Lab Assignment #5

Due 2/28/2020 at 11:59pm on bCourses

The assignment will be graded by an autograder which will check the outputs of your functions. For your functions to be scored properly, it is important that the names of your functions **exactly match** the names specified in the problem statements, and input and output variables to each function are in the **correct order** (i.e. use the given function header exactly). Instructions for submitting your assignment are included at the end of this document.

If any problem specifies a certain method and/or algorithm to be used, you must implement that method and/or algorithm or you will receive a zero (0) on that problem.

On this assignment and future assignments, you are required to check that the input follows the format we provide in the table. For example, if we tell you that x is a positive integer, your code must evaluate if the x provided is a positive integer. If it is not valid, your program should terminate and you should display (disp) a message to the user why it has terminated.

1 Grades Search

`function grade = searchGrade(name, scores_lab04)`		
Input	Type	Description
`name`	`1xN char`	a name in the scores_lab04.mat file formatted with last name followed by first name and no spaces or capitalization (see details)
`scores_lab04`	`Nx2 cell`	the name(s) and the score(s) in the `scores_lab04.mat` file
Output	Type	Description
`grade`	`1x1 double`	The score recieved by `name`

Details

You happened to have acquired a file called `scores_lab04.mat` with the names and scores of everyone in a class. The first column of the file contains all the names of the students in the class. The second column contains the corresponding grades of the students for that class. You want to look up the grade that a certain student has received in the class.

You are to write a function that **recursively** searches for and returns the grade of a student. The name of the student whose grade you wish to look up must be entered in as a char array with the last name first and no spaces or capitalization. For example, Raja Sengupta would be 'senguptaraja'.

Tips

- You may notice the second input `scores_lab04` is a cell array, while you only download a .mat file from bcourse(**This document is purely fictitious. All scores in this file were generated using a random number generator.**) We provide a skeleton code for you to start with. The skeleton code will check how many inputs you have. If you

only have one input (which is `name`), the code will load `scores_lab04.mat` as the variable `scores_lab04`.

- **You will also need to include the original scores_lab04.mat file in your submission.**

2 A Mouse and a Cheeseboard

`function path = cheeseBoard(current, prev_path)`		
Input	Type	Description
`current`	`1x2 double`	Current position (row, column) of the mouse on the grid
`prev_path`	`mxn char array`	Previous path of the mouse, made up of a sequence of 'R's and 'D's. m represents the number of paths, and n represents the number of moves.
Output	Type	Description
`path`	`pxq char array` or an empty array, `[]`	Previous path, plus an additional step, unless the mouse reaches a position in which it is unable to move both right and down. If this is the case, then `path` will be an empty array.

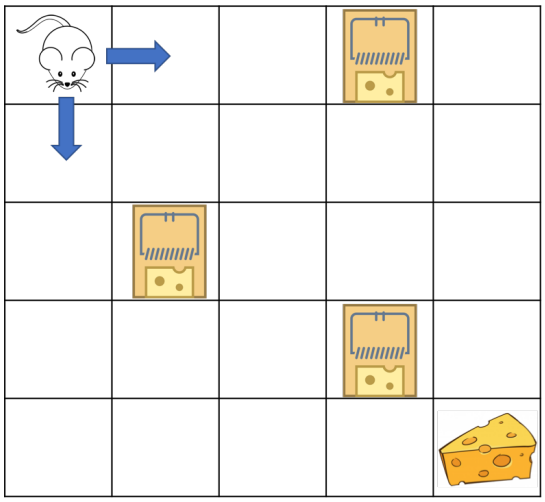
Details

A mouse confined to a 5x5 grid board is trying to reach a piece of cheese on the opposite corner. However, there are mouse traps on certain spots of the board that the mouse must avoid on its journey to obtain the cheese. You will write a function that finds all possible shortest paths the mouse can take to reach the cheese while avoiding all the traps.

You only want to know the shortest paths, which means the mouse can only move right ('R') or down ('D'). The mouse can only move one grid at a time. The mouse starts at the position (1, 1), i.e., the top left corner of the board, and is attempting to reach the cheese located at (5, 5), i.e., the bottom right of the board. Note that you may not know where the traps are beforehand. However, you will have a function `myCheck.m` whose input is a `1x2 double array` which represents the position that the mouse is trying to get to, and its output is a `logical` value. If the output of `myCheck.m` is `true`, this means the mouse can go to that position. If the output of `myCheck.m` is `false`, then the mouse cannot go to that position.

Tips

- The top-left square of the board is (1, 1), and the bottom right of the board is (5, 5). This way we can treat each tile of the board as a matrix component. Note that the first index refers to the vertical direction going downwards and the second index refers to the horizontal direction going to the right.
- An example of a valid path for the above picture would be 'RRDDRRDD'. However, there exist several other shortest paths that your code must also find.



- You can technically see where the traps are located beforehand by reading through the `myCheck.m` file. However, **do not hard code** around the trap locations in the file. We will test your code with different locations and number of traps.
- Recursion could be helpful here.

3 Berkelize it!

`function mat = berkelize(mat, row, col, prev_color, new_color)`		
Input	Type	Description
`mat`	`MxNx3 unint8`	The matrix representation of the image
`row`	`1x1 double`	The row of the selected pixel
`col`	`1x1 double`	The column of the selected pixel
`prev_color`	`1x1x3 unint8`	The color of last painted pixel
`new_color`	`1x1 char`	`'B'` for 'Berkeley Blue' or `'G'` for 'California Gold'
Output	Type	Description
`mat`	`MxNx3 unint8`	The matrix representation of modified image

Details

You have probably seen a Paint Bucket Tool in Photoshop or MS Paint. With this tool, you pick a pixel and designate it a color. The tool then fills all adjacent pixels whose colors are identical to the pixel you clicked to the designated color. In this problem, you will write a function that recursively implements this tool for an input image.

A script file `berkelizeDriver.m` is provided to you on bCourses. It shows you how your function can be called to "Berkelize" a switch logo image. An example is shown below. If you only run line 4 (i.e. to comment out line 6 12), you should see the output image on the left; If you run the whole script, you should see the output image on the right:

Figure 1: Original Image

Figure 2: Berkelized!

Tips

- An image is represented by a 3D array with dimensions MxNx3. The first two dimensions M and N describe the size/dimensions of the image, and each of the "layers" in the third dimension represent the RGB values corresponding to that pixel location in the image.
- Look up the Official Berkeley Color and the HEX representation of color to RGB values. This website can help you convert HEX to RGB values.
- Your function should make only one color change each time it is called. However, your function should be able to change a previous color into either Berkeley Blue or California Gold, depending on what the previous color is.
- Be careful when trying your function out on large images: it might cause the infamous stack overflow.

4 Floating point numbers

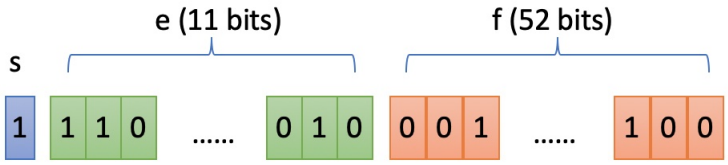
`function [result] = myDoubleFromBinary(binary)`		
Input	Type	Description
`binary`	`1x64 logical`	Logical array of length 64 made of only zeros (false) and ones (true).
Output	Type	Description
`myDouble`	`1x1 double`	The floating point number (in base 10) that is represented by `binary` using the IEEE-754 64-bit double precision binary representation.

Details

In `lab02` you already programmed your own `decimalToQuaternary` function that could convert the data between different data types. For this problem, you will write a conversion function that converts binary data into floating point numbers, i.e., numbers with fractional or decimal components.

There are multiple ways to represent floating point numbers in binary format (i.e. with only zeros and ones). The Institute of Electrical and Electronics Engineers (IEEE) defined a standard (called IEEE-754) for representing floating point numbers in binary format. IEEE-754 specifies

different formats, depending on how many bits (e.g. 16 bits, 32 bits, 64 bits, 128 bits) are used to represent each floating point number. The formats that use 32 bits and 64 bits to represent each number are commonly known as "single precision" and "double precision", respectively. In MATLAB, you can experiment by defining a variable `a` that contains the value 1 (`a = 1;`) in the command window and then using the function `whos` to inspect the variables currently defined in the workspace. You should see that the class of variable `a` is "double" and that it occupies 8 bytes = 