also given you one of the paths, to indicate both what we mean and what notation to use):

1. B Z
2.
3.
4.
5.
6.
7.
8.

D) (2 points) To implement his idea, Ben then creates a simple scoring function: the score for each path is just its length, with lower scores being preferred to higher scores (ties are broken by using rule order). Indicate below the order in which questions are asked when using Ben's scoring function to determine the order in which the search tree is explored. Assume as above that all questions are answered NO.

1.
2.
3.
4.
5.

E) (2 points) Having tried out Ben's suggestion, you soon realize something. Select the single statement that best characterizes Ben's suggested revision.

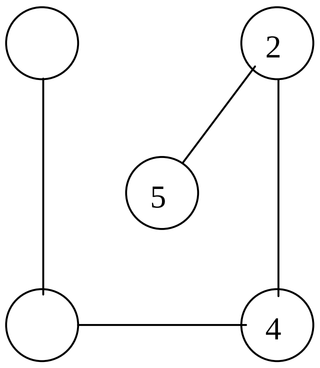
- ❑ It always asks more questions than the previous method.
- ❑ It always asks fewer questions than the previous method.
- ❑ It always asks questions in an order different from ordinary depth-first ordering.
- ❑ It never asks questions in an order different from ordinary depth-first ordering.
- ❑ It is likely to construct a larger inference tree than ordinary depth-first ordering.
- ❑ It is likely to construct a smaller inference tree than ordinary depth-first ordering.
- ❑ None of the above

Problem 3, Constraint Satisfaction (12 Points)

Note -- the Parts of this problem can be done independently.

Consider the following constraint graph. Assume each variable has the same domain D_i={A, B, C}. The only valid assignments to pairs of constrained variables are given in the table below.

Constraint (V_i-V_j)	**Valid assignments (V_i-V_j)**	
1-3	A-C	B-A
2-4	A-A	B-B
3-4	A-B	C-A
2-5	B-A	A-C



Part 1 (7 points)

A. Do constraint propagation repeated until you achieve arc consistency and show the legal domain values for each variable after the constraint propagation.

