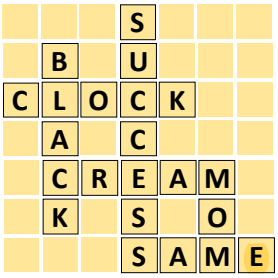
Introduction to AI NCTU Spr2019 Programming Assignment #1 Due 4/6/2019

The objective of this assignment is for you to practice with solving constraint satisfaction problems with searching (the backtracking search in the lecture slides).

The problems to be solved in this assignment are crossword puzzles without hints. Any words from a given word list can be used in the solution as long as the constraints are satisfied. To the right is an example crossword puzzle with a possible solution. (Note: Since we do not provide hints, the solution is not unique.)



A puzzle is specified in the format of (start_x, start_y, length, direction) for each word. For example, for the sample puzzle here, the specification is

```
0 2 5 A   1 1 5 D   3 0 7 D   1 4 5 A   3 6 4 A   5 4 3 D
```

The origin (0,0) is at the top-left corner. For directions, "A" means "across" and "D" means "down".

To start, each word in the puzzle is a variable (6 variables here), and the provided word list is the initial domain. Consider the length of each word to be its unary constraint. Each block where two words intersect gives a binary constraint (there are 7 here).

The following are some notes about implementation:

- Before you start the search, apply the unary constraints so that the domain of each variable contains only words of the correct length. Optionally, you can use AC-3 to further trim the domains.
- Each node in the search tree needs to maintain the following information:
  - The variable+value combination to be assigned at this node (only for non-root nodes).
  - The list of assignments between the root and the current node.
  - The domains of all the variables. (This is required only if you want to apply forward checking or AC-3 after each assignment.)
- Use a stack for the backtracking search. Initialize the stack with only the root node with an empty list of assignment and initial variable domains.
- When popping a node from the stack (node expansion), do the following:
  - Add the variable+value combination of this node to its list of assignments, and set the domain of the assigned variable to contain only the assigned value.
  - (If forward checking / AC-3 is used) Update the domains of the unassigned variables, if any.
  - Consistency check: 1. Do the current list of assignments satisfy all the constraints? 2. (If forward checking / AC-3 is used) are all the variables still have nonempty domains?
  - If consistency check returns TRUE and all the variables are already assigned, a solution is found, and the program returns with the solution.
  - If consistency check returns TRUE and there are still unassigned variables:
    - Generate a list of child nodes, one for each variable+value combination of the unassigned variables.
    - Copy the list of assignments and variable domains of the current node to the child nodes.
    - Push the child nodes into the stack. (If the MRV/degree/LCV heuristics are used, you should push the child nodes in the order of increasing preference.)
- If the stack becomes empty, there is no solution, and the program returns failure.

A file containing about 3000 English words is provided. Also the file `puzzle.txt` is provided, where each line represents a puzzle.

You submission is a report file in Word or PDF format. The report (maximum 5 pages single-spaced) should describe your experiments and results, especially the comparison between the different algorithms. Several topics that you can experiment with include

- Numbers of visited nodes for the different puzzles.
- How the numbers of visited nodes change with or without any of the heuristics.
- How the numbers of visited nodes change with or without the initial run of AC-3.
- How the numbers of visited nodes change with or without forward checking or AC-3 during the search.
- Is it possible to do a complete search (not stopping when solutions are found) and determine the numbers of valid solutions for the different puzzles?
- Can you somewhat randomize the search so that each run can generate a different solution?

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.

Some example solutions to the provided puzzles: