ITASoftware Hiring Puzzle: Strawberry Fields

Strawberry Fields
Strawberries are growing in a rectangular field of length and width at most 50. You want to build greenhouses to enclose the strawberries. Greenhouses are rectangular, axis-aligned with the field (i.e., not diagonal), and may not overlap. The cost of each greenhouse is $10 plus $1 per unit of area covered. Write a program that chooses the best number of greenhouses to build, and their locations, so as to enclose all the strawberries as cheaply as possible. Heuristic solutions that may not always produce the lowest possible cost will be accepted: seek a reasonable tradeoff of efficiency and optimality. Your program must read a small integer 1 ≤ N ≤ 10 representing the maximum number of greenhouses to consider, and a matrix representation of the field, in which the '@' symbol represents a strawberry. Output must be a copy of the original matrix with letters used to represent greenhouses, preceded by the covering's cost. Here is an example input-output pair:

Input		Output

`4`
`..@@@@@..............`
`.`
`..@@@@@@........@@@..`
`.`
`.....@@@@@......@@@..`
`.`
`.......@@@@@@@@@@@@..`
`.`
`.........@@@@@.......`
`.`
`.........@@@@@.......`
`.`
		`90`
		`..AAAAAAAA...........`
		`.`
		`..AAAAAAAA....CCCCC..`
		`.`
		`..AAAAAAAA....CCCCC..`
		`.`
		`.......BBBBBBBCCCCC..`
		`.`
		`.......BBBBBBB.......`
		`.`
		`.......BBBBBBB.......`
		`.`

In this example, the solution cost of $90 is computed as (10+8*3) + (10+7*3) + (10+5*3).

Run your program on the 9 sample inputs found in this file and report the total cost of the 9 solutions found by your program, as well as each individual solution.

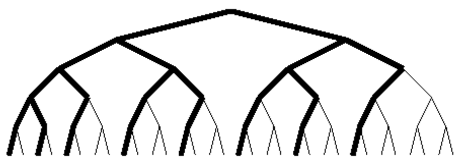
URL: http://www.itasoftware.com/careers/work-at-ita/hiring-puzzles.html

Note: All Text in `courier new` is code.

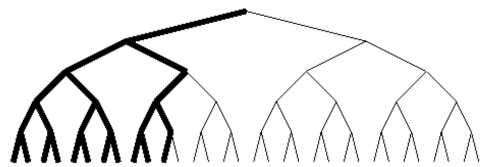
Description:
To solve the strawberry fields problem I chose to implement a solution based on constraint propagation and heuristic backtracking search in the (constraint-logic) programming language ECLiPSe. My solution is based on a representation of the greenhouses as a pair of coordinates

(top-left, bottom-right) ranging over the entire (1..W,1..H) field. By constraining these coordinates to be valid solutions a search space is obtained in which it is possible to find reasonable solutions in a reasonable amount of time.

The search algorithm used amounts to credit search, always expanding the most-constrained variable with the smallest domain size first, followed by a bounded backtracking search over 8 levels. Intuitively i’m using credit search to spread out over the search space (leaning slightly to the left), and finish up using a bounded backtracking search to not miss any slightly better solutions close by. The exact values were chosen empirically (using the speed predicate).



credit(16)



bbs(10)

Results:

Problem	Solution Cost / Running Time
100	50 / 2.62s
101	62 / 5.60 s
102	99 / 8.03s
103	262 / 50s
104	309 / 37.41s
105	318 / 25.60s
106	426 / 33.27s
107	297 / 118s
108	297 / 118s

Total Cost: 2120

Note: the ordering of the problems corresponds to the ordering in the problems file but I had to number them starting from 100 instead of 0 to avoid conflicts.

Instructions:

To run the program the ECLiPSe runtime is needed. It can be obtained (for free) from here. After installation, start the doseclipse tool (or equivalent command line executable) in the directory containing the *rects.ecl* and *problems.ecl* file and issue the command (note the ‘.’ at the end):

```
compile(['problems.ecl', 'rects.ecl']).
```

(It should answer: Yes) after which the program can be used.

The solve predicate takes a number of arguments: Nr - which is the number of a problem (from problems.ecl, first argument of the problem predicate), Rects and Total which are the resulting list of rectangles (greenhouses) and the total cost of the solution found. The solve predicates then requires 3 arguments determining the search method to use: Selection, Choice, Method (see here for possible values for these parameters) and these should be set to: most_constrained, indomain_interval, credit(1024, bbs(8)). And lastly there are 2 parameters that made development easier, namely: Optimal and Timeout which indicate the optimal value for the current problem and give the maximum amount of time the search is allowed to run, respectively.

