Name:

Examination # **1** Date: **10/16/97** 6.034 Item # **17**

Please indicate the section you were assigned to, or if you do not remember, put a tick mark on the last line.

Time	Room	Instructor	TA	Tick
10–11	34-302	Randall Davis	Brad Banks	
10–11	34-303	Tomás Lozano-Pérez	Lisa Tucker-Kellogg	
11–12	34-302	Randall Davis	Brad Banks	
11–12	36-156	Tomás Lozano-Pérez	Lisa Tucker-Kellogg	
12–1	34-301	Paul Viola	Constantine Papageorgiou	
12–1	34-302	Robert Berwick	Adrian Danieli	
1–2	24-407	Paul Viola	Constantine Papageorgiou	
1–2	34-303	Robert Berwick	Adrian Danieli	
2–3	34-303	David Cliff	Oded Maron	
3–4	34-303	David Cliff	Oded Maron	
—	—	—		

Problem 1	
Problem 2	
Problem 3	
Problem 4	
Total	

Problem 1 Search (25 points)

You are driving a cab in a weird, unfamiliar city with streets laid out as shown by the lines in the figure on the following page. Black dots indicate intersections ("the nodes" from the search perspective). The winding line leaving intesection J is one street with no intersections occuring before the next black dot. Two lines that cross at a place where there is no black dot are not connected at that place.

Lines with arrows are one-way streets in the indicated direction.

At intersection S, a customer says he wants to go to intersection G.

Important special notes:

- The map is drawn to scale.
- When you are asked to use a method that involves a heuristic measure of "goodness," you are to use straight-line, as-the-crow-flies, distance-to-goal as that heuristic measure.
- For all searches in which branch order is not defined by a measure of "goodness," assume that the search moves down the branches in left-to-right order, if there is a left-to-right order, and bottom-to-top order, otherwise
- The dynamic programming principle says that if you find two paths to the same place, you throw away the longer of the two, or in case of a tie, you throw away the most recently discovered path.

Extremely important special note:

- The word **generate**, as used in this problem, means that the path is constructed in the course of your work (but not necessarily placed on the search queue).

Part A (5 points)

You decide to look for a route using depth-first search, with backtracking, but without use of the dynamic programming principle. That is, you keep all non-circular paths on the search queue, even if there are multiple paths that get to the same place.

You generate (see definition above) paths terminating at place I in the search tree before you find a path from S to G.

Part B (5 points)

Later, over coffee, another driver, also MIT educated, says you blundered; you should have used hill climbing, with backtracking, but without use of the dynamic programming principle. If you had, you would have generated paths terminating at place I in the search tree before you find a path from S to G.

