Name:

Examination # **1** Date: **10/15/97**

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

Time	Instructor	
10-11	Davis	
11-12	Davis	
10-11	Szolovits	
11-12	Szolovits	
1-2	Shrobe	
1-2	Berwick	
2-3	Berwick	
2-3	Lozano-Perez	
3-4	Lozano-Perez	

Please indicate the TA you are assigned to.

Calista Tait	
Dennis Ruhl	
Irene Wilson	
Jonathan Loflin	
Ozlem Uzuner	
Serkan Arslanalp	

Problem 1	
Problem 2	
Problem 3	
Problem 4	
Total	

Problem 1 Forward Chaining (25 points)

This problem has two unrelated sections; one on genealogical data and one on robots. Throughout the problem assume (unless told otherwise) a forward-chaining system exactly as given to you in PS 2, that is, firing the first triggered rule instance and adding new assertions to the front of the list of assertions.

Part A (5 points)

You are given a large genealogical database made up of millions of assertions like the following:

```
(A is-parent-of C)
(B is-parent-of C)
(A is-of-gender Male)
(B is-of-gender Female)
(C is-of-gender Female)
```

There are no other types of assertions in the database, only `is-parent-of` and `is-of-gender` assertions.

Write an IF-THEN rule which, given this database, adds assertions of the form `(D is-uncle-of E)`. Do not make distinctions between siblings who share two parents or just one parent. Please give your variables mnemonic names. Hint: to force two variables `(? x)` and `(? y)` to match different people you can add a clause to the rule like this:

```
AND-IF (not (eq? (? x) (? y)))
```

Part B (4 points)

Assume that you are using a forward chaining rule-based system that uses the Rete algorithm (as described in the text). When you load the database, you see some uncle assertions being added and then the time between assertions grows and grows. Why is this happening? Explain very briefly. Be sure that you indicate which step(s) in the description of the Rete algorithm (page 160 of the text) is (are) primarily responsible for the slowdown.

Part C (3 points)

Now, returning to the original database, without uncles, if you are intersted in finding who your maternal grandmother's uncles were using this database, should you use a forward chaining system

(run `(chain)` as in PS2)

or a backward chaining one

(run `(backchain '((?  x) is-uncle-of Granny))` as in PS3)?

Explain very briefly.

Part D (5 points)

Consider a robot that moves around in the following environment:

```
---------------------------------
| Room 5  |  Room 2  |  Room 3  |
|                               |
|         |          |          |
---   -------   ----------   ----
| Room 6  |  Room 1  |  Room 4  |
|                    |          |
|         |          |          |
---------------------------------
```

The connectivity between these rooms is described by a set of assertions for a rule-based system indicating what rooms are connected by doors:

```
(door r1 r2)
(door r2 r5)
(door r2 r3)
(door r4 r3)
(door r5 r6)
(door r6 r1)
```

In addition, there is an assertion indicating the current position of the robot.

```
(loc r1)
```

You are given the following rules:

```
(GO   if (loc (? x))
         (door (? x) (? y))
      add (loc (? y))
      delete (loc (? x)))

(STOP if (loc r4)
      add (stop))
```

Below, write the sequence of `(loc ...)` assertions that would result from running these rules with the assertions given above. Write only the entries you need and do not write entries beyond what fit in the table.

Step	Room the robot is in: (loc .))
1.	R1
2.	
3.	
4.	
5.	
6.	

Part E (4 points)

How would the behavior of the system change if we added the following rule (at the end of the rule list):

```
(SYM if (door (? r) (? s))
     add (door (? s) (? r)))
```

Explain briefly.

Part F (4 points)

Let's consider the general case. You are given

- rules `GO` and `SYM` above,
- a set of `door` assertions (indicating connectivity),
- an initial `loc` assertion (indicating the start room), and
- a rule like `STOP` (indicating the goal room).

Assume a path exists from the start to the goal. You have no control of the order of the assertions or the rules.