Putting it all together, running the following query will search the example problem from the problem description on page 1:

```
solve(11, X, C, most_constrained, indomain_interval,
credit(1024,bbs(8)), 1, 10000).
```

rects.ecl

```
:- lib(fd).
:- lib(fd_search).
:- lib(fd_global).
:- import problems.

% solve(11,R,C, anti_first_fail, indomain_split, bbs(W + H + Max), 1,30).

% a strawberry should be covered by exactly 1 greenhouse
covered(X, Y, Rects, Max) :-
	dim(Cover, [Max]),
	Cover[1..Max] :: 0..1,
	( for(I, 1, Max), param(X, Y, Rects, Cover) do
		((Rects[I,1] #<= X) #/\ (Rects[I,2] #<= Y) #/\ (Rects[I,3] #> X) #/\
(Rects[I,4] #> Y)) #<=> (Cover[I] #= 1)
	),
	occurrences(1, Cover, 1).

% count occurences of Ex in List, return in Count
count(List, Ex, Count) :-
	( foreach(El, List), param(Ex), fromto(0, S1, S2, Count) do
		( El = Ex -> S2 is S1 + 1 ; S2 is S1 )
	).

% count occurences of Ex in list-of-lists List, return in Count
count_all(List, Ex, Count) :-
	( foreach(El, List), param(Ex), fromto(0, S1, S2, Count) do
		count(El, Ex, C),
		S2 is S1 + C
```

```
	).

% rectangle R1 shouldn't overlap with rectangle R2
not_overlap(R1, R2) :-
	X11 is R1[1], Y11 is R1[2], X12 is R1[3], Y12 is R1[4],
	X21 is R2[1], Y21 is R2[2], X22 is R2[3], Y22 is R2[4],
	(Y22 #<= Y11) #\/ (X21 #>= X12) #\/ (X22 #<= X11) #\/ (Y21 #>= Y12).

% each rectangle should not overlap with each other rectangle
not_overlapping(Rects, Max) :-
	( for(I,1,Max), param(Rects, Max) do
		( for(J,1,Max), param(I, Rects) do
			( I \= J ->
				R1 is Rects[I],
				R2 is Rects[J],
				not_overlap(R1,R2)
			;
				true
			)
		)
	).

% speed tests
speed(Nr) :-
	Methods = [anti_first_fail, smallest, first_fail, largest, occurence,
most_constrained, input_order],
	Choices = [indomain, indomain_min, indomain_split, indomain_interval],
	( foreach(Selection, Methods), param(Choices, Nr) do
		( foreach(Choice, Choices), param(Selection, Nr) do
			writeln([Choice, Selection]),
			profile(solve(Nr, _, _, Selection, Choice, 91, 60))
		)
	).

% main solve routine
solve(Nr, Rects, Total, Selection, Choice, Method, Optimal, Timeout) :-
	problem(Nr, Max, W, H, Field),
	%prettyprint(Field, Max),
	% setup constraints for the greenhouses: each greenhouse represented by the
topleft and bottomright corners as a list [X1, Y1, X2, Y2]
	dim(Rects, [Max, 4]),
	dim(Costs, [Max]),
	( for(K,1,Max), param(Rects, Costs, W, H, Max, Field) do
		% top left: X1, Y1
		Rects[K,1] :: 0..W,
		Rects[K,2] :: 0..H,
		% bottom right: X2, Y2
		Rects[K,3] :: 0..W + 1,
		Rects[K,4] :: 0..H + 1,
		% both coordinates of the greenhouse are 0 or >= 1
		((Rects[K,1] #>= 1) #/\ (Rects[K,2] #>= 1) #/\ (Rects[K,3] #>= 1) #/\
(Rects[K,4] #>= 1)) #\/ ((Rects[K,1] #= 0) #/\ (Rects[K,2] #= 0) #/\ (Rects[K,3] #= 0)
#/\ (Rects[K,4] #= 0)),
		% each rect should have bottom right > top left
		((Rects[K,3] #> Rects[K,1]) #/\ (Rects[K,4] #> Rects[K,2])) #\/
((Rects[K,3] #= 0) #/\ (Rects[K,1] #= 0) #/\ (Rects[K,4] #= 0) #/\ (Rects[K,2] #= 0)),
		% costs of each greenhouse
		((Rects[K,3] - Rects[K,1]) #> 0) #/\ ((Rects[K,4] - Rects[K,2]) #> 0)
#<=> (Costs[K] #<=> 10 + (Rects[K,3] - Rects[K,1]) * (Rects[K,4] - Rects[K,2])),
		((Rects[K,3] - Rects[K,1]) #= 0) #/\ ((Rects[K,4] - Rects[K,2]) #= 0)
#<=> (Costs[K] #<=> 0),
		% all strawberries must be covered by the greenhouse
		( foreach(Row, Field), param(Rects, Max, W), count(J,1,H) do
```

```
                    ( foreach(Cell, Row), param(Rects, J, Max), count(I,1,W) do
                        ( Cell = 1 -> covered(I, J, Rects, Max) ; true )
                    )
                )
        ),
        % greenhouses shouldn't overlap
        not_overlapping(Rects, Max),
        % setup the search
        collection_to_list(Rects, RectsList),
        % setup the minimization objective
        collection_to_list(Costs, CostsList),
        sumlist(CostsList, Total),
        % do the actual search
        minimize(search(RectsList, 0, Selection, Choice, Method, []), Total, Optimal,
1000, 0, Timeout),
        % alternative calls
        %minimize(ilabeling(RectsList), Total, Optimal, 1000, 0, Timeout),
        %minimize(plabeling(RectsList), Total),
        % show the result as required
        show(Rects, Field, Total, Max, W, H).

% test, parallel labelings
plabeling([]).
plabeling([Var|Rest]) :-
        par_indomain(Var),
        labeling(Rest).

% instrumented labeling, counts nr of backtracks
ilabeling(AllVars) :-
        init_backtracks,
        ( foreach(Var, AllVars) do
                count_backtracks, % insert this before choice!
                indomain(Var)
        ),
        get_backtracks(B),
        printf("Solution found after %d backtracks%n", [B]).

:- local variable(backtracks), variable(deep_fail).

init_backtracks :-
        setval(backtracks,0).

get_backtracks(B) :-
        getval(backtracks,B).

count_backtracks :-
        setval(deep_fail,false).

count_backtracks :-
        getval(deep_fail,false), % may fail
        setval(deep_fail,true),
        incval(backtracks),
        fail.
```