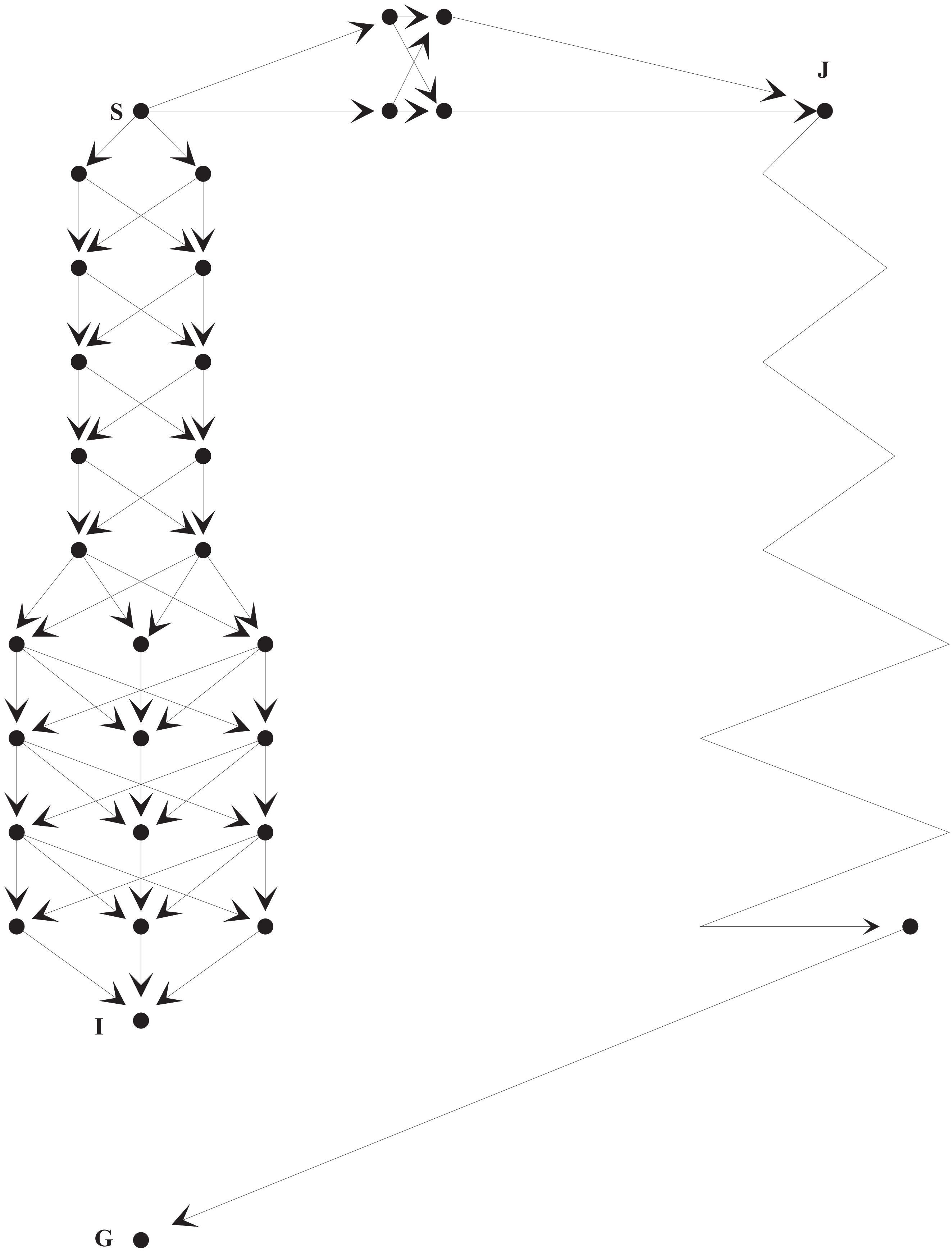


Figure 1: A weird city.

Part C (5 points)

Your boss says you should have used depth-first search, with backtracking, and with the use of the dynamic programming principle. If you had, you would have generated $\boxed{}$ paths terminating at place I in the search tree before you find a path from S to G.

Part D (5 points)

Your boss's boss says you should have used beam search, with a beam width of two, without the use of the dynamic programming principle, and without any sort of backtracking. If you had used your boss's boss's suggestion, you would have generated $\boxed{}$ paths terminating at place I. Would you eventually find a path from S to G? $\boxed{}$

Part E (5 points)

Your passenger says he wants an optimal path, so you decide to use A* search (which includes, by definition, the application of the dynamic programming principle). Use 0 as the lower bound estimate on distance remaining from any place to the goal (equivalent to not using a lower bound estimate). You will generate $\boxed{}$ paths terminating at place I in the search tree before you find the optimal path from S to G. You will also generate $\boxed{}$ paths from S to J before you find the optimal path from S to G.

Problem 2 Games (25 points)

We have just received word that Kasparov, dejected after losing to Deep Blue, has been in seclusion, training his protege and successor, Boris Notquitegoodenough. You have been assigned the important task of looking for ways to improve Deep Blue's performance, to produce Very Deep Blue.

In the original match, Deep Blue chose the move but a person moved the pieces on the board. This time in order to really rub in the machine character of the program, we've added a robot arm to move the pieces.

One problem is that it's a battery-operated arm and hence we have to be careful not to use up all of its power before the game ends. We can save power by moving light pieces (pawns, rooks, knight) in preference to heavy pieces (king, queen, bishop) when we have a choice.

The Very Deep Blue programming team needs your help. In order to show Notquitegoodenough who's boss, we want you to help them improve min-max search.

Part A (7 points)

To help you get started, we'll have you do a slight variation on min-max search on the tree shown in Figure 2.1; the variation is explained below. Be sure to read to the end of Part A before beginning work on this part of the problem.

Next to each non-leaf node in Figure 2.1 is a box. When you're doing the search, put into each box the score that min-max assigns to that node, and the name(s) of the leaf/leaves corresponding to that score.

A hypothetical entry in the box would be of the form `7 A18`, indicating that the node next to the box received a score of 7, and that score came from leaf node A18.

Note that when you are done, the root node will have a score and the names of all the leaf nodes corresponding to the best score. This differs from standard min-max search, where only one of the best-scoring leaves is remembered.

Hence your assignment in this part of the problem is to do a variation on min-max search on the tree shown, keeping careful record of the steps by annotating the tree, and producing at the root node the best score and all of the leaves that cause that score.

Recall that by convention, Very Deep Blue goes first and tries to maximize the score; Notquitegoodenough goes second and tries to minimize the score.

