Assignment 6: Prolog Searching
CISC 260, Winter 2018

Due Date: 9 a.m. on Friday, April 6
You may submit late up to 9 a.m. on Saturday, April 7 with a 10% late penalty

I'm giving you longer than usual for this assignment because I know how busy the last two weeks of the semester can be. I can't extend this deadline any further except for people with special circumstances (see next paragraph). Even if you're really busy, please look at the assignment early and make sure you understand what is required. I am hoping to set up a help session on the evening of Monday, April 2, so try to get a start by then.

Administrative Notes:
If there are special circumstances such as illness that prevent you from completing the assignment on time please submit the appropriate form to the Arts & Science portal https://www.queensu.ca/artsci/accommodations (or the equivalent form for your home faculty if you are registered in a faculty other than Arts & Science). The instructor will be notified when a student submits one of these forms; you do not have to contact her directly. If you have special circumstances that force you to miss the deadline for ONE assignment there's no action necessary; the assignment you miss will be the one you drop. If you have special circumstances that affect more than one assignment we will will need to talk and make a plan that fits your situation.

The Puzzle:
For this assignment, you must solve a puzzle using Prolog. The original puzzle is taken from an old "Problems and Puzzles" web page, which doesn't seem to exist anymore. It claims that this problem has been given at Microsoft job interviews. Here it is, as stated on that web page.

> A family of 4 is trying to cross a bridge at night. One needs a flashlight to cross the bridge, and only two persons can cross the bridge at the same time, moving at the speed of the slower of the two. Father crosses the bridge in 1 minute, Mother in 2 minutes, Child in 5 minutes and Granny in 10. What is the fastest way for them to cross the bridge?
>
> The obvious solution is for F to always carry the flashlight back, i.e., FM+F+FC+F+FG which means 19 minutes (the notation means: first, F and M cross, then F carries the flashlight back, third, F and C cross together, fourth, F carries the flashlight back again, finally, F and G cross). Can you do faster?

Here is a solution for the puzzle as given (one of two possible solutions that get the job done in 17 minutes):

1. Mother and Father cross the bridge (2 minutes)
2. Father goes back across the bridge alone (1 minute)
3. Granny and Child cross the bridge (10 minutes)
4. Mother goes back across the bridge alone (2 minutes)
5. Father and Mother cross the bridge (2 minutes)

For this assignment, we will generalize the problem to work with any family, not just this particular family. To define a family, we'll use a "family" fact, where the first parameter is a name for the family and the second is a list of the family members. Each family member is in the form Name/Time. For example, we would define the family in the original example with this fact:

```
    family(original, [father/1, mother/2, child/5, granny/10]).
```

A second modification is that you're not required to find the fastest way to cross the bridge. You're only required to find all of the possible ways in which a family could cross the bridge, and the time that each way would take, keeping under a specified time limit.

I'm going to assume that the family starts off on the north side of the river and will cross to the south side -- just because it helps to have names for the two sides.

The moveFamily Predicate

The only predicate that you're required to provide is this one. `moveFamily` must take four parameters:

1. The name of the family (as given in a family fact) -- must be bound
2. A maximum time for the total crossing -- must be bound
3. A list of moves that would take the family across the river -- must be unbound
4. The time this list of moves would take -- must be unbound

Each element of the list of moves must be a single name or a pair of names connected with "+". See the sample run below for examples.

You may, of course, create as many helper predicates as you wish.

Your code should assume that the user will be bringing in "family" facts from another file; you shouldn't include any in the file you hand in. I will be posting a file called families.pl on OnQ with these instructions and you can import that as well as your solution for testing. You can also make up test families of your own, but please don't put them in the same file as the solution you hand in.

Observation

The flashlight starts off on the north side of the river, so the first move will take one or two people from north to south. The second move will take one or two people from south to north, and so on, ending in a move from north to south. It is safe to assume that each trip from north to south will involve exactly two people and that each trip from south to north will involve exactly one person. This observation will simplify the search and also ensure that there will be no *no cycles*. So in contrast to some of the graph searches we discussed in class and in the "virtual lecture" it's not necessary to keep track of previous states.

