Homework 3

Due May 26th

June 2, 1999

Problem 3.1. = 2.4.

Problem 3.2. In this problem you will write a program to evaluate an expression in the 2-valued (truth-valued) boolean algebra from the last assignment. You may assume that the expression is a constant expression.

Call the relation you define `booleval/2`. A goal of the form

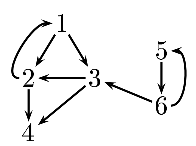
```
?- booleval(Term,Val)
```

should succeed whenever (i) `Term` is a member of T_Σ, (ii) `Val` is a member of the set $\{0, 1\}$, where we take 0 to represent falsehood and 1 to represent truth, and (iii) `Val` is the value that should be assigned to `Term` under the standard interpretation of the symbols of Σ, that is, under the interpretation defined by the truth tables below.

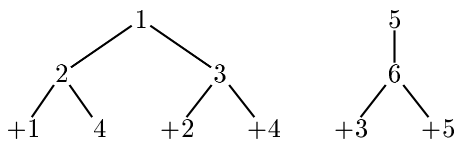
p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \Rightarrow q$
0	0	1	0	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	1	0	1	1	1

Not required, but interesting: Can you get your implementation of `booleval/2` to generate all of the true (or equivalently all of the false) expressions in this algebra?

Problem 3.3. Define a Prolog relation that will unfold a graph into a forest of trees. The trick to this is the following: Obviously, when there are arcs that point to already visited nodes, you need some other way of representing those arcs: they can't be represented in a tree, of course. This means that our trees will have two distinct kinds of leaf: a "real" leaf (representing a node in the graph with no outgoing arcs) and a "pointer" leaf (representing a node that has already been visited before). Suppose we have the following graph, for example:



This should be unfolded to the following forest.



Observe that both the graph and the forest have exactly the same (number of) arcs. Observe also that what are node *names* in the graph get transformed into node *labels* in the trees. In Prolog, this should look as follows, using our standard representation of (ordinary) leaves as `-Label`.

```
| ?- graph(2,G),unfold(G,T).

G = [1-[2,3],2-[1,4],3-[2,4],4-[],5-[6],6-[3,5]],
T = [1/[2/[+(1),-(4)],3/[+(2),+(4)]],5/[6/[+(3),+(5)]]] ? ;

no
```

You may choose to use the other representation of leaves for this assignment, that is, as `Label/[]`. That will save you some code in your solutions, which may make it worth wrestling with the less familiar notation. In that case, the previous goal should now have the following result.

```
| ?- graph(2,G),unfold(G,T).

G = [1-[2,3],2-[1,4],3-[2,4],4-[],5-[6],6-[3,5]],
T = [1/[2/[+(1),4/[]],3/[+(2),+(4)]],5/[6/[+(3),+(5)]]] ? ;

no
```

The version of `unfold/2` used above returns only one solution per graph. However, the forest you get from any particular graph depends heavily on which node you start from. My program always starts with the first node of the remaining graph, but you might prefer to implement a version which nondeterministically selects a node to start from, in which case you should get multiple distinct solutions.

Problem 3.4. *Das Königsberger Brückenproblem.* This problem is also called the problem of finding an Eulerian circuit in a graph. Presumably Euler either lived in Königsberg or else spent one extremely boring afternoon there when all there was to do was to imagine how one might make a walking tour of the city crossing each of its many bridges exactly once.

Well, there you have it. Das Königsberger Brücke Problem. You will represent the city as a graph, with edges for the bridges. A circuit in a graph is a path covering every node or edge in the graph exactly once, and ending where it began. Here it is the edges that matter. You can visit the nodes as often as you like, but each edge only once.

Of course, a bridge can be crossed in either direction. So this is really a problem over undirected graphs. You will probably find this rather tricky to do if you try to use an adjacency list representation of the graph. Simpler is to use a simple list of edges, that is, a term explicitly representing the edge relation.