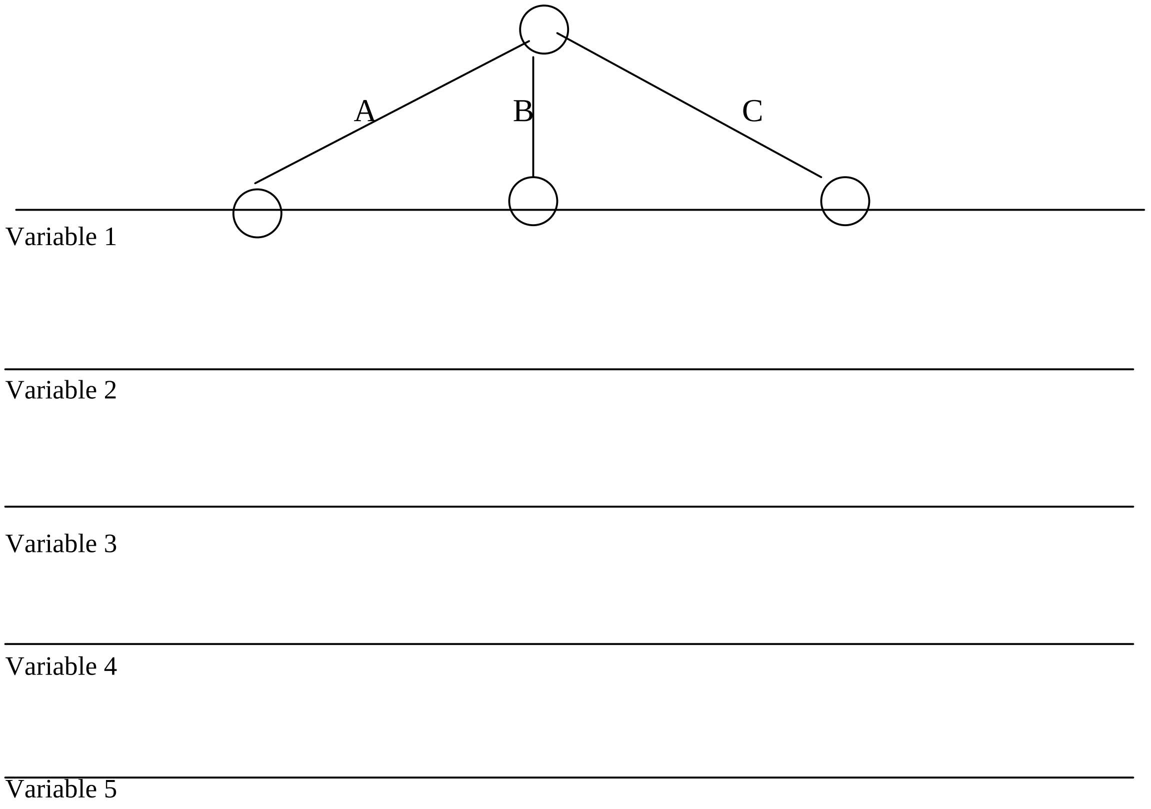
Var	**Legal Values**
1	
2	
3	
4	
5	

B. Given only the list of legal values in Part A, what can you say about the number of possible solutions to this problem. Give either a **definite** number or a **definite** range of numbers (specify both the **lower** and upper bounds of the range).

Part 2 (5 points)

In what follows, do not assume that any constraint propagation has been done. Start from scratch!

Find **one** of the valid solutions for this problem using backtracking with forward checking (BT-FC). Examine variables in numerical order and values in alphabetical order. Draw the search tree below; we have started the tree for you. For every node in the tree draw **only** the valid descendants at that point. Each horizontal line indicates the level of the tree corresponding to the assignment of values of the specified variable. **Number the nodes in the tree in the order in which assignments are considered.**

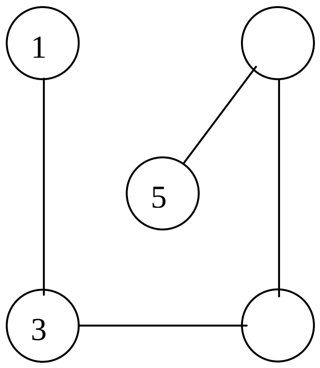


The answer found is:

1	
2	
3	
4	
5	

Constraint (V_i-V_j)	**Valid assignments (V_i-V_j)**	
1-3	A-C	B-A
2-4	A-A	B-B
3-4	A-B	C-A
2-5	B-A	A-C

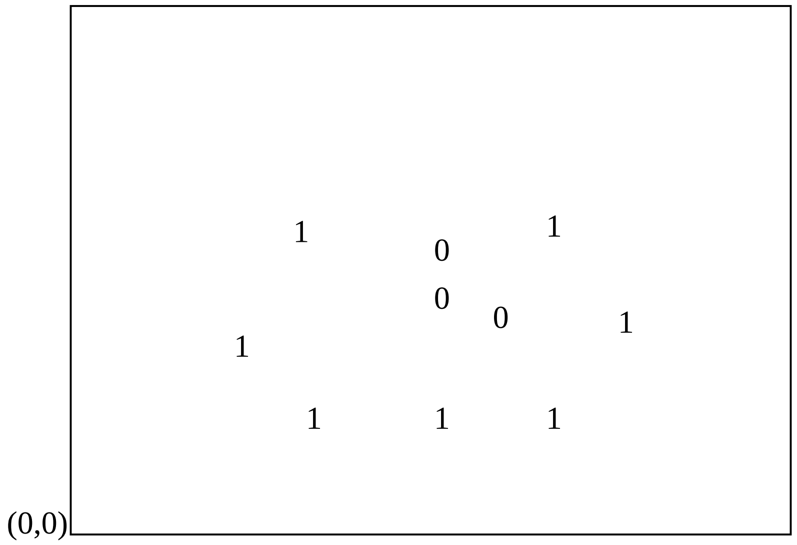
Repeated for your convenience



Problem 4, Machine Learning I (18 Points)

Note -- the Parts of this problem can be done independently.