Part B (8 points)

Min-max normally finds the best move. We not only want Deep Blue to find the best move, we want it to find the least-cost best move, i.e., of the best moves, we want the one that requires moving the lightest pieces. Hence if two or more nodes are tied for the best move according to the static score, we want to find the one that has the least cost path.

Figure 2.2 shows the same tree as before, only now associated with each move by Very Deep Blue we can see a cost figure (we've left out the static evaluations at the leaves to avoid cluttering the figure). Moves by Notquitegoodenough don't cost anything. Note that the number next to each branch is the cost for that single move.

Imagine you were to do a branch and bound search on the tree so that you can choose a move that takes you along the path that is the least costly path that leads to one of the tied leaf nodes selected in part A. The branch and bound search terminates, of course, when the least-cost path to one of the tied leaf nodes has been determined.

At the time of search termination, the branch and bound algorithm's queue may contain many paths or partial paths. For each path in the queue at the end of the search, if any, circle the node in the tree that corresponds to the last node in that path.

Part C (4 points)

You get to thinking and decide that it seems wasteful to do this in two phases; why not combine them? Suppose we define a modified static score that includes the path cost:

Modified Static Score = Original Static Score - Path Cost

That is, the modified static score is the original static score reduced by the sum of the weight-costs of the moves to get to that node. (Note that the modified static score can go negative, but we still want to find the algebraically maximal score.)

The idea is that by combining the two scores we combine the two processes and do less work.

Do a max-min search on the fragment of the tree shown in Figure 2.3, using the Modified Static Score defined above. (You can do a standard min-max search, and not the variation described in part A.)

In the boxes next to each node in Figure 2.3, fill in the modified static score.

Part D (6 points)

You should have found in Part C that the modified score won't necessarily cause Very Deep Blue to choose the same node as the "wasteful" two-phase algorithm that uses the original static scoring and then breaks ties using weight cost.

Goal: We want to construct a Very Modified Static Score that will have the same effect as using the original static score and then breaking ties using weight cost.

Assume the heaviest piece has a weight of 50, there are at most 50 moves per game, and the static scores are integers.

Which expressions below could replace X and Y in the definition below, and still meet our goal?

Very Modified Static Score = $X \cdot$Original Static Score - $Y \cdot$ Path Cost.

Clearly circle any/all that apply.

1. $X = 1$ and $Y = 0$

2. $X = 0$ and $Y = 1$

3. $X = 10$ and $Y = 3.1$

4. $X = 1,000,000$ and $Y = \pi$

5. $X = \pi$ and $Y = 1,000,000$

6. There are no possible values for X and Y.

Problem 3 BC Rules! (25 points)

