

## CS2402: Lab assignment #2

Due date: October 4th, at 5pm

Send your report (YourName.doc or YourName.pdf) and a zipped file of your program (YourName.tar.gz) to your TA (Carlos Acosta, ceacosta@utep.edu) and cc to me (mceberio@cs.utep.edu).

### Topics covered in this lab assignment

stacks, trees, depth-first search traversal of search tree, constraint solving

### Objective of this lab assignment

- understand the workings of stacks
- manipulate stacks in a complex program
- review trees, use the concept of tree as a way to traverse the search space, using a stack to simulate the tree traversal
- understand the DFS method
- understand how constraint propagation can help to speed-up problem-solving
- run appropriate experiments (and theoretical analysis if you think it can help) to point out the difference between two methods applied to the same problem

### Description of the problem

Let us consider the following problem of building walls in a museum, under a couple of constraints:

The museum is a labyrinth: you can visit all rooms, reach all room of the museum from all other room (i.e., no room is closed and isolated). When in a room, you can see other rooms, those that are not hidden behind a wall.

You are given a map of the museum, without the wall. However, at the location of some rooms, is indicated a number standing for the number of rooms you can see from this room (including this room). *cf.* Figure 1.

Your objective is to determine the locations of the wall in the museum, so that all constraints on the number of rooms that you can see from given rooms are satisfied.

### What you are expected to do

**Answer the following question:** given a museum of size  $n \times n$ , what is the maximum number of walls that you can build inside the museum to separate all rooms (consider

## Example and solution

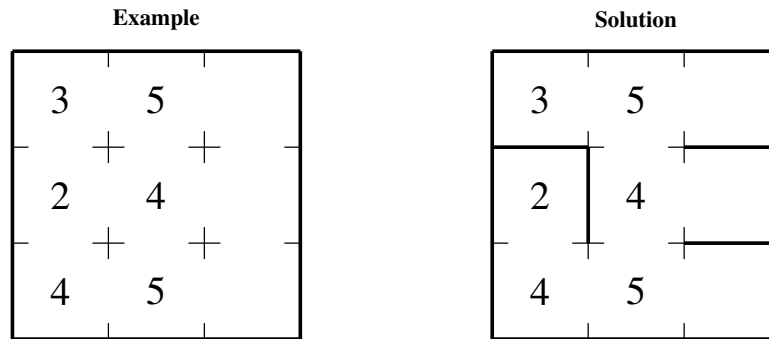


Figure 1: An example of a problem and its solution

that after building this maximum number of walls, all rooms will be isolated). Let us call  $M$  this number.

### Implement a program:

- whose input is such a map of the museum without wall, but with numbers indicating constraints
  - the input will be a text file, defined as shown on Figure 2.
- whose output is a  $M$ -tuple of 0s and 1s representing the partition of the museum that satisfies the constraints as given in the input file.

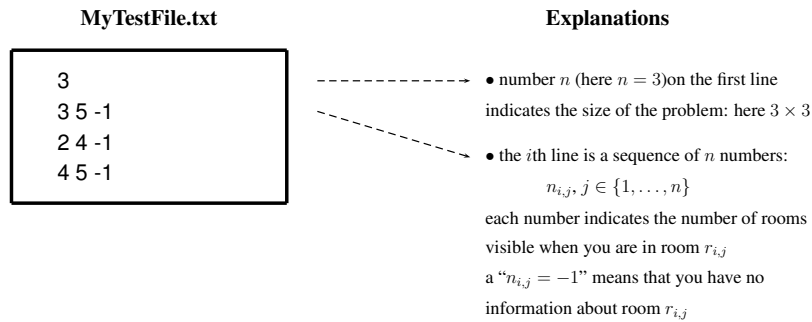


Figure 2: An example of an input file and its description

### Run experiments:

- to point out the difference between running DFS without or with constraint propagation.
- note: you are not compelled to actually run DFS without using constraints: you can just do a theoretical study of this option, and compare it against the practical experiments run when using constraints

**Write a report**, whose general outline is described in the policy rules for labs, available at [www.cs.utep.edu/mceberio/teaching](http://www.cs.utep.edu/mceberio/teaching), under Current Semester, and CS2402.

