Project
Zebra Puzzle

Thursday 31st March, 2016

The zebra puzzle is a well-known logic puzzle. Many versions of the puzzle exist, including a version published in Life International magazine on December 17, 1962 (slightly modified here – note that there are 5 houses):

1. The Englishman lives in the red house.
2. The Spaniard owns the dog.
3. Coffee is drunk in the green house.
4. The Ukrainian drinks tea.
5. The green house is immediately to the right of the ivory house.
6. The Snakes and Ladders player owns snails.
7. Cluedo is played in the yellow house.
8. Milk is drunk in the middle house.
9. The Norwegian lives in the first house.
10. The man who plays Pictionary lives in the house next to the man with the fox.
11. Cluedo is played in the house next to the house where the horse is kept.
12. The Travel the World player drinks orange juice.
13. The Japanese plays Backgammon.
14. The Norwegian lives next to the blue house.

Now, who drinks water? Who owns the zebra?
In the interest of clarity, it must be added that each of the five houses is painted a different color, and their inhabitants are of different national extractions, own different pets, drink different beverages and play different board games. One other thing: in statement 5, *right* means *your* right."

To solve this problem, we number houses from 1 (leftmost) to 5 (rightmost) and we consider 25 variables (the unknown values):

- 5 nationalities: English, Spaniard, Ukrainian, Norwegian, Japanese
- 5 colours:
- 5 pets:
- 5 board games: Snakes and Ladders, Cluedo, Pictionary, Travel the World, Backgammon.
- 5 beverage:

Each variable has a *domain*, i.e., the set of possible values it can take. Here at the start of the puzzle every variable has a full initial domain: $\{1, 2, 3, 4, 5\}$, corresponding to the house numbers.
Our approach consists in transforming every hints of the problem into *constraints*, such as:

1. $Englishman = red$
2. $Spaniard = dog$

3. $coffee = green$
4. $Ukrainian = tea$
5. $green = ivory + 1$
6. $snakes\ and\ ladders = snails$
7. $Cluedo = yellow$
8. $milk = 3$
9. $Norwegian = 1$
10. $|Pictionary - fox| = 1$
11. $|Cluedo - horse| = 1$
12. $Travel\ the\ World = orange\ juice$
13. $Japanese = Backgammon$
14. $|Norwegian - blue| = 1$
15. as well as 50 difference constraints to specify that for each set of variables the value of each variable is different from the others (e.g., $milk \neq tea$ indicates that milk and tea drinkers are two different persons in two different houses).

Solving the puzzle consists in assigning a house number to every variable. The algorithm (a.k.a., the solver) will use two classical functions:

- *domain reduction*: consists in removing some of the values in a domain. For instance the constraint $Norwegian = 1$ reduces the domain of the variable $Norwegian$ to $\{1\}$.
- *domain splitting*: when no domain reduction is possible, and at least one domain has at least 2 values, then we can split this domain in half and consider two subproblems of the general problem. You can consider this process iterative or recursive (see below).

1 Representing Domains and Variables

- every domain can be created, destroyed, printed, split in half, compared with another domain (using $==$ and $!=$). Other methods that a domain should have are: $is\_empty()$ and $is\_reduced\_to\_only\_one\_value()$. We also need a function, defined over the set of domains, $largest\_domain$ which returns the largest domain.
- each variable has to be defined with a name, a unique identifier and a domain. We want to be able to print the name and the identifier of a variable, and get/set its domain.
- the set of all variables is another data type (e.g., it can be represented using an ADT sequence, possibly sorted on the variables names).
- design and implement all these using data structures/classes and test them on the puzzle.

2 Representing Constraints

A constraint contains a set of variables. A constraint has a satisfaction method, i.e., a boolean function testing whether the constraint is satisfied for the current values of its variables, and a domain splitting function.

1. The `Constraint` class is the basic class for constraints. The set of variables from a constraint is managed at this level. A constraint can be created, deleted and printed. Moreover the `Constraint` class implements two methods: $is\_satisfied$ (return a boolean value) and $reduction$ which can reduce the domain of its variables (return $False$ if an empty domain results from the reduction, $True$ otherwise).

2. `Constraint_set` contains all constraints of our problem.