Consider the rules below, which encode a small set of differentiation facts. The

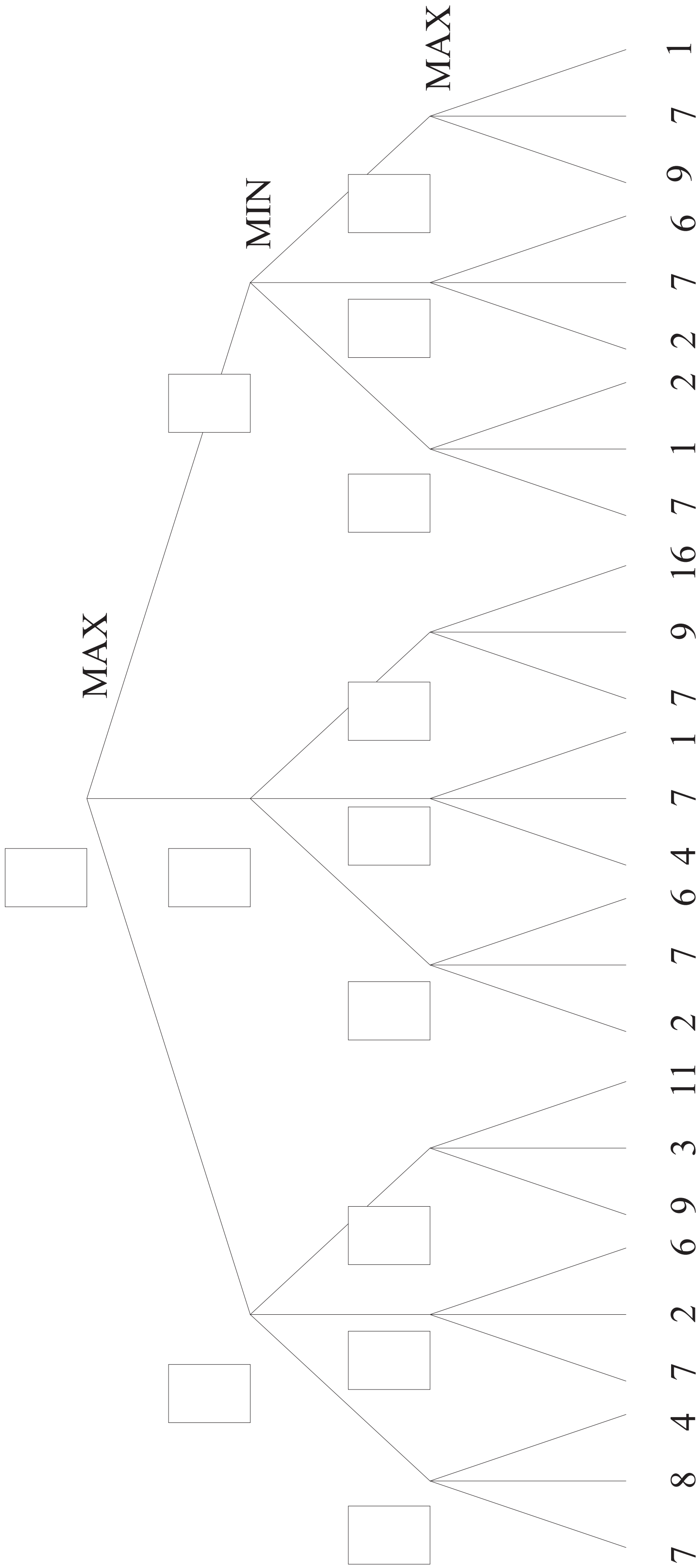
```
(deriv a b)
```

assertions are intended to encode that `b` is the derivative of `a` with respect to variable `x`. That is,

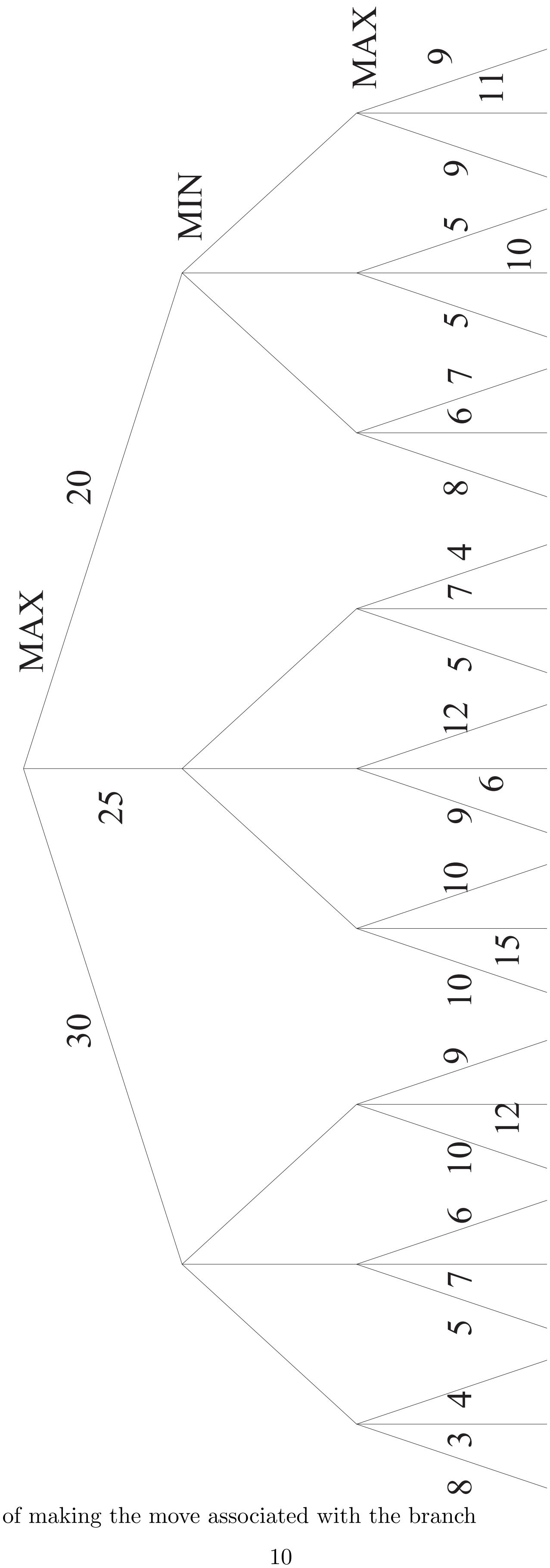
$$b = da/dx$$

Note that we will ONLY differentiate with respect to the `x`, so the input has to be expressed in terms of `x`.