Which of the following conflict resolution rules *guarantee* that the rule-based system will halt with the robot at the goal room. Circle the number of all that apply:

1. Fire first triggered rule instance, as in the unmodified PS2 chainer.
2. The "rule recency" based system you implemented in PS2.
3. A "data recency" based system, which fires the triggered rule instance that matches the assertion which has been **least** recently used in a fired rule. Any ties are settled by order of rules and assertions (as in the unmodified PS2).
4. A combination of "data recency" and "rule recency" where data recency (as described above) is primary and instances that are tied with respect to data recency are chosen on the basis of rule recency (as in PS2). Any remaining ties are settled on the basis of order of rules and assertions.
5. A combination of "data recency" and "rule recency" where rule recency is primary but rule instances that are tied with respect to rule recency (as in PS2) are chosen on the basis of data recency (as described above). Any remaining ties are settled on the basis of order of rules and assertions.
6. Fire all triggered rule instances.

Problem 2 Backward Chaining (25 points)

Having had some practice with family trees and robots, you decide to plow right ahead and see what it would mean to do some of the same tasks with a backward chaining system.

Note that there are two separate sub-problems here, one concerning family trees, the other concerning robots.

Note also that the rule sets in this problem are different from the ones in the forward chaining problem.

Part A (8 points)

As before, you have a collection of assertions expressed in terms of `is-parent-of` and `is-of-gender`.

You are also given the following two rules, to be used by a backward chainer.

```
(R1 IF
     ((? h) is-husband-of (? w))
     ((? m) is-parent-of (? w))
     ((? m) is-of-gender Female)
THEN ((? m) is-motherinlaw-of (? h)))

(R2 IF
     ((? h) is-of-gender Male)
     ((? h) is-married-to (? w))
THEN ((? h) is-husband-of (? w)))
```

You have the following assertions in the database:

```
(Sam is-married-to Betty)
(Bob is-parent-of Betty)
(Jane is-parent-of Betty)
(Jane is-parent-of Judy)
(Fred is-married-to Judy)
(Bob is-parent-of Judy)
```

You know (but the system does not) that Sam is male and that Jane is female. You do not know anything about other people.

The system now tries to infer by backward chaining whether `(Jane is-motherinlaw-of Sam)`. In space below, draw a tree showing the sequence of subgoals produced, show the database queries, and show the questions that the system would generate.

Tree

Database queries

Questions asked

Part B (3 points)

Imagine that the backward chainer always tried to accomplish its subgoals by asking before doing anything else. Assume that you try the same query as in the example above, and when trying to accomplish its first subgoal beneath the initial query, the system asks you a question. In response, you reply "Why." Indicate how the system should reply if it were trying to make its reasoning process clear.

Part C (3 points)

Write a rule that infers `is-soninlaw-of` (Sam is the son-in-law of Jane because he is the husband of Jane's daughter).

Part D (8 points)

Now let's try getting robots to work from a backchainer. In this case we want to infer how to get from the room the robot is `currently-in` to the goal room.

The rules for this problem are:

```
(Rule1
  IF
     (door (? x) (? y))    ; if there's a door between the rooms
     (currently-in (? x))  ; and I'm IN the first room
THEN (get-to (? y)))       ; then I can GET TO the second

(Rule2
  IF
     (door (? x) (? y))    ; if there's a door between the rooms
     (get-to (? x))        ; and I can GET TO the first room
THEN (get-to (? y)))       ; then I can GET TO the second
```

The room layout is shown below, along with the set of assertions used to represent it, and the assertion indicating where the robot is. Assume the rules and assertions are used in the order shown. If an assertion is not in the database, you can answer "no" when the system asks about that assertion.

```
   -----------------------------------------
   |                                       |
   |                                       |
   |             Goal-Room                 |
   |                                       |
   +---  ----+----  ---+---  ----+----  ---+
   |         |         |         |         |
   |         |         |         |         |
   |  R1     |  R2     |  R3     |  R4     |
   |         |         |         |         |
   +- ---- --+- ---- --+- ---- --+- ---- --+
   |    |    |    |    |    |    |    |    |
   |    |    |    |    |    |    |    |    |
   |R5  |R6  |R7  |R8  |R9  |R10 |R11 |R12 |
   |    |    |    |    |    |    |    | XX |
   +----+----+----+----+----+----+----+----+
(door R1 Goal-Room)
(door R2 Goal-Room)
(door R3 Goal-Room)
(door R4 Goal-Room)
(door R5 R1)
(door R6 R1)
(door R7 R2)
(door R8 R2)
(door R9 R3)
(door R10 R3)
(door R11 R4)
(door R12 R4)
(currently-in R12)
```

The system now tries to infer by backward chaining whether `(get-to Goal-Room)`. In space below, draw a tree showing the sequence of subgoals produced, show the database queries, and show the questions that the system would generate.

