CS 120 (Spring 21): Introduction to Computer Programming II

Long Project #4

due at 5pm, Tue 16 Feb 2021

NOTE: Starting with this project, the `itertools` and `copy` libraries in Python are both **banned.**

Acknowledgements

I used `https://websudoku.com/` to auto-getnerate some puzzles. Thanks!

1 Overview

In this Long project, you will be writing a tool which will help people solve Sudoku puzzles (`https://en.wikipedia.org/wiki/Sudoku`). You will be building an interactive interface (a lot like the Twine problem); you will be reading in a grid (like the Word Search problem), and you will be doing a lot of searches and manipulations of a 2d array.

In addition, we are going to have a stack like the Twine problem; however, instead of implementing it with an array, you will be required to implement it with a linked list. Sound complex? Not at all! You can implement push and pop simply by inserting at, or removing from, the head of the list.

Your program will be named `sudoku_helper.py` .

2 Terminology and Conventions

See the Short problem spec.

3 Functions to the Rescue

Functions are always useful in programming - but (for me at least), they **really** helped me make my code simpler in this project. Look for ways to keep things common! For instance, a common operation you will perform is to "search 9 spaces; make sure there are no duplicate numbers in those spaces, and then figure out which of the digits haven't been used." That sounds a little tricky - especially if you have to do it for rows, columns, **and** the 3x3 sub-regions. But what if you had a function which, given 9 values, could scan those 9 values and give you an answer? (You don't even have to have the values in the grid; your function could require you to collect these values into an array before it looked at them.) Then your search breaks down into two simple steps (a) collect 9 values; (b) call the function to analyze them.

Similarly, you may find it valuable to implement helper functions for some or all of the following operations:

- Given a grid, look to see if there are any conflicts (that is, numbers that violate the Sudoku rules), anywhere
- Given a grid and the (x, y) location of an empty space within it, figure out all of the numbers that the space **might** be
- Duplicate a grid, and set one of the values in it
- Read one row, column, or sub-region of the grid

Remember: you can use functions to break complex pieces of code into simpler steps. But **even more importantly,** you can use functions to share functionality across two different pieces of code! If you do the same thing twice, think about creating a function for what's common.

4 Your Program Requirements

Your program must:

- Prompt the user for a filename. Read the file; it is a Sudoku puzzle, in exactly the format I described in the Short problem.
- Use a loop to read commands from the keyboard (just like Twine). You will implement four commands:
  - `set x y value` - set a single square to a new value.
  - `conflicts` - scans the Sudodku, and reports any row, column, or sub-region that has a conflict (that is, the same number shows up twice)
  - `search` - scans the Sudoku, and reports all squares that are empty, but either (a) they only have one value possible (a solution); or (b) they have no values possible (meaning we've made a mistake)
  - `back` - rewinds the history one step
- Keep track of the history of the board, using a linked list. I have provided the `ListNode` class, in `list_node.py`, for this purpose.

5 Output

To figure out what the proper output is for each command, look at the output files from the testcases.

One question you might have, however, is what order to search the board in the `conflicts` and `search` commands. For the `conflicts` command, search the columns, then the rows, then the sub-regions; when searching the sub-regions, search all of the sub-regions on the top, then the middle, then the bottom.A

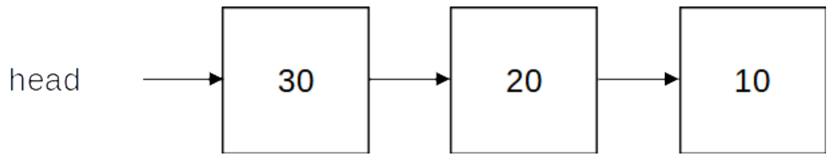
Likewise, in the `search` command, search the entire top row, then move down to the next row, and so on, through the entire board.

6 Using a Linked List as a Stack

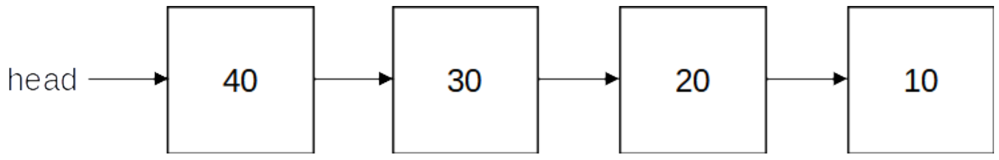
Sometimes, people implement a stack using an array; other times, they use linked lists. Both are good options, with different tradeoffs.

For this project, you are **required** to use a linked list to implement the stack. To push a new value onto the stack, create a new `ListNode` object, and then make it the new head, "pushing" it in front of whatever was there before. To pop a value off of the stack, you remove the head node, and make the second node the new head. (As with Twine, don't allow the user to pop off the last element from the stack.)

Here's how it looks in a reference diagram. Let's start with a small stack (we're using integers here; your stack should contain Sudoku boards):



When we insert a new value onto the stack, the new node becomes the head, and the old head becomes the `next` node:



7 Turning in Your Solution

You must turn in your code using GradeScope.