An Additional Requirement: If you're not careful you will get a lot of results which are duplicates except for the order of names in the pairs. For example, the simplest test case I used for experiments was this one:

```
    family(two, [fred/1, george/2]).
```

for which the solution, of course is that Fred and George cross the bridge together and that's all. For this case my initial solution generated these two solutions:

```
    20 ?- moveFamily(two,2,Moves,Time).
    Moves = [fred+george],
    Time = 2 ;
    Moves = [george+fred],
    Time = 2 ;
    false.
```

For larger families you're going to get irritating amounts of duplicate answers if you include both possible orders for each pair of people. For example, in the original example a solution requires three different pairs of people to cross the bridge together. There will be *eight* different versions of each solution, with the pairs in all eight possible combinations of orders. You must avoid this by adding an extra requirement to your rules that only generates solutions in which the names in each pair are in alphabetical order. You can use the Prolog operator `@<`, which compares alphabetical atoms in the usual lexicographical order. For example, `cat @< dog` is true but `monkey @< dog` is false. Before you consider a solution in which `Person1` and `Person2` cross the bridge together, add the requirement `Person1 @< Person2`. If you leave this requirement out you're going to get large numbers of equivalent solutions, just differing by the order people are listed in the pairs. This is going to make it harder for you to reason about your results and tell if they're correct. It's also going to make it harder for us to mark your solution, so we will deduct a point if you neglect this simple check.

Helper Predicates (Optional)

The only definite requirement for this assignment is that you provide a correct `moveFamily` predicate. In this section, I'm going to describe the two helper predicates I used in my solution. If you have a different idea about structuring your solution, feel free to go with it. Otherwise, you can use this section as a bit of guidance to get you started.

I wrote a pair of helper predicates: `moveNorth(North, South, Moves, Time)` and `moveSouth(North, South, Moves, Time)`. In each predicate, `North` and `South` are lists of people who are currently on the north and south sides of the river and would be bound when you invoke the predicate. Each predicate binds `Moves` to a sequence of moves that will move everyone still on the north side to the south side, and `Time` to the number of minutes this will take. The difference is that `moveNorth` must generate a set of moves that starts with a move of one person from the south to the north -- i.e. it assumes that the flashlight starts off on the south side of the river and its first move will be towards the north. And `moveSouth` starts with a move of two people from the north to the south -- assuming the flashlight starts off on the north with a move towards the south.

As long as there are people left on the north side of the river, `moveSouth` and `moveNorth` invoke each other recursively. My `moveFamily` predicate uses `moveSouth`, since the flashlight always starts off on the north side.

My helper predicates didn't include a maximum time parameter. You could add one to make your solution more efficient, but I was lazy and just used `moveNorth` and `moveSouth` to generate all possible solutions and then cut off solutions over my time limit in the main `moveFamily` predicate. I found that my solution was efficient enough with families of reasonable sizes.

A Useful Prolog Function
When two people go across the bridge together, the time they take is the maximum of the time each would take by themselves. In the original puzzle, the mother takes 2 minutes and the child takes 5 minutes. If they cross the bridge together they take 5 minutes. So you're going to need to find the maximum of pairs of numbers. Prolog contains an arithmetic function called `max` that finds the maximum of two numbers. For example:

```
?- N is max(5,2).
N = 5.

?- X is max(3,7).
X = 7.
```

Note that `max` is a *function*, not a predicate. It is like a function in Haskell, Java, and most other languages: it takes a parameter and returns a result. It may only be used inside an arithmetic expression (i.e. on the right side of "is" or on either side of "=:=").

Testing Advice
Make sure that you write your predicates without any assumptions about the family. I will post a file on OnQ called `families.pl`, which defines several example families, including the original one described in the puzzle.   I will also post a transcript of a test using my solution (details below).

You can use these families for testing and/or create your own. If you haven't tested your `moveFamily` predicate with at least two different families of different sizes (preferably more than two), you haven't tested enough to be sure your predicate is completely general.

The TAs will test your solution with families that are different from any of the test cases given. The more testing you do, the more likely you are to shake all the bugs out, so that your solution will work with the test families used in marking.

Test Transcript:

To get the transcript I posted on OnQ, I consulted `families.pl`, then consulted my solution, then typed the queries you can see in the transcript.

One More Useful Predicate:

In its default mode, Prolog tends to cut off long lists when it displays them, which can interfere with the testing of a program like this. Changing the settings involves a rather long, complicated command. I have put that command in a file called `fixOutput.pl,` which I will post on OnQ for your use. If you are seeing output with "…" in it, make sure `fixOutput.pl` is in the same directory as the file you're working with, and type the command `fixOutput.` For the rest of the session your lists should display in full. If you use my `families,pl` file you don't have to bother with this because I've included the commands from `fixOutput.pl` in `families.pl`.

Marking Scheme:

- finding all legal sequences of moves: 4
- finding correct times for all sequences of moves: 3
- pairs of people given in alphabetical order (i.e. fred+george instead of george+fred or both): 1
- total: 8

We reserve the right to deduct a 10% administrative penalty if you disregard the instructions in a way that costs us time and aggravation -- for example, if you change the predicate name so that we have to change our testing module to try your program. And as usual, will deduct a 10% late penalty if you submit your program during the "grace period" after the on-time deadline.