Part 1 (4 points)

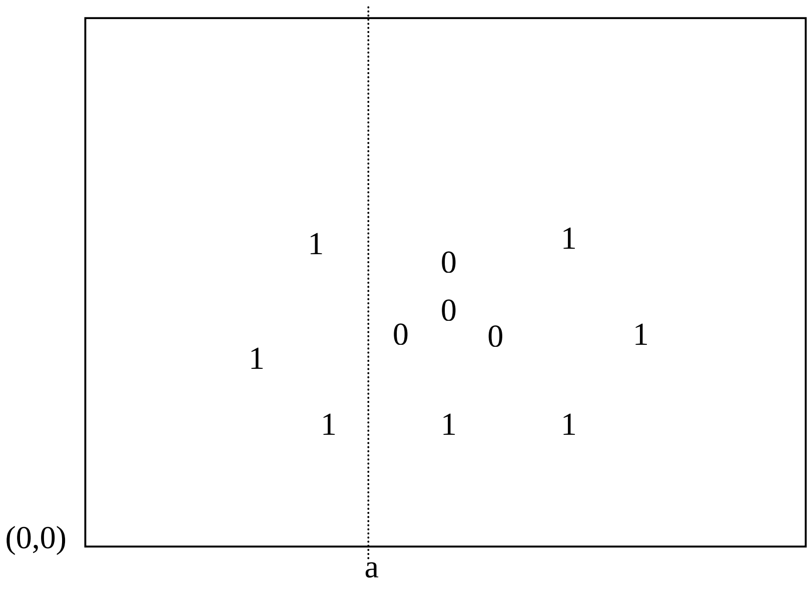


A. Suppose you decide to use the **1-nearest-neighbor** rule for identification. Pick the smallest set of points (from the set given above) that would, if used as the 1-nearest-neighbor training set, give perfect classification of all the points shown. Circle the chosen points.

B. Draw the decision boundary created if your **chosen points** are the only points in the training set.

Part 2 (6 points)

Note that this data set is not identical to the one in Part 1.

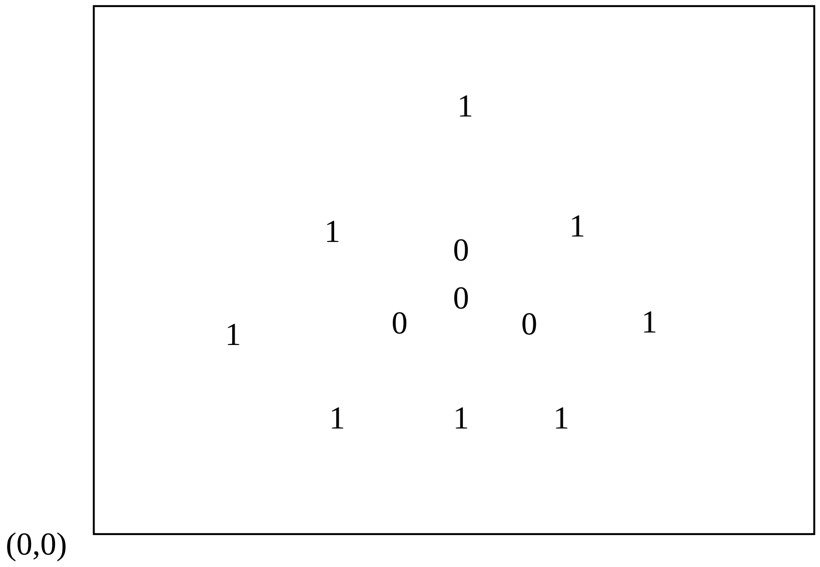


A. What is the value of the average disorder for the test shown above? You may give the result using fractions and logs, but we expect you to simplify it as much as possible (without using a calculator).

