Task

Your task is to implement three separate programs in Prolog to solve the following problems:

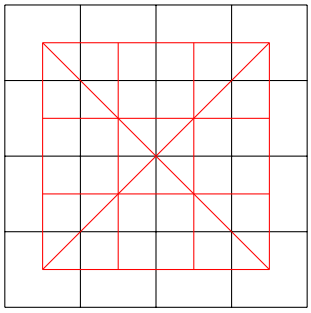
1. Magic Squares for the 4×4 case
2. The Gap Problem
3. Maximum Matching Submatrix

Your programs must implement the predicates specified in the descriptions below. All of your files will be loaded using the `consult/1` predicate for the file specified in the problem descriptions.

The `trace` command of SWI-Prolog will be helpful with understanding the unification process.

Magic Squares

A magic square is set of distinct non-negative integers aligned on a square grid such that all of the rows, columns, and the two diagonals sum to the same goal value. Below is an example 4×4 grid with the straight red lines indicating what cells sum together.



You will implement the predicate `magic/3` satisfying `magic(+Goal, +Values, -Solutions)`. When this predicate is true, `Goal` is the goal value, `Values` are the existing values of the magic square with "_" (underscore) for unknown values in row major order, and `Solutions` is a list of lists of all solutions satisfying `Goal` and `Values` with each individual solution in row major order. Place your code into a file named `magic.pl`.

To help with your testing, below are two example 4×4 magic squares. They both have a goal value of 34.

0	1	18	15
7	19	2	6
13	3	10	8
14	11	4	5

4	14	15	1
5	8	11	10
12	9	6	7
13	3	2	16

With a goal value of 42, the following square with 8 unknowns leads to two solutions.

3	20	_	_
4	15	_	_
14	5	_	_
21	2	_	_

$\rightsquigarrow$

3	20	9	10
4	15	6	17
14	5	16	7
21	2	11	8

3	20	19	0
4	15	16	7
14	5	6	17
21	2	1	18

You are encouraged to use a simple brute force solution for magic squares. It won't be fast, but it will show the power of Prolog for this type of problem. If you want to make your implementation fast, look into the `clp/clpfd` library that is part of SWI-Prolog.

Hints:

- Consider writing your program to work on a 3×3 magic square first. While not required for the assignment, it may prove easier to refine your program's logic on this smaller case.
- The `findall/3` predicate will be useful to collect all of the answers.

The Gap Problem

Given an *unordered* list of integers, the gap problem asks what is the largest element of the last gap in the list.

For example, suppose you are given as input the list $(4, 7, 2, 3, 8, 5)$, the answer is 6. For the list $(0, 1, 9, 4, 8, 6, 2, 5)$ the answer is 7, even though there is also a gap for 3. For the list $(4, 9, 8, 5, 7, 6, 0)$, the answer is 3, even though the gap is 1, 2, and 3. For the list $(-6, -3, -2)$ the answer is -4.

There are several special cases. If there is no gap, the output is the minimum element less one. For example, with an input of $(0, 1, 2, 3, 4, 5)$, the output is -1. If the input is the empty list, the problem is undefined. That is, it doesn't matter what your program answers, so long as it gives some answer and does not crash. If the input contains duplicate elements, as in $(0, 0, 0)$, or $(2, 6, 3, 5, 4, 6)$, the problem is undefined.

You will implement the predicate `gap/2` satisfying `gap(+List, -Gap)`. When this predicate is true, `List` is a list of unordered integers, and `Gap` is the solution to the gap problem for `List`. Place your code into a file named `gap.pl`.

Maximum Matching Submatrix

Let A be an $n \times n$ matrix with integral entries and B be an $m \times m$ matrix with integral entries such that $m \leq n$. The maximum matching submatrix problem is to find the pair (i, j) such that when the $(1, 1)$ entry of B is overlaid upon the (i, j) entry of A, the number of entries in B that match entries in A is maximized. If B does not entirely fit within A when moved to position (i, j), then the entries will wrap around horizontally or vertically as necessary. In the event of a tie, the lexicographically smaller pair is the answer. The coordinates of the first row and column in both matrices is $(1, 1)$ (i.e. 1-based indexing instead of 0-based indexing).

You will implement the predicate `submatrix/3` satisfying `submatrix(+A, +B, -Answer)`. When this predicate is true, `A` and `B` are row major lists for the matrices A and B of the problem, and `Answer` is a list containing three elements: i, j, and the number of overlaps, in that order. Place your code into a file named `submatrix.pl`.

Several examples follow:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \text{Answer: } (2, 1),\ 9 \text{ matches}$$

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \qquad B = \begin{pmatrix} 4 & 1 & 1 \\ 1 & 5 & 1 \\ 1 & 1 & 9 \end{pmatrix} \qquad \text{Answer: } (3,4),\ 6 \text{ matches}$$

$$A = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 3 & 1 & 2 & 0 \\ 1 & 3 & 1 & 1 \\ 1 & 1 & 4 & 1 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \qquad \text{Answer: } (2,2),\ 8 \text{ matches}$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \qquad \text{Answer: } (1,1),\ 4 \text{ matches}$$

Submission

Your programs will be submitted via Subversion. Your repository for the class is `https://svn.cs.du.edu/courses/comp3351/f2011/myusername` where `myusername` is your CS account name.

- Put the files for magic squares into a subdirectory named `hw4/magic_squares`.
- Put the files for the gap problem into a subdirectory named `hw4/gap`.
- Put the files for the maximum matching submatrix into a subdirectory named `hw4/submatrix`.

In addition to your program source, you must include a text file that demonstrates the usage of your programs. Call this text file `usage.txt` and put it into the `hw4` subdirectory.

Further information about using Subversion can be found at `https://svn.cs.du.edu`.

Deadline

Your program must be submitted by Subversion before 11:59PM on Thursday 2011-11-10. Programs will only be accepted by Subversion. Do not email any portion of your solution for submission. Late homework will *not* be accepted.