problems.ecl

```
:- module(problems).
:- export problem/5.
:- export answer/3.
:- export show/6.
:- export prettyprint/2.
```

```
% show an array of rects as letters on the field as required by the assignment
show(Rects, Field, Total, Max, W, H) :-
	Letters = []('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M'),
	writeln(Total),
	( foreach(Row, Field), param(Rects, Letters, W, Max), count(J,1,H) do
		( foreach(_, Row), param(Rects, Letters, J, Max), count(I,1,W) do
			( for(K,1,Max), param(Rects, Letters, I, J, Output) do
				Letter is Letters[K],
				Rect is Rects[K],
				( ((Rect[1] =< I), (Rect[2] =< J), (Rect[3] > I), (Rect[4]
> J)) ->
					write(Letter),
					Output is 1
				;
					true
				)
			),
			( integer(Output) -> true ; write('.') )
		),
		writeln('')
	),
	writeln('').

prettyprint(Field, Max) :-
	writeln(Max),
	( foreach(Row, Field) do
		( foreach(Cell, Row) do
			( Cell = 1 -> write("@") ; write(".") )
		),
		writeln('')
	),
	writeln('').

%        Nr  Max,  W,  H, Field
problem(  1,   1,  1,  1, [
	[0]
]).

%        Nr  Max,  W,  H, Field
problem(  2,   1,  1,  1, [
	[1]
]).

%        Nr  Max,  W,  H, Field
problem(  3,   2,  2,  1, [
	[0,0]
]).

%        Nr  Max,  W,  H, Field
problem(  4,   2,  2,  1, [
	[1,0]
]).

%        Nr  Max,  W,  H, Field
problem(  5,   2,  2,  1, [
	[0,1]
]).

%        Nr  Max,  W,  H, Field
problem(  6,   2,  2,  1, [
	[1,1]
]).
```

```
%        Nr  Max,  W,  H, Field
problem(  7,   2,  2,  2, [
       [1,1],
       [1,1]
]).

%        Nr  Max,  W,  H, Field
problem(  8,   2,  2,  2, [
       [1,0],
       [0,1]
]).

problem(9, 4, 2, 15, [
       [0,1],
       [0,0],
       [0,0],
       [0,0],
       [0,0],
       [0,0],
       [0,0],
       [1,1],
       [0,0],
       [0,0],
       [0,0],
       [0,0],
       [0,0],
       [0,0],
       [1,0]
]).

problem(10, 3, 13, 10, [
       [0,0,0,0,0,0,0,0,0,0,0,0,0],
       [0,0,0,0,0,0,0,0,0,0,0,0,0],
       [0,0,0,0,0,0,0,0,0,0,0,0,0],
       [0,0,0,1,1,1,1,0,0,0,0,0,0],
       [0,0,0,1,1,1,1,0,0,0,0,0,0],
       [0,0,0,1,1,1,1,0,0,0,0,0,0],
       [0,0,0,0,0,0,0,1,1,1,0,0,0],
       [0,0,0,0,0,0,0,1,1,1,0,0,0],
       [0,0,0,0,0,0,0,1,1,1,0,0,0],
       [0,0,0,0,0,0,0,0,0,0,0,0,0]
]).

problem(11, 4, 22, 6, [
       [0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
       [0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,0,0,0],
       [0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0],
       [0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0],
       [0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0],
       [0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0]
]).
% optimal solution: 3, $90 (10+8*3) + (10+7*3) + (10+5*3)

problem(12, 4, 13, 9, [
       [0,0,0,0,0,0,0,0,0,0,0,0,0],
       [0,0,1,0,1,0,1,0,0,0,0,0,0],
       [0,0,0,1,0,1,0,1,0,0,0,0,0],
       [0,0,1,0,1,0,1,0,0,0,0,0,0],
       [0,0,1,0,1,0,1,0,0,0,0,0,0],
       [0,0,0,1,0,1,0,1,0,0,0,0,0],
       [0,0,0,0,1,0,1,0,0,0,0,0,0],
       [0,0,0,0,0,0,0,0,0,0,1,1,1],
       [0,0,0,0,0,0,0,0,0,0,1,1,1]
```

```
]).


% assigment

problem(100, 3, 13, 10, [