B. Draw (on the figure above) the decision boundaries for a complete ID tree that classifies the data**, starting with the test indicated in part A**. Pick further tests the way the greedy algorithm would – but don't do all the computations. Label the specific value used for each test on the figure with a letter (the way we have labeled **a** in the figure).

C. Draw the complete tree below (be sure to specify the tests, which branches are *yes* branches and which are *no* branches, and the classes that are the output). Use tests of the form X < a.

Part 3 (8 points)



A. Draw the simplest neural net architecture (neurons and connections) that should be able to classify this data set above without training error. Include the offsets/thresholds for each unit. Assume that **all units are Perceptron units**. Label each unit with a number for reference in the next problem.

B. How many weights (including offset/threshold weights) does your architecture have?

C. Draw decision boundaries on the data set figure above for each of the units in your solution to part A whose inputs are X and Y. Label each boundary with the unit number. Label which side of the boundary is supposed to be 0 and which side 1.

D. Pick one of the units whose decision boundary you drew in C and give the approximate weights for the unit. Assume that the origin of the X-Y feature space is at the lower left of the picture and the sides of the box are of size = 1. Note that, for the unit you have selected, Wx is to be the weight on the X feature, Wy is to be the weight on the Y feature and Wo is to be the weight on the offset/threshold.

Unit selected:

Weights:

W_x =

W_y=

W_o=

Problem 5, Machine Learning II (17 Points)

Part 1 (5 points)

Please answer briefly.

A. To classify the data set in Part 1 of Problem 4 using an SVM, what type of kernel should you choose?

B. Backpropagation is guaranteed to find the global minimum of the error assuming one picks a small enough step size and waits long enough. True or False?

C. Why do we initialize the weights of a sigmoid neural net to small numbers rather than large numbers?

D. What is the maximum value of the average disorder of a test for an arbitrary data set with N classes and a total of M instances?

E. How can you test whether a particular feature is likely to contribute to the classification accuracy of a 1-nearest-neighbor classifier?

Part 2 (6 points)

A. Assuming **nearest-neighbors**, describe (briefly) a situation where normalizing the data would be **harmful** to classification accuracy. Or, indicate why such a situation could not arise.

B. Assuming **id-trees**, describe (briefly) a situation where normalizing the data would be **harmful** to classification accuracy. Or, indicate why such a situation could not arise.

Part 3 (6 points)

Check any of the actions/situations that are **often** associated with **overfitting** for the indicated learning method. **You may check <u>one or more items</u> per learning method.**

A. K nearest neighbors

- ❑ Increasing the size of K
- ❑ Decreasing the size of K
- ❑ None of the above

B. ID-trees

Here, you are to assume that the ID-tree user has stopped tree growth before all samples are separated into homogeneous sets at the leaf nodes.

- ❑ Increasing the depth of the tree
- ❑ Decreasing the depth of the tree
- ❑ Increasing the number of instances at a leaf node (by decreasing depth)
- ❑ Decreasing the number of instances at a leaf node (by increasing depth)
- ❑ None of the above

C. Neural nets

- ❑ Increasing the number of units (neurons)
- ❑ Decreasing the number of units (neurons)
- ❑ Increasing the number of training cycles
- ❑ Decreasing the number of training cycles
- ❑ None of the above

D. SVM

Here, you are to study the radial basis kernel below carefully

- ❑ Reducing sigma when using the radial basis function kernel
- ❑ Increasing sigma when using the radial basis function kernel
- ❑ None of the above

$$K(\mathbf{x}_1, \mathbf{x}_2) = e^{\frac{-\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2}}$$

Problem 6, Miscellaneous (14 points)

Many of these questions are similar to questions on the second examination. None are exactly the same.

