Assignment 2 (10 marks) Due by: 24 September 11:59pm

Mike Barley

August 17, 2018

Contents

1 Getting Prolog

You should use SWI prolog. It is free and has a graphical debugger. You should be able to easily find it on the web.

2 Part 1: Logic Problems

1. Family Problems

   Assume you have fact predicates: *ChildOf(child, parent)*, *Male(person)*, *Female(person)*.

   You need to define the following in terms of the above predicates:

   - *MotherOf(mother, person)*
   - *SisterOf(sister, person)*

2. Tree Problem

You need to define *InSameTree(nodde1, node2)* in terms of *ParentOf(Parent, Child)* where *Parent* and *Child* are nodes. Note that there may be many trees and different nodes will be in different trees.

Remember you are allowed to write helper predicates to use in your definition of *InSameTree*, but you will also have to provide definitions for those helper predicates.

Note: *InSameTree(X, X)* is always false. In other words, we don't want nodes to be in the same tree as themselves, this is similar to not wanting people to be their own siblings.

3. General Comments

   Remember if you write a recursive definition, you must give the base cases as well as the recursive cases.

   For the quantifiers $\forall$ and $\exists$ use "A" and "E", e.g., $\forall$(x) $\exists$(y) x < y should be written A(x)E(y) x < y. For connectives $\iff$, $\wedge$, and $\vee$ use "iff", "and", and "or", e.g., $(x < y \wedge y < z) \vee (x > z) \iff comparable(x, y, z)$ should be written $(x < yandy < z)or(x > z)$ iff $comparable(x, y, z)$.

3 Part 2: Prolog Predicates

Take your logic definitions from above and rewrite them as Prolog predicates.

4 Part 3: Search as Problem Solving

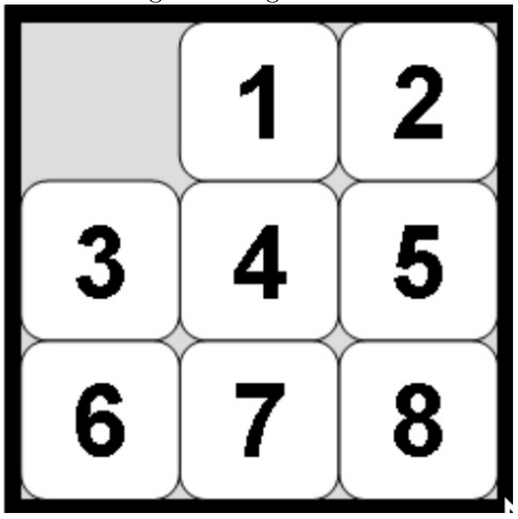
In order to keep this as simple as possible, we will not use heuristics and only do problems in a unit-cost domain. Specifically, you will implement a breadth-first algorithm for the 8-puzzle domain. The predicate will be called *breadthFirstSearch* and have three arguments: an input argument, the *InitialState*, and two output arguments, *Solution* and *Statistics*.

1. *breadthFirstSearch(+InitialState, -Solution, -Statistics)*

   The domain will be the eight-puzzle as shown in Figure 1, see Wikipedia for "Fifteen Puzzle" for a description of the goal and the rules. We need to define the input and output formats. The *state* format will be used to describe the initial state and a solution is simply a list of states. An eight puzzle *state* is represented as a list of 9 numbers (the numbers from 0 to 8). The number 0 denotes a blank space, and numbers 1 through 8 denote tiles labelled 1 through 8.

   The nine places in the list represent the nine positions on the puzzle, so, the first position states which tile is in the upper left-hand corner, and the ninth position states which tile is in the lower right-hand corner. The state shown in Figure 1 is represented by the list [0,1,2,3,4,5,6,7,8]. The state [1,2,3,4,5,6,7,8,0] represents the goal state. All eight puzzle problems

Figure 1: Eight Puzzle



have this as their goal state. A solution represents a legal path from the initial state to the goal state. A solution is represented as a list of states starting with the initial state and continuing with intermediate states until the goal state.

Besides a solution, *breadthFirstSearch/3* also returns some statistics. The statistics argument is a list of statistic terms starting with the statistics for g-level 0 and giving the statistics for each successive g-level until the final g-level (when the goal is found). A statistics term is the predicate *stat(GLevel,Generated, Duplicated, Expanded)*, where *GLevel* is the g-value of the node doing the "action". So, *(3,23,7, 16)* says there were 23 nodes with a g-value of 3 generated, 7 of them were duplicates and 16 of the non-duplicates were expanded. If the number of duplicate nodes and expanded nodes don't add up to the number generated, then the difference represents the nodes left on the open list (this only happens on the final g-level).

Normally, breadth-first search stops as soon as a goal state is generated but, like A^*, your algorithm should stop when a goal state is chosen to be expanded. This makes the algorithm less efficient but is easier to code.

2. Resources

   You are provided with four auxillary files that you will want to use in writing your code:

   - counter.pl
   - queues.pl
   - eightPuzzle.pl

- problems.pl

The first will help you to keep track of the statistics.

The second will simplify your writing of the breadth-first algorithm, these algorithms use queues to store the open list.

The third will implement the eight puzzle domain for you. You will need to use the *goal8(State)* and the *succ8(State,Successors)* predicates. The former tests whether a State is a goal state, and the latter returns a list of all the neighbors of State. The neighbors are ordered pairs *(EdgeCost, Neighbor)*, where EdgeCost will always be one since this is a unit-cost domain, and Neighbor will be a state that is a single edge away from State.

The fourth will contain simple problems for you to test your code.

Writing the code will probably be easier if you use the SWI library "record" to define fields for your open list nodes. Since you will need to return a solution, you will want to store the parents of states.

5 What and When You Need To Turn In

You need to turn in:

- family.pl which should contain both the logic definitions for the family predicates *motherOf/2*, *sisterOf/2*, and *inSameTree/2* and the prolog implementation of these definitions. This file should NOT contain any facts. Marks will be taken off if they are included!!

- breadthFirstSearch.pl which should contain *breadthFirstSearch/3* and all the predicates you have written to make it work. You should not include any of the predicates from queues.pl, eightPuzzle.pl, or counter.pl.

You will need to turn these into the ADB (`https://adb.auckland.ac.nz/`) no later than 24 September 11:59pm.

6 Marking

There will be a total of 4 marks for the both the logic definitions and the prolog implementations. There will be .5 marks each for correctly defining in predicate logic *motherOf* and *sisterOf* and another .5 marks each for their correct prolog implementation. There will be 1 mark each for the correct predicate logic definition of *inSameTree* and for the correct prolog implementation for *inSameTree*.

There will be a total of 6 marks for the breadth-first search implementation.

- 4 marks for the correct implementation of breadth-first search

  - 1 mark for correctly solving "trivial" problems, i.e., problems where the initial state is the goal state
  - 1 mark for correctly returning the solution with the states in the correct order from the initial state to the goal state
  - 2 marks for correctly solving non-trivial problems

- 2 marks for the correct logging of the statistics
  - 1 mark for correctly recording the g-level of the statistics
  - 1 mark for correctly recording all three statistics for the g-level up to but not including the level where the optimal solution is found