        [0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,1,1,1,1,0,0,0,0,0,0],
        [0,0,0,1,1,1,1,0,0,0,0,0,0],
        [0,0,0,1,1,1,1,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,1,1,1,0,0,0],
        [0,0,0,0,0,0,0,1,1,1,0,0,0],
        [0,0,0,0,0,0,0,1,1,1,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0]
]).

problem(101, 4, 13, 9, [
        [0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,0,1,0,1,0,0,0,0,0,0],
        [0,0,0,1,0,1,0,1,0,0,0,0,0],
        [0,0,1,0,1,0,1,0,0,0,0,0,0],
        [0,0,1,0,1,0,1,0,0,0,0,0,0],
        [0,0,0,1,0,1,0,1,0,0,0,0,0],
        [0,0,0,0,1,0,1,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,1,1,1],
        [0,0,0,0,0,0,0,0,0,0,1,1,1]
]).

problem(102, 4, 13, 9, [
        [0,0,0,0,0,0,0,0,0,1,0,0,0],
        [0,0,0,0,0,0,0,0,0,1,0,0,0],
        [1,1,1,1,1,1,1,1,1,1,0,0,0],
        [0,0,1,0,0,0,0,0,0,1,0,0,0],
        [0,0,1,0,0,0,0,0,0,1,0,0,0],
        [0,0,1,0,0,0,0,0,0,1,0,0,0],
        [0,0,1,1,1,1,1,1,1,1,1,1,1],
        [0,0,1,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,0,0,0,0,0,0,0,0,0,0]
]).

problem(103, 7, 25, 14, [
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0],
        [0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0],
        [0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
        [0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
        [0,0,1,1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,0,1,0],
        [0,0,1,1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,0,0,0],
        [0,0,1,1,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0],
        [0,0,1,1,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0],
        [0,0,1,1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,0,0,0],
        [0,0,1,1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,0,0,0],
        [0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
        [0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
]).

problem(104, 6, 25, 16, [
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
```

```
        [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,1,0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,0,0,0,0],
        [0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,1,0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,0,0,0,0],
        [0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
]).

problem(105, 5, 40, 18, [
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
]
]).

problem(106, 8, 40, 18, [
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
        [0,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
```

```
	[0,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
]
]).

problem(107, 10, 40, 19, [
	[1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
```

```
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
]
]).

problem(108, 10, 40, 25, [
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
],
	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
]
]).

% answers
```

```
answer(10, 2, 41).
answer(11, 2, 90).

answer(100, 2, 41).
answer(101, 2, 62).
```