Note: *You may stop after showing what happens as a consequence of the first 8 questions asked.*

Tree

Database queries

Questions asked

Part E (3 points)

What general inference can you draw from this problem about the kinds of tasks for which backward chaining systems are *not* well suited? Describe briefly what about the task above makes it not well suited.

Problem 3 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. Two copies are provided in case you need more than one for scratch work.

Black dots indicate intersections ("the nodes" from the search perspective).

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

Important special notes

- A node is considered to be in the search tree if it appears anywhere on any path that appears at any time on the search queue.
- The map is drawn to scale.
- Nodes S, 1, 3, and G lie on a straight line.
- All nodes other than S, 1, 2, and G lie on one of four circles centered at node 1.
- When you are asked to use a method, such as hill climbing, that involves a heuristic measure of "goodness," you are to use straight-line, as-the-crow-flies, distance-to-goal as that heuristic measure.
- When you are asked to use a method that involves a path length, you use the sum of the Euclidian distances between adjacent nodes along the path.

Important variation on breadth-first search

You are frequently asked to report the **total number of nodes** in a search tree. As mentioned above, a node is considered to be in the search tree if it appears anywhere on any path that appears at any time on the search queue.

To **reduce ambiguity** and **simplify** your calculations, you are to use the version of breadth-first search described, in part, in the following paragraph. **This version differs slightly from the version that is described in the book.**

- During breadth first search, extension and termination checking alternate. That is, after all paths containing n nodes are placed on the search queue, then all those newly-placed paths are checked to see if any has a terminal node that is the goal node, and if so, search stops. Otherwise, all paths are extended so that they contain $n+1$ nodes, and after all those paths are placed on the search queue, then all those newly-placed paths are checked to see if any has a terminal node that is a goal node. And so on. In other words, extension to a level is never interrupted by a termination check.

Reminders

- No paths containing loops are ever placed on the search queue.
- During hill climbing search, only the first path on the search queue is checked to see if its terminal node is the goal node. Thus, hill climbing, if done without backtracking, could generate paths to the goal that never become the first path on the search queue. Such paths would be overlooked.