Circle the **single** phrase that **best** completes the following fragments. All multiple votes will be rejected. Each question is worth one point. No penalty for guessing.

Progressive deepening, also known as iterative deepening, works well for games because:

- ❑ Alpha-beta allows you to go approximately twice as deep in a typical game tree.
- ❑ The branching factor varies from layer to layer.
- ❑ Almost none of the nodes in a game tree of a given depth are in the final layer.
- ❑ The number of nodes in a layer increases exponentially with depth.
- ❑ All of the above.
- ❑ None of the above.

Alpha-beta:

- ❑ Doubles the speed of minimax.
- ❑ Halves the speed of minimax.
- ❑ Ensures that there will be an answer when time runs out.
- ❑ Takes minimax approximately twice as deep in a typical game tree.
- ❑ All of the above.
- ❑ None of the above.

A frame system may be useful because it provides:

- ❑ Property inheritance.
- ❑ Method inheritance.
- ❑ Domain-specific slot vocabularies.
- ❑ Domain-specific slot values.
- ❑ All of the above.
- ❑ None of the above.

A key virtue of the transition-space representation is that it:

- ❑ Focuses on change.
- ❑ Can be translated to relational-database records.
- ❑ Provides a foundation for semantic transition-tree grammars.
- ❑ Ensures that all descriptions will satisfy Newtonian mechanics.
- ❑ All of the above.
- ❑ None of the above.

Recognition by Ullman's alignment method (predicting point locations) requires:

- ❑ An ability to find corresponding points or sets of points.
- ❑ Rigid objects (i.e. nothing floppy, such as an article of clothing)
- ❑ Libraries containing multiple images of each object to be recognized
- ❑ A mechanism for solving linear equations
- ❑ All of the above
- ❑ None of the above

Four points in three images (the model) are sufficient to predict the location of points in a fourth image unless:

- ❑ Not all the points are visible in all the model images
- ❑ The points in the model images are not in proper correspondence
- ❑ The differences among the three model images are produced by single-axis rotation only
- ❑ The differences among the three model images are produced by translation along one axis only
- ❑ All of the above
- ❑ None of the above

The Yip-Sussman method for learning phonological rules:

- ❑ Generalizes seed patterns
- ❑ Specialize seed patterns
- ❑ Specializes and generalizes seed patterns.
- ❑ Uses A* search to find minimal patterns
- ❑ All of the above
- ❑ None of the above

One reason for the success of the Yip-Sussman method is that:

- ❑ Thousands of examples are provided
- ❑ The rules learned are bidirectional
- ❑ Sparse high-dimensional spaces are involved
- ❑ Binary spaces are involved
- ❑ All of the above
- ❑ None of the above

A major reason for the success of near-miss learning from samples is that:

- ❑ Thousands of examples are provided
- ❑ The teacher ignores the student's state of knowledge
- ❑ The student is only required to specialize seed examples
- ❑ The student generates its own near miss samples
- ❑ All of the above
- ❑ None of the above

In near-miss learning

- ❑ Near misses (not examples), cause generalization (not specialization)
- ❑ The forbid-link heuristic is triggered by examples, not near misses
- ❑ The drop-link heuristic is triggered by examples, not near misses
- ❑ The require-link heuristic is triggered by examples, not near misses
- ❑ All of the above
- ❑ None of the above

Learning by explanation:

- ❑ Generalizes from specific situations
- ❑ Speeds subsequent reasoning
- ❑ Can affect what is concluded in practice
- ❑ Depends fundamentally on appropriate representation
- ❑ All of the above
- ❑ None of the above

Good representations:

- ❑ Make the right details explicit
- ❑ Suppress irrelevant details
- ❑ Expose constraint
- ❑ Bring helpful descriptive details together
- ❑ All of the above
- ❑ None of the above

