CUHK CSCI3230
Lisp Programming Assignment (Optional)

Due date: 11th Oct., 2015 23:59

Introduction

Nonogram, also known as Hanjie or Griddler, is a 2-D logic puzzle. It is a puzzle invented in 1987 by Non Ishida. The game is played within a rectangular grid. For each cell, the player may either colour it or left it empty according to the constraints on the sides of the grid. The puzzle is solved when the combination of the coloured cells and the empty cells meets the provided constraints. You are going to write a Lisp program to find out one of these combinations in the puzzle based on the given hints. In other words, you are going to write a *Nonogram Solver*.

Specification

In this assignment, the state of a MxN Nonogram grid is encoded by a list[1] of lists

$$'((x_{00}, x_{01}, x_{02}, ..., x_{0(N-1)}), '(x_{10}, x_{11}, x_{12}, ..., x_{1(N-1)}) , ..., '(x_{(M-1)0}, x_{(M-1)1}, x_{(M-1)2}, ..., x_{(M-1)\,(N-1)})$$

Each entry x_{ij}, for $0 \le i < M$ and $0 \le j < N$ represents one of the three states of a cell. The three states are *box*, *space* or *unknown*. And we will use **b**, **s** and **u** to encode these three states respectively. If a cell is coloured, it will be in the box state. If a cell is left empty, the cell will be in the space state. The cell is in the unknown state when one cannot determine whether a cell is a box or a space. Any legitimate solution should never have any unknown state. If a puzzle has multiple solutions, the solver should return one of the solutions.

The constraints of the puzzle can be divided into two types: the constraints for the rows and the constraints for the columns. Each constraint composes of some non-negative integers and specifies how many unbroken lines of coloured-squares there are in any given row or column. For example, a row constraint “1 3 2” means that the line of blocks contains a line of length 1, a line of length 3 and a line of length 2. These lines must be separated by at least one cell in space state. Suppose the row is 9-cell long, it can only be satisfied by three combinations as shown in fig. 1.

Combination 1	b	s	b	b	b	s	b	b	s
Combination 2	b	s	s	b	b	b	s	b	b
Combination 3	s	b	s	b	b	b	s	b	b

Figure 1: The possible combinations for a row of length 9 given the row constraint “1 3 2”. The character in a cell shows the state of a cell.

[1] *list* refers to Lisp list.

The constraints of the rows and columns can be found on the sides of the grid. A MxN Nonogram should have M row constraints and N column constraints. The constraints of the row are represented by a list of lists

$$'\left(\left(r_{0,0}, r_{0,1}, r_{0,2}, \dots, r_{0,(N_0-1)}\right), \left(r_{1,0}, r_{1,1}, r_{1,2}, \dots, r_{1,(N_1-1)}\right), \dots, \left(r_{(M-1),0}, r_{(M-1),1}, .., r_{(M-1),(N_{M-1}-1)}\right)\right)$$

,where N_i is the total number of integers of the constraint (i.e. count of unbroken lines) in the i^{th} row. Similarly, the constraints of the columns are also represented by a list of lists

$$'\left(\left(c_{0,0}, c_{0,1}, c_{0,2}, \dots, c_{0,(M_0-1)}\right), \left(c_{1,0}, c_{1,1}, c_{1,2}, \dots, c_{1,(M_1-1)}\right), \dots, \left(c_{(N-1),0}, c_{(N-1),1}, .., c_{(N-1),(M_{N-1}-1)}\right)\right)$$

, where M_i is the total number of total number of integers of the constraint in the i^{th} column.

An example of a solved Nonogram is shown in fig. 2. Note that there are several forms to represent the same constraint, such as `'(3 1)` and `'(3 0 1)`. For simplicity, all zeros are omitted from the constraints unless when the whole line of blocks contains only spaces, the list is `'(0)`.

