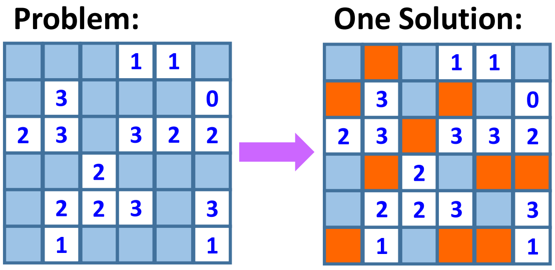
Introduction to AI   NCTU Spr2020   Programming Assignment #2   Due 5/1/2020

In this assignment, your task is to find solutions of Minesweeper problems using Backtrack Search. Here we will consider a type of Minesweeper problems: In a board where part of the cells are already marked as hints, your task is to find an arrangement of mines in the other cells such that the hints are correct. An example is given below:



🟦	🟦	🟦	1	1	🟦
🟦	3	🟦	🟦	🟦	0
2	3	🟦	3	2	2
🟦	🟦	2	🟦	🟦	🟦
🟦	2	2	3	🟦	3
🟦	1	🟦	🟦	🟦	1

🟦	🟧	🟦	1	1	🟦
🟧	3	🟦	🟧	🟦	0
2	3	🟧	3	3	2
🟦	🟧	2	🟦	🟧	🟧
🟦	2	2	3	🟦	3
🟧	1	🟦	🟧	🟧	1

The formulation of this task as a constraint satisfaction problem will be like:

- The cells are indexed starting from the top-left cell at (0,0). The hint (0~8) at cell (x,y) is h(x,y).
- The value assigned to cell (x,y) is n(x,y), which can only be zero or one, representing whether it contains a mine.
- Each non-hint cell represents a variable. The initial domains are {0,1}.
- Each hint gives a constraint in the form of an equation. For example:

  $n(0,0) + n(1,0) + n(2,0) + n(0,1) + n(2,1) + n(2,2) = h(1,1) = 3.$
- There is a global constraint that the summation of all the variables equals the total number of mines.

For problems given in this assignment, we will use 6x6 boards with 16 hints (17 constraints), 20 variables, and 10 mines in total. Note: Solutions may not be unique.

Notes about implementation:

- Implement backtrack search with a stack. Each node contains (1) the list of all the variable+value assignments made from the root to that node, and (2) the domains of all the unassigned variables at that node. Initialize the stack with a node containing an empty assignment list and initial domains of all the variables.
- When popping a node from the stack:
  - ➢ (Optional) Do the forward checking to update the domains of unassigned variables.
  - ➢ Generate the child nodes from all the variable+value combinations applicable after the current node. Insert them to the stack (in increasing order of preference, if any heuristic is used).

Consistency check for forward checking:

- For each constraint, we compute the lower and upper bounds of the sum of the variables given their domains, and compare the bounds with the hint.
- If the lower bound is larger than the hint, the constraint cannot be satisfied.
- If the upper bound is smaller than the hint, the constraint cannot be satisfied.
- If the lower bound equals the hint, the domains of all the unassigned variable in the constraint should be limited to their respective minimal values.
- If the upper bound equals the hint, the domains of all the unassigned variable in the constraint should be limited to their respective maximal values.
- For other cases, the domains are unchanged.

Heuristics

- MRV: This means that variables whose domains have become singletons are assigned first.

- Degree heuristic: How many constraints contain this variable?
- LCV (only for variables whose domains are still {0,1}): You need to try both values using the consistency checks mentioned above to see how they affect the domains of the other variables.

Sample problems are given below. You can also generate your own problems with different board sizes and/or numbers of mines. You can try to compare the difficulties statistically by randomly generating many problems. What is the largest board your program can solve in reasonable time?

The use of forward checking and heuristics should be made optional, so that you can compare the performance (number of expanded nodes until solution) with and without them.

You submission is a report file in PDF format. The report (maximum 5 pages single-spaced) should describe your experiments and results, especially the comparison between the different algorithms. In your report, also include a section describing your observations, interpretations, things you have learned, remaining questions, and ideas of future investigation. Include your program code as an appendix (not counting toward the 5-page limit), starting from a separate page. You can use C/C++, Java, Python, or MATLAB to write your program. In general, the TAs will not actually compile or run your programs. The code listing is used to understand your thoughts during your implementation, and to find problems if your results look strange. Therefore, the code listing should be well-organized and contain comments that help the readers understand your code; this will also affect your grade.

The submission is to be through E3. Late submission is accepted for up to a week, with a 5% deduction per day.

Inputs: (board size, #mines, board (-1: variable))
6 6 10 -1 -1 -1 1 1 -1 -1 3 -1 -1 -1 0 2 3 -1 3 3 2 -1 -1 2 -1 -1 -1 -1 2 2 3 -1 3 -1 1 -1 -1 -1 1
6 6 10 -1 -1 -1 1 1 1 3 4 -1 2 -1 -1 2 -1 -1 -1 -1 -1 -1 -1 2 2 -1 2 1 2 -1 -1 1 -1 -1 1 -1 1 0 -1
6 6 10 -1 -1 -1 -1 -1 -1 -1 2 2 2 3 -1 -1 2 0 0 2 -1 -1 2 0 0 2 -1 -1 3 2 2 2 -1 -1 -1 -1 -1 -1 -1
6 6 10 -1 1 -1 1 1 -1 2 2 3 -1 -1 1 -1 -1 5 -1 5 -1 2 -1 5 -1 -1 -1 -1 2 -1 -1 3 -1 -1 -1 1 1 -1 0
Sample solutions:

🟦	🟧	🟦	1	1	🟦
🟧	3	🟦	🟧	🟦	0
2	3	🟧	3	3	2
🟦	🟧	2	🟦	🟧	🟧
🟦	2	2	3	🟦	3
🟧	1	🟦	🟧	🟧	1

🟧	🟧	🟦	1	1	1
3	4	🟧	2	🟦	🟧
2	🟧	🟦	🟦	🟧	🟧
🟦	🟧	2	2	🟧	2
1	2	🟧	🟦	1	🟦
🟦	1	🟦	1	0	🟦

🟦	🟧	🟦	🟧	🟧	🟦
🟦	2	2	2	3	🟦
🟧	2	0	0	2	🟧
🟧	2	0	0	2	🟧
🟦	3	2	2	2	🟦
🟦	🟧	🟧	🟦	🟧	🟦

🟧	1	🟦	1	1	🟦
2	2	3	🟧	🟦	1
🟦	🟧	5	🟧	5	🟧
2	🟧	5	🟧	🟦	🟧
🟦	2	🟦	🟧	3	🟦
🟧	🟦	1	1	🟦	0