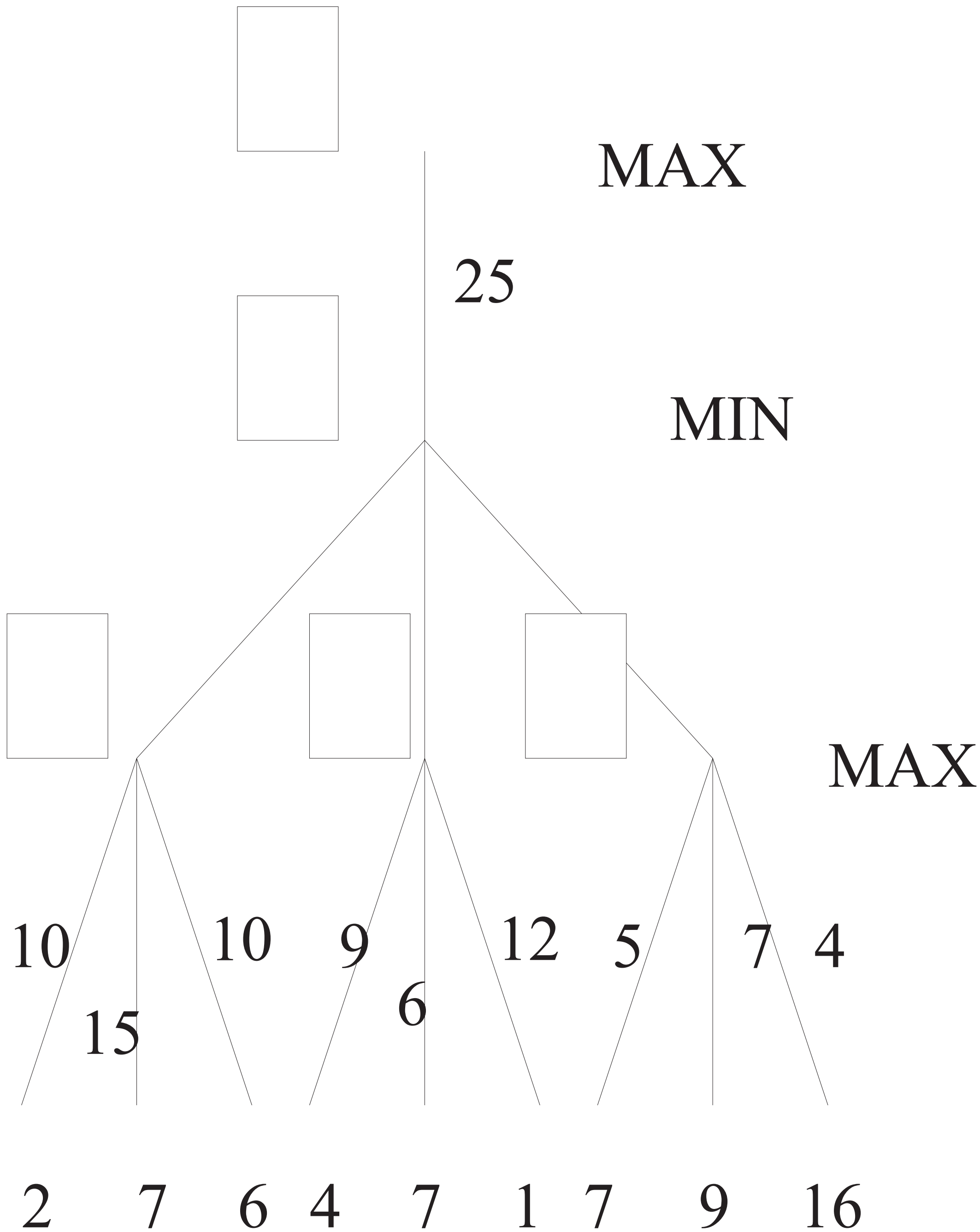
```
R1  IF   (constant ?x1)
    THEN (deriv ?x1 0)

R2  IF   (deriv ?x2 ?dx2)
         (deriv ?y2 ?dy2)
    THEN (deriv (+ ?x2 ?y2)  (+ ?dx2 ?dy2))

R3  IF   (deriv ?x3 ?dx3)
         (deriv ?y3 ?dy3)
    THEN (deriv (- ?x3 ?y3)  (- ?dx3 ?dy3))

R4  IF   (deriv ?x4 ?dx4)
         (deriv ?y4 ?dy4)
    THEN (deriv (* ?x4 ?y4)
                (+ (* ?x4 ?dy4) (* ?dx4 ?y4)))

R5  IF   (deriv ?x5 ?dx5)
         (deriv ?y5 ?dy5)
    THEN (deriv (/ ?x5 ?y5)
                 (- (/ ?dx5 ?y5) (/ ?x5 (expt ?y5 2))))

R6  IF   (deriv ?x6 ?dx6)
    THEN (deriv (- ?x6)  (- ?dx6))

R7  IF   (deriv ?x7 ?dx7)
         (constant ?n7)         ; the exponent must be a constant.
    THEN (deriv (expt ?x7 ?n7)
                (* ?n7 (expt ?x7 (- ?n7 1)) ?dx7))
```



