Izel Maras

Last updated 30 June 2014

Sudoku Checker (C)

The Task

The purpose of this program is to check if a sudoku is solved or not. The program reads in a file that may or might not contain a solved sudoku formatted according to the a set of specifications and check if the input file contains a solved sudoku or not. The program calls exit(0) if it is a solved sudoku and calls exit(1) otherwise. This and Unblackedges took us 5hours to analyze and write the code, and approximately 30 hours testing and problem solving. This program is created via a great technique called ‘Peer Programming’ with my partner in crime Sabina Campero. Credits to all the bugs.

Design

INTERFACES

UArray (Hanson's supplement)
UArray2 (An unboxed-array abstraction to support two-dimensional arrays.)
Pnmrdr
Try-except (Hanson)
Stack (Hanson)

Architecture

WHAT HAS BEEN CORRECTLY IMPLEMENTED?

Warning: This part is long because it matters.

The program reads in a image file from standard input or a file. The correct image file should be a single graymap file.We use Hanson's Try-except interface and the Pnmrdr interface to check if the image is in the right format. If the image fits the criteria we map the image data using the Pnmrdr interface. We then use our UArray2 interface to create a two dimensional array that will be later used to store the mapped image data.

We use the mapping function of the UArray2 interface to put the image pixel into our 2d array, a getter function from the Pnmrdr interface helps us to get the image pixels. For checking if a sudoku is correct, we map through each row, column and 3x3 sub squares of the sudoku calling a checker function. We check each index if the element is a number between 0 and 9. In order to check repetition we store each row, column and 3x3 sub square in a UArray of length 9 with elements in each index had been previously initialized to 0. As we attempt to add each element to a corresponding index in the array, if the index does not equals to 0,meaning the element has already added to the index, we call exit(1) meaning we have found a repetition. In order to check if there is a repetition of numbers in 3x3 subsquares,we wrote an algorithm to store loop through the 2d array and store each mini square in the UArray.

Izel Maras

Last updated 30 June 2014

Unblackedges (C)

The Task

The purpose of this program is to removes the black edges from a scanned image which is expected to be a pbm image. This and Sudoku Checker took us 5hours to analyze and write the code, and approximately 30 hours testing and problem solving. This program is created via a great technique called ‘Peer Programming’ with my partner in crime Sabina Campero. Credits to all the bugs.

Design

INTERFACES

UArray (Hanson's supplement)
UArray2 (An unboxed-array abstraction to support
two-dimensional arrays.)
Bit (Hanson)
Bit2 (An interface to support two-dimensional array of bits)
Pnmrdr
Try-except (Hanson)
Stack (Hanson)

Architecture

WHAT HAS BEEN CORRECTLY IMPLEMENTED?

Warning: This part is long because it matters.

The program reads in a image file from standard input or a file. The correct image file should be a single portable bitmap We use Hanson's Try-except interface and the Pnmrdr interface to check if the image is in the right format. If the image fits the criteria we map the image data using the Pnmrdr interface. We then use our Bit2 interface to create a two dimensional bit array that will be later used to store the mapped image data. We use the mapping function of the Bit2 interface to put the each bit in the image into our 2d bit array, a getter function from the Pnmrdr interface helps us to get the image pixels. In order to clean the edges we use another 2d bit array and a stack. Using the mapping functions of the Bit2 interface we scan the edges of the image and store the coordinates of the black edge bits in structs, then we push these structs into a stack. We use another 2d bit2 array to keep track of the visited pixels. We initialize the edges in this Bit2 array to 1, meaning visited. In order to unblack the edges. We start popping the black edges from the stack, we transform the popped bit into white and then check for its neigbours, pushing the coordinates of the black bits into the stack. We continue doing this until the stack is empty. We print the Bit2 array containing the bits from the unblacked image to standard output using Bit2_map_row_major function taken from the bit2 interface.