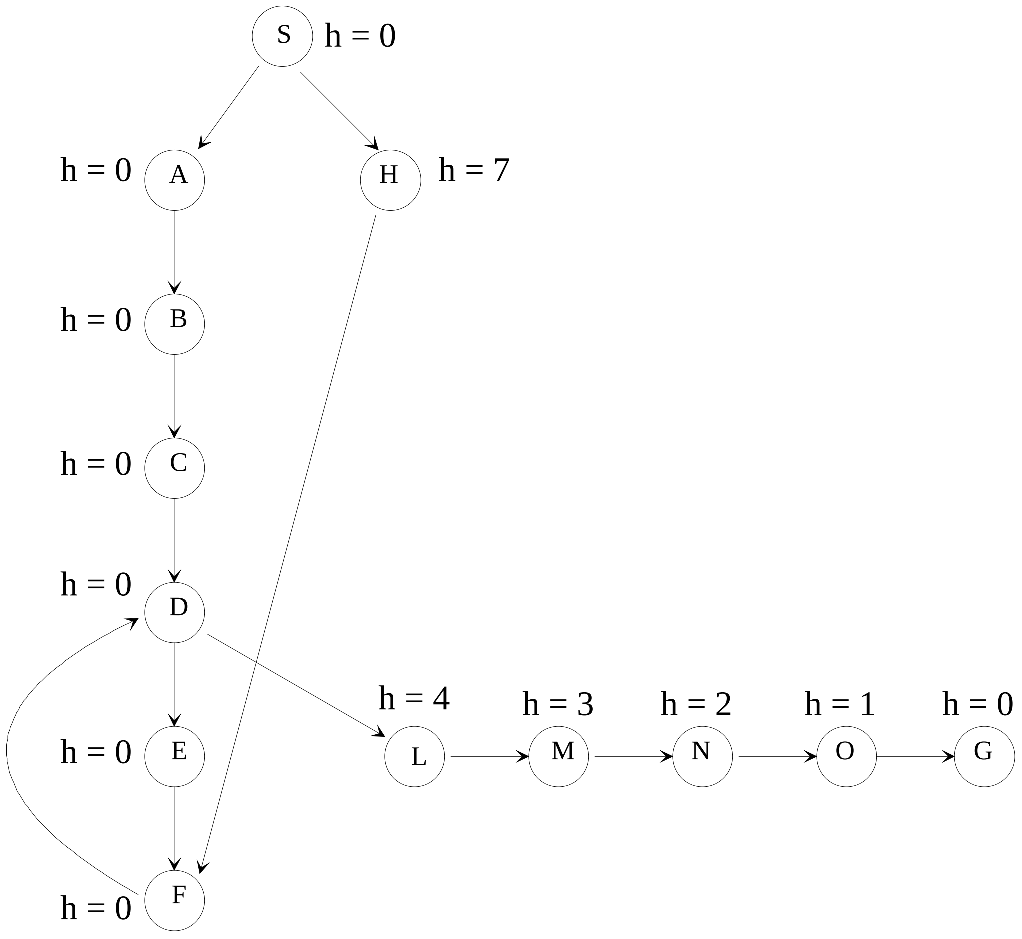
The hippocampus of a rat:

- ❑ Seems to enable a kind of practice during sleep
- ❑ Seems to have something to do with forming long-term memories
- ❑ Seems to have neurons that are responsive to the rat’s position
- ❑ Seems to support mechanisms that exhibit a kind of generalization
- ❑ All of the above
- ❑ None of the above

If you lost your hippocampus in high school, you would be:

- ❑ Unable to pass a 6.034 final
- ❑ Unable to recognize old friends
- ❑ Unable to walk
- ❑ Unable to talk
- ❑ All of the above
- ❑ None of the above

Diagram repeated for your convenience.


S
h = 0
A
h = 0
H
h = 7
B
h = 0
C
h = 0
D
h = 0
E
h = 0
F
h = 0
L
h = 4
M
h = 3
N
h = 2
O
h = 1
G
h = 0