2.1 Numbers are static evaluation scors for the node



2.2 Numbers are the cost of making the move associated with the branch



2.3 The middle segment of the same tree, showing both static scores and move costs

This page intentionally blank

IMPORTANT: We assume that we use the UNMODIFIED back-chaining system as handed to you in Problem Set 3, in particular, the system knows NOTHING about arithmetic – everything is done by pattern matching. We repeat, in these rules the arithmetic operations such as + and − are only symbols, we could have used foo and bar instead.

You are given ONLY rules, there are no assertions provided.

Part A (5 points)

We want to use these rules to find $\frac{d}{dx}x^2$. What input do we give the backchainer to find the answer to this problem? The expression itself is written as `(expt x 2)`. Write your answer below.

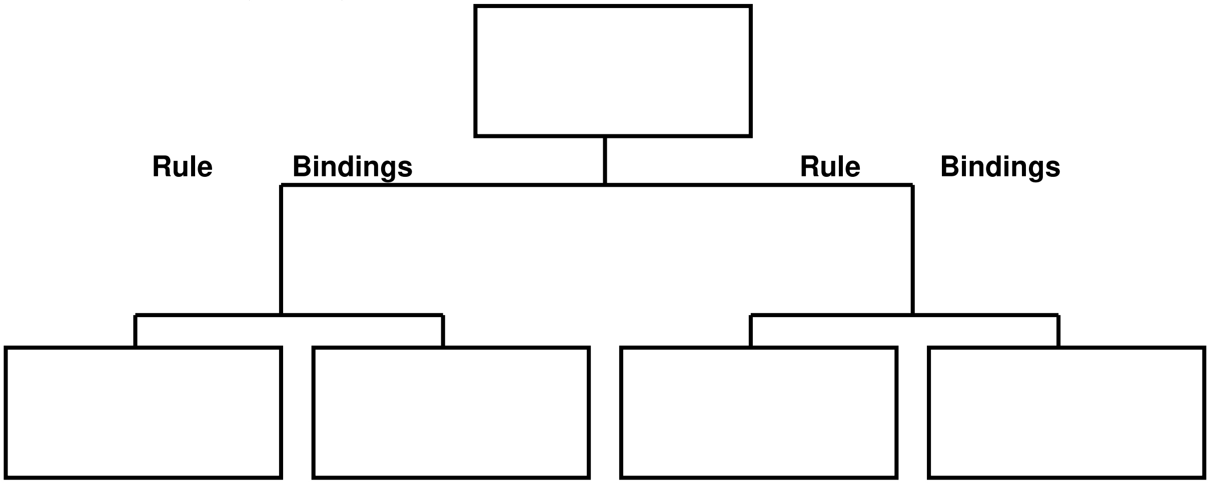
```
(back-chain                                                )
```

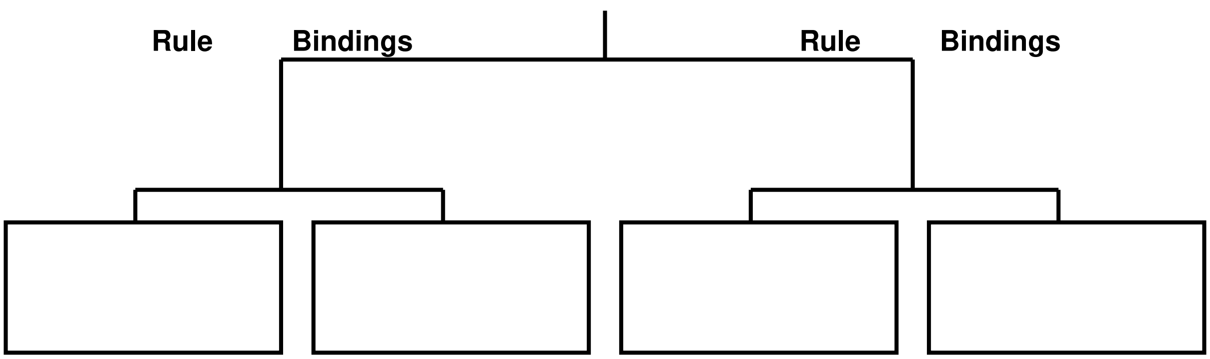
Part B (10 points)

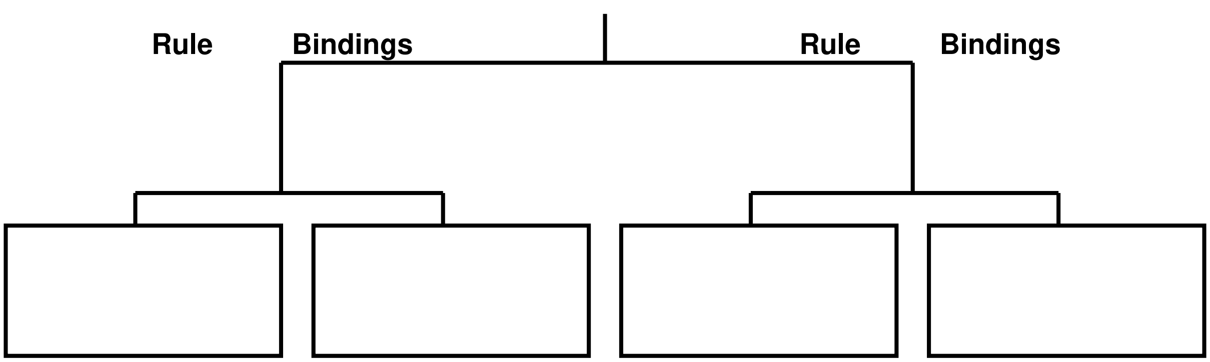
Write the tree of rules that are used when one starts backchaining on the problem in Part A. On the next page, we have provided a template with (more than) enough slots to write your answer. Be sure to indicate ALL rules that are found relevant to establishing a premise. Also indicate the bindings of the variables required for the match. When you write premises (patterns) in the tree, show the premise with all appropriate variable substitutions. Also, indicate any questions asked of the user. Order the rules from left to right in the order they would be explored. The typical entry in your tree should look like:

```
                 pattern
                    |
            |------------------|
         R1 | ?x <- foo     R2 | ?y <- bar
            |                  | ?z <- ?w
     |-------|                 |
   pattern pattern          pattern
                           (Question)
```

NOTE: You don't need to fill in all the slots! Also, don't be surprised if you find that you cannot compute the correct derivative with the rules provided (see Part C). Fill in your tree starting with the top template. If you need additional levels, use the two bottom templates as needed, draw in (clearly) the connection of any extra levels to the top tree.







Part C (5 points)

You should have found that you could not actually compute a complete answer to the problem in Part A using the rules provided. Please give any general rules and/or assertions that are necessary to construct a valid answer. We want only GENERAL rules and/or assertions, not ones specific to the problem in Part A.

2in

Part D (5 points)

Is there any change we can make to the rules (beyond the one you made in Part C) to avoid asking the user ANY questions, when finding derivatives covered by the rules given above? We are asking for a general solution, not one specific to the particular problem in parts A and B. If yes, explain how. If not, explain why not.

2in

This page intentionally blank

Problem 4 Puzzling CSP's (25 points)

Imagine a young friend receives a strange puzzle as a birthday present. The puzzle consists of a large playing board to frame the puzzle, and some (in this example, four) square tiles with shape patterns on them. The shape patterns are halves of squares, diamonds, or circles. The tiles have to be arranged in the frame so that their shape patterns form complete squares, diamonds, and circles. Since each tile can be rotated (but not flipped), the placement of any single tile is locally very ambiguous. A (very simple) example of such a puzzle is shown below.

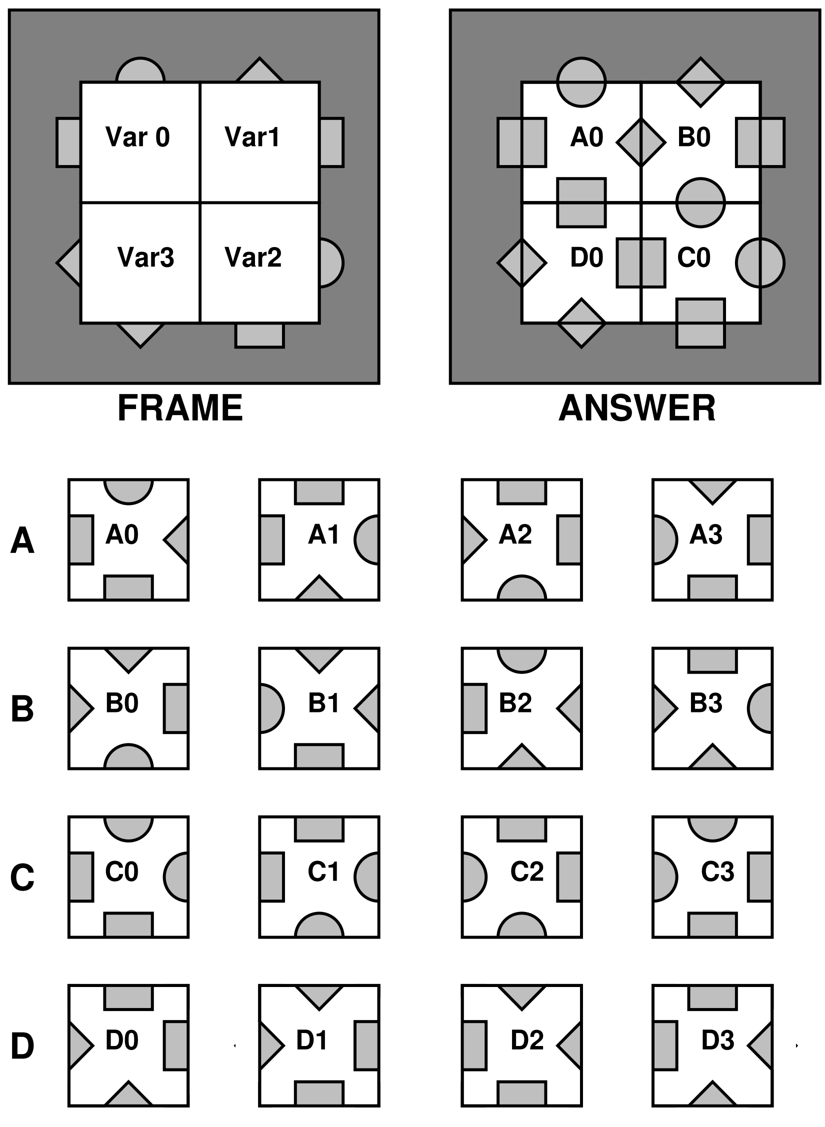


Figure 2: The top pictures show the puzzle frame and the answer with the four pieces (in their correct orientations – A0, B0, C0, D0) positioned. Below are the four pieces (A,B,C,D) in each of their four valid orientations (A0,A1, ..., D3). No other orientations are valid.

This page intentionally blank

Your friend asks for help with the puzzle. Instead of trying to do it manually, you figure that this is a perfect example of a CSP. So you decide to use the 6.034 CSP solver to find the answer[1].

Part A (5 points)

Assume we use the positions in the puzzle as the variables in this problem (keeping the numerical order shown above). For each variable, give the smallest possible domain of legal values. You should exploit any available information local to a single variable (but without doing propagation) to reduce the size of the domains. We have already filled in a couple of domains.

Variable	Legal Domain Values
0	A0 B2 C0
1	
2	
3	B3 D0

Part B (5 points)

Describe succintly what ALL the TYPES of constraints are for this problem. For example, in map coloring one has only one TYPE of constraint, namely, that adjacent regions do not have the same color. Write a number next to each constraint type.

2in

[1] By the way, this actually happened to one of the instructors whose daughter asked him to help her solve her puzzle. Now, his daughter believes that he actually does something useful at work...

Part C (5 points)

Draw a constraint graph for this problem, with a node for each variable and a line connecting pairs of variables related by a constraint. Label each line with the number corresponding to the constraint type (from Part B).

3in

Part D (5 points)

Starting with the domains in Part A, show the domain of the variables after calling `revise` once on the arc whose tail is var 0 and whose head is var 3.

Variable	Legal Domain Values
0	
3	

Part E (5 points)

Assume we run `(backtrack-fc-dynamic pick-most-constrained domain-of)` from the initial state of the problem (the domains in Part A). If the most-constrained variable is ambiguous, choose the lowest-numbered variable among those tied for first. We take a step in the search to be defined by all the work done between calls to `backtrack-fc-dynamic-aux`. At each step in the search, show (in the table provided below) which variable is explored, the value assigned to it and show the active domains for all the variables at the END of the step. The number of columns below is not intended to suggest that you need exactly that many steps.

	search step 1	search step 2	search step 3	search step 4
Chosen Var				
Chosen Value				
Var 0				
Var 1				
Var 2				
Var 3				