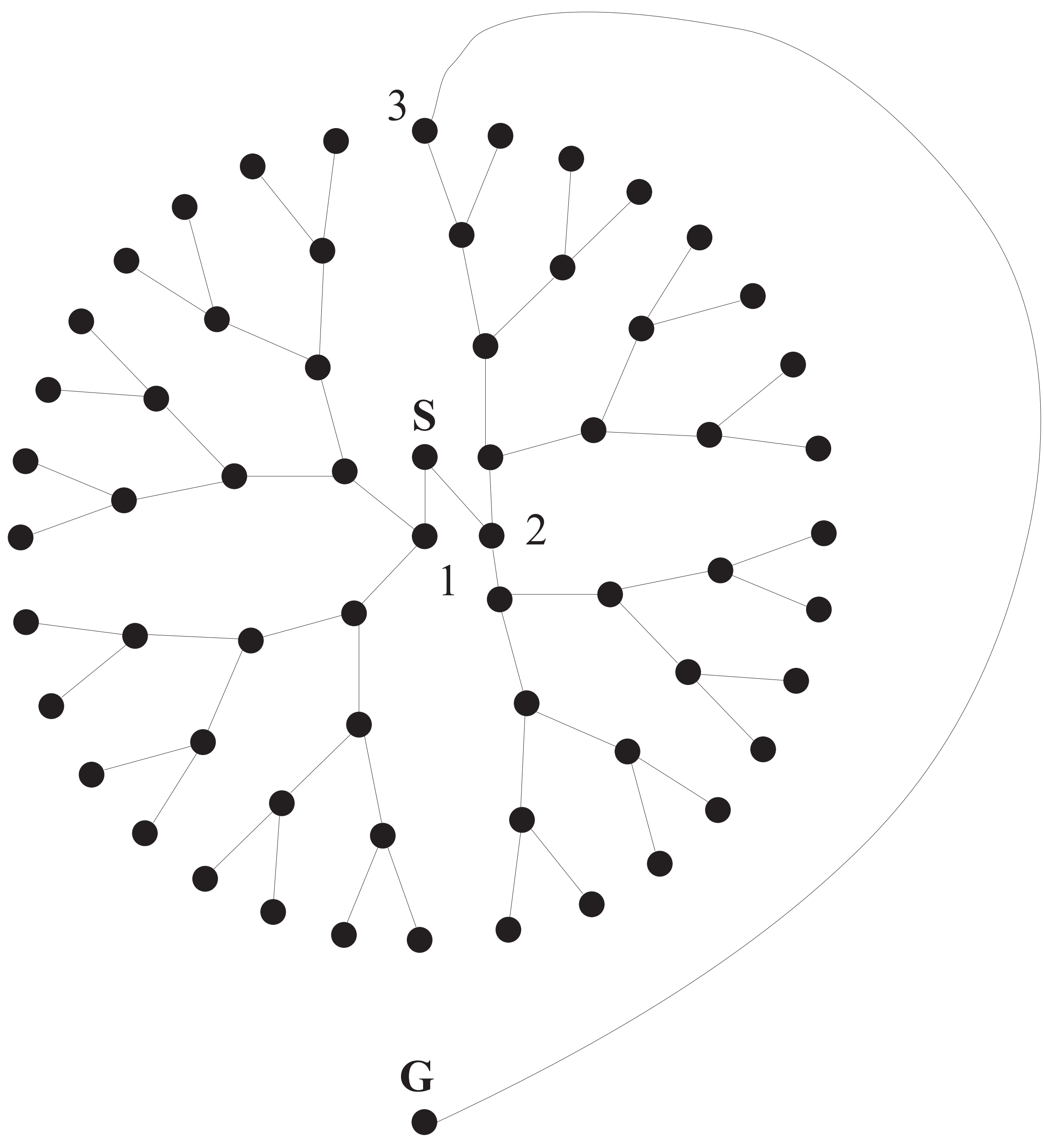


Figure 1: A weird city.