### Preliminary notions / definitions

- **constraints:** here what we call constraints are the rules of the game. i.e., each time you read a value (say  $n$ ) in a cell (say  $c$ ),  $n$  indicates how many rooms are visible from this cell: the constraint is that when being in room  $c$ , you can see  $n$  other rooms  
as a result, when determining whether you build a wall or not at a given location, you should make sure that building this wall does not contradict a rule of the game.
- **Depth-First Search:** please read <http://www.nist.gov/dads/HTML/depthfirst.html>
- **tree:** a tree structure can help traverse a search space when there are decision/choices to make.
  - in particular, in the problem you have to solve, you have to determine for each location of a wall whether there is a wall or not at this location: basically this comes down to having a variable associated to each location of a potential wall, and this variable is set to 1 if there is a wall, 0 otherwise.
  - so at the end, given all variables ( $x_1, \dots, x_M$ ), the solution of your problem is a  $M$ -tuple of 0s and 1s, representing the values of these variables  $x_i$  ( $1 \leq i \leq M$ ). (*cf.* Figure 3 for indexing of the variables)
  - a tree structure will help you represent and traverse all possible combinations of 0s and 1s for variables  $x_i$ , as follows: (*cf.* Figure 4).
  - note 1: using a tree to traverse all possibilities is a guarantee that you won't miss any possible configuration, and that you are not likely to miss a solution. However, there may be a way (that you need to figure out) not to traverse the whole tree and yet to guarantee that you don't miss the solution. Hint: use constraints to prune branches of your tree (when you use a constraint to narrow down the set of possibilities: here you basically cut branches, this is called constraint propagation).
  - note2: you don't have to actually implement the tree. Use the stack to simulate the tree traversal.

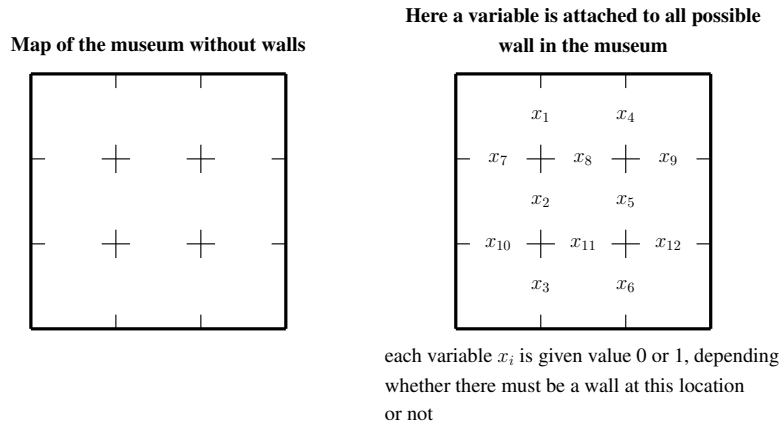


Figure 3: An possible way to define the variables to instantiate

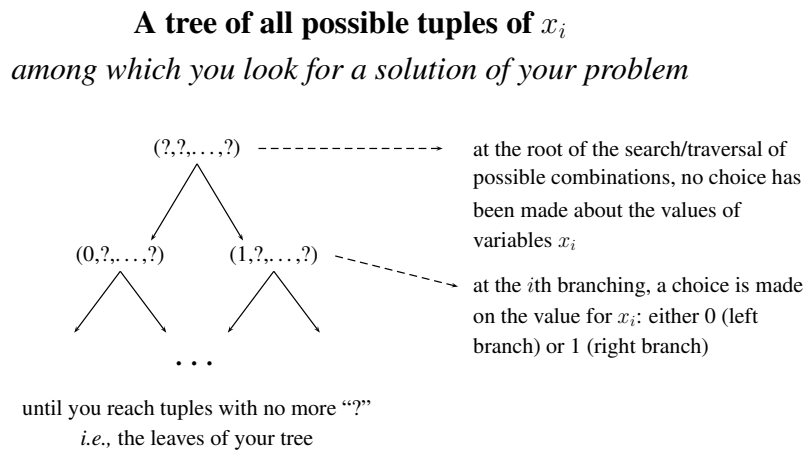


Figure 4: Description of the search tree

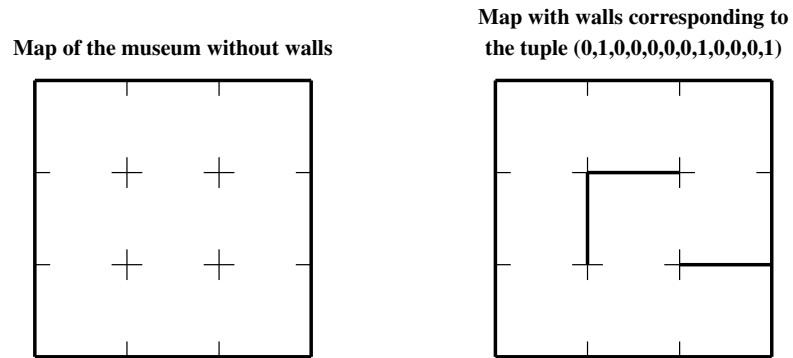


Figure 5: Partition of the museum corresponding to  $(0,1,0,0,0,0,0,1,0,0,0,1)$

- **a partition of a  $n \times n$  museum:** example: let us consider a  $3 \times 3$  museum, and the following partition:  $(0,1,0,0,0,0,0,1,0,0,0,1)$  (*cf.* Figure 5 for visual partition).