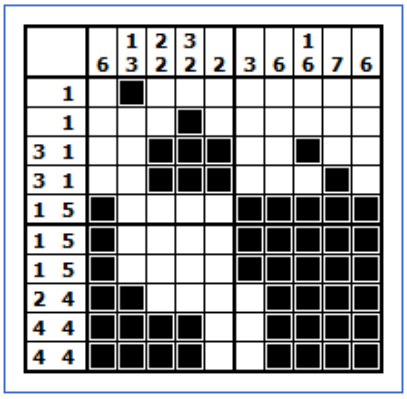

Figure 2: An example of a Nonogram with a solution.

To familiarize with the game, you are strongly encouraged to solve some puzzles manually. There are some solving methods found on the web and you may program them in your solver. Although there are cases required deeper recursion to discover contradictions, we will not check this in your program. An example of such case is shown in fig. 3.

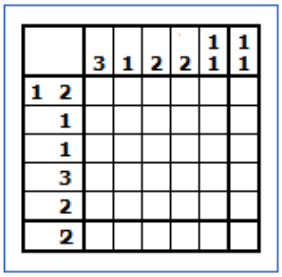

Figure 3: An example of a Nonogram with multiple solutions and required deeper recursion.

Inputs

Using fig.2 as an example, a **list of lists R** representing the constraints of the rows, for example:

```
'((1) (1) (3 1) (3 1) (1 5) (1 5) (1 5) (2 4) (4 4) (4 4))
```

A **list of lists C** representing the constraints of the columns, for example:

```
'((6) (1 3) (2 2) (3 2) (2) (3) (6) (1 6) (7) (6))
```

Output

For each test case, output a **list of lists A** representing the correct solution, for example:

```
((s b s s s s s s s) (s s s b s s s s s s) (s s b b b s s b s s) (s
s b b b s s s b s) (b s s s s b b b b b) (b s s s s b b b b b) (b s s
s s b b b b b) (b b s s s s b b b b) (b b b b s s b b b b) (b b b b s
s b b b b))
```

Submission

You only need to provide a Lisp file with the specific solver function **nonogram-solver** which can be invoked as follows:

(nonogram-solver R C)

,which reads the row constraints R and column constraints C, and finally returns one of the correct solutions. That means you do not have to worry about the I/O of the inputs or outputs. You may implement other functions in your program and solve the puzzle using any of the searching methods but not enumerating of all the states.

A single Lisp file, named **StudentID_nonogram.lisp**, should be prepared and **uploaded to the submission system**.

Grading

Since this is an optional assignment, the grading is loose: your raw mark is the percentage of the test cases (about 10) solved by your program. Make ensure that your program can return the results for all the test cases in 1 minute in our machine. If your program can finish any 30x30 puzzle for about 1 second running on a machine in the CSE lab, it will meet the time requirement. If you submit your assignment late, it is subject to the following penalty scheme.

Number of Days Late	Marks Deduction
1	10%
2	30%
3	60%
4 or more	100%

Suggested Solving Strategy

You may have your own solution.

1. Implement a pattern generator.
    - Input: a constraint, cell states of a line
    - Output: all the potential combinations with
    - Example: (pattgen (1 3) (b u u u u u)) → ((b w b b b w) (b w w b b b))
2. Combine the generated patterns to make progress.
    - Input: patterns (any number of them) for a constraint
    - Output: combined decision on the patterns
    - Example: (combdecision ((b w b b b w) (b w w b b b)))→ (b w u b b u)
3. Fill the combined decision to the board.
4. Repeat the above steps until all the blocks are filled.
5. Guess the states and check for contradictions if there remains unfilled cells.

Additional information

Nonogram – Wikipedia, the free encyclopedia:
http://en.wikipedia.org/wiki/Nonogram

Play Nonogram:
http://www.puzzle-nonograms.com/?size=1

Constraint programming and propagation:
http://sadaf2605.appspot.com/blog/284003/Constraint%20programming%20and%20propagation#sthash.rx8JGeWD.dpbs

Guide to Constraint Programming:
http://kti.ms.mff.cuni.cz/~bartak/constraints/

Have fun!