3. For our puzzle, we need 4 types of constraints (use class inheritance from `Constraint`):

   - `Constraint_equality_var_var`: for $\alpha = \beta$ constraints where α and β are variables. This constraint is satisfied if the two variables' domains have at least one value in common. The reduction consists in removing all values that are not shared. E.g., a constraint $fisherman's friend = stapler$ with the domain of the variable $fisherman's friend$ being $\{1, 3, 5, 7, 8\}$ and the domain of the variable $stapler$ being $\{2, 3, 4, 7\}$ would result in both domains being $\{3, 7\}$, i.e., composed only of the values that are shared.
   - `Constraint_equality_var_cons`: for $\alpha = c$ constraints where α is a variable and c a constant (integer). It is satisfied if c is in α's domain. The reduction consists in removing all values in α but c. E.g., a constraint $fisherman's friend = 3$ with the domain of the variable $fisherman's friend$ being $\{3, 7\}$ will update $fisherman's friend$'s domain to $\{3\}$ *[I guess you start seeing that doing iteratively this constraint and the one seen in the previous item above will also modify the domain of the variable stapler as well:* $\{3\}$*.]*
   - `Constraint_equality_var_plus_cons`: for $\alpha = \beta + c$ constraints where α and β are variables and c a constant (integer). This constraint is satisfied if there exists a in α's domain and b in β's domain such that $a = b + c$. α's domain is reduced by removing all values a such that there is no b in β such that $a = b + c$. β's domain is reduced by removing all values b such that there is no a in α such that $b = a - c$. E.g., $snowman = Bowie + 1$ with the domain of the variable $snowman$ being $\{1, 2, 3, 5, 7, 8\}$ and the domain of the variable $Bowie$ being $\{1, 2, 3, 4, 7\}$ would result in $snowman$'s domain being updated to $\{2, 3, 5, 8\}$ (1 and 7 cannot be in $snowman$'s domain because there is no 0 and 6 in $Bowie$'s domain) and $Bowie$'s domain being updated to $\{1, 2, 4, 7\}$.
   - `Constraint_difference_var_var`: for $\alpha \neq \beta$ constraints where α and β are variables. This constraint is not satisfied if both domains are reduced to a single identical value. This is how α's domain is reduced: if β's domain is reduced to a single value b then b is removed from α's domain. Likewise, if α's domain is reduced to a single value a then a is removed from β's domain.

4. A good option here would be to use method override in each constraint type/class so that you use the same methods with different implementations.

5. Create the class `Problem` as a set of constraints and a set of variables (implement at least constructor and print methods).

6. Design and implement all these classes/types and test them on the puzzle.

3 Domains Reduction

Each constraint can reduce the domains of its variables – but obviously we need to propagate those reductions iteratively. For instance constraint $Norwegian = 1$ reduces the domain of variable $Norwegian$ (to $\{1\}$) which in turns will be used in the constraint $|Norwegian - blue| = 1$ to reduce the domain of $blue$ (its domain being now $\{2\}$). Etc.

Design and implement an algorithm with one loop that iterates over all constraints and reduces the domains of their variables until nothing happens or an empty domain is found (this problem is unsatisfiable).

4 Domains Splitting

Often the algorithm for domains reduction presented above is not sufficient to find a solution (a solution corresponds to all variables' domains reduced to one single value). This is especially the case for problems with several solutions (not only one).We then need a domain splitting algorithm, i.e., a recursive algorithm that uses a binary tree or a stack where each node of the tree/element in the stack is a subproblem of the initial problem.

The root of the tree or the first element in the stack, is the initial problem. In each recursive call, the base cases are (i) an empty domain is found (which means the current subproblem is unsatisfiable) (ii) every domain has only one value (which corresponds to a solution to the subproblem). Otherwise, the recursion consists in picking one of the domains with at least 2 values and splitting it to create two subproblems: it is like taking two hypotheses and

checking whether they lead to a solution (every domain has only one value) or to an impossible stage (if an empty domain is generated).

[Optional] Try different strategies for domain splitting (the domain with the smallest number of values, or the highest etc.). Would it be better to split in more than two?

5 General Instructions

- You are encouraged to collaborate with your peers on this project, but all written work must be your own. In particular we expect you to be able to explain every aspect of your solution.
- To this end, we will run a presentation session when we'll ask you questions about your solution and so on.
- We ask you to hand in an archive (zip or tar.gz) of your solution: commented Python code, README.txt file describing how to run your program, a 5-10 page pdf report of your work (no need to include code in it).
- Due date: 26/04/2016