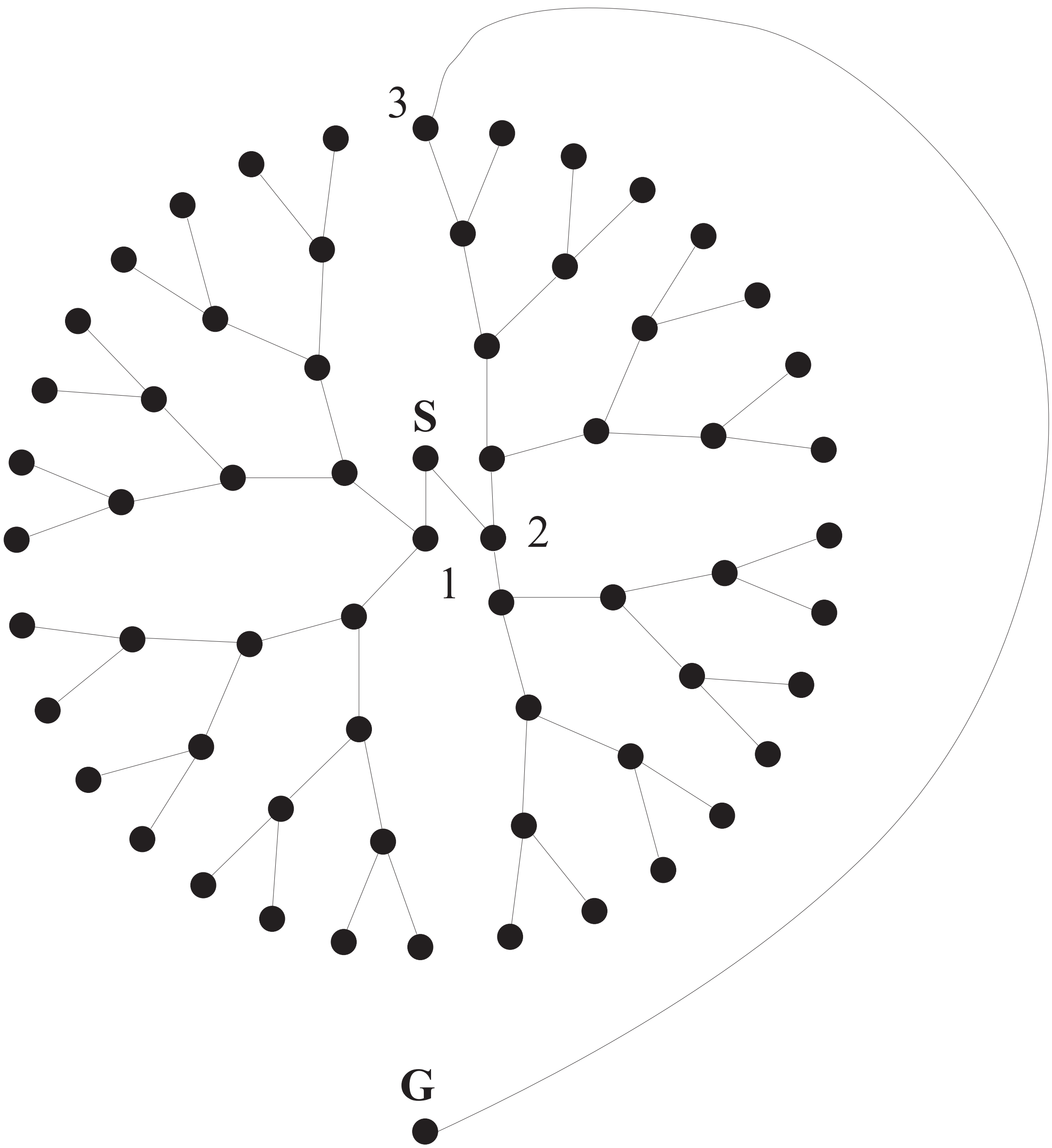


Figure 2: A weird city. Second copy

Part A (4 points)

You ask several friends for advice in finding a route. One suggests hill-climbing, **without** backtracking. That is, once a choice is made at a node there is no possibility of backing up and trying other choices.

The suggested method finds a path to the goal (T or F)?

What is the total number of nodes in the search tree?

Be sure to count carefully, as there will be little or no partial credit given in this problem. You should probably reread the various notes, above, to be sure you understand which measure of goodness you are to use in hill climbing and what nodes are placed on the search tree.

Part B (3 points)

Another friend suggests hill-climbing, **with** backtracking.

The suggested method finds a path to the goal (T or F)?

What is the total number of nodes in the search tree?

Part C (3 points)

Another friend suggests breadth-first search.

The suggested method finds a path to the goal (T or F)?

What is the total number of nodes in the search tree?

Part D (3 points)

Not yet having run out of friends, still another suggests that you use hill-climbing, without backtracking, but this time you search from G to S.

The suggested method finds a path to the goal (T or F)?

What is the total number of nodes in the search tree?

Part E (3 points)

Still another friend suggests that you use bidirectional breadth-first search. That is, you expand alternately from the S and G, producing two search trees.

More precisely,

- You perform one complete level of tree expansion from S, then one complete level of tree expansion from G, then another from S, and so on.
- When extending one of the two trees, all paths are extended to the next level before any of those newly extended paths are checked to see if they terminate at a leaf node of the other tree.

What is the total number of nodes in the search trees? $\boxed{}$

Part F (3 points)

Yet another friend suggests that you use branch-and-bound search so as to ensure an optimal path. What is the total number of nodes in the search tree? $\boxed{}$

Part G (3 points)

Your boss orders you to use A* search from S to G so as to not only ensure an optimal path, but reduce the size of the search tree. What is the total number of nodes in the search tree? $\boxed{}$

Part H (3 points)

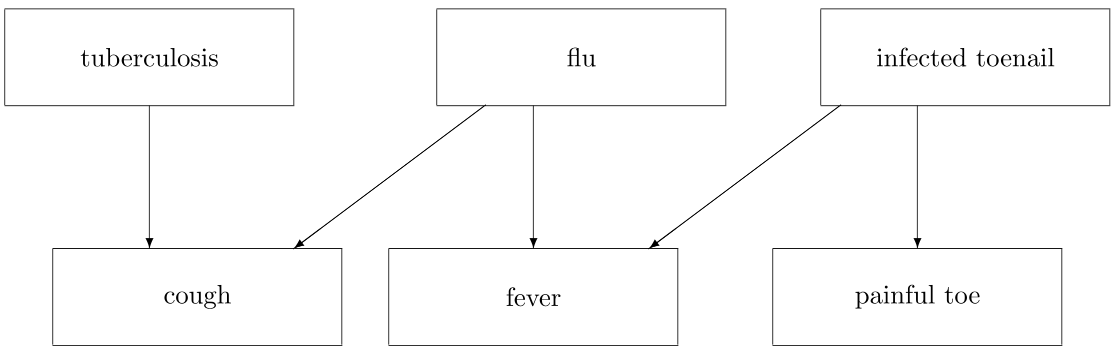
Your boss then, after further thought, orders you to use A* search from G to S, and claims that the search tree will be much smaller than if you were to use A* from S to G.

Your boss is correct (T or F)? $\boxed{}$

Problem 4 Constraints (25 points)

In this problem, we consider using constraint propagation techniques to solve the problem of diagnosis. Consider a simplified version of medical diagnosis, in which people have *symptoms* (observable abnormalities, such as a cough) and *diseases* (underlying causes of their symptoms, which may not be directly observable, such as a flu). Diseases are related to symptoms by a *may cause* relationship, which says that a particular disease may cause a particular symptom. Note that having a disease does not necessarily mean that the patient has each of the symptoms it may cause, but having a symptom means that the patient must have at least one of the diseases that may cause it.

A *diagnosis* is a set of diseases that may together cause all of the observed symptoms of the patient. For example, in the simplified disease world below



for a patient who has both a cough and a fever, but not a painful toe, the diagnosis of *flu* would be adequate, because it can, according to this model, cause both observed symptoms. A diagnosis consisting of all three diseases would also explain the two symptoms, but would not be *minimal*, because a simpler (smaller set of diseases) explanation is already adequate. If we also added that the patient has a painful toe, then no single-disease diagnosis is adequate, but either {flu, infected toenail} or {tuberculosis, infected toenail} would work. In the first case, fever is actually explained twice, so we may arbitrarily choose one of its causes.

Dr. Ben Bitdiddle (he's now completed medical school and has a license to kill) suggests the following strategy for implementing a constraint propagation method such as that of problem set 5 for solving the diagnosis problem:

1. Represent the problem domain by a graph of variables such as the diagram above. To diagnose a specific case, make a copy of the domain that omits all symptom variables (and links to them) that the patient does not have. Thus, for a specific case, the graph consists of all the disease variables and all those symptom variables that correspond to symptoms seen in the patient.
2. By analogy with map coloring problems where we try to find the minimum number of colors needed for a particular map, consider first whether a diagnosis consisting of a single disease is adequate. If not, try diagnoses consisting of two diseases, then three, etc., until a solution can be found.
3. We use the same trick that PS5 employed for limiting schedules to a fixed number of "term slots". When trying to find a diagnosis with (say) two diseases, we define two disease values, DX1 and DX2. The domain of each disease variable, then, is DX1, ..., DXn, NOTPRESENT. NOTPRESENT is the value of a disease variable that represents the patient *not* having that disease.

Complete Ben's strategy by answering the questions below:

Part A (5 points)

What should be the domains of the symptom variables? (Remember that only those symptom variables are represented that are known to be present in the patient.)

Part B (5 points)

What are the constraints, if any, among disease variables?

Part C (5 points)

What are the constraints, if any, among symptom variables?

Part D (5 points)

What are the constraints, if any, between disease and symptom variables?

Part E (5 points)

For this subproblem, suppose that we are trying to get your completed implementation of Ben's suggested diagnostic approach to find the single disease diagnosis of flu for a patient with just cough and fever.

Using backtracking search with `propagate-constraints` (`backtrack-pc` in problem set 5), if the search engine first assumes that tuberculosis is the diagnosis (i.e., its variable is assigned the value DX1), it will eventually run into a failure, from which it must backtrack to find the correct solution. Write down the tentative variable assignments and list the domains of the unassigned variables at the point of failure, and indicate the specific failure that leads to backtracking.

Assume that search explores all alternatives in left-to-right order according to the diagram, where that makes sense. For example, variables are ordered by their left edges, in the order tuberculosis, cough, fever, flu, infected toenail. Disease variables take on possible values in the order DX1, ..., DXn, NOTPRESENT. Make a left-to-right consistent choice for the order in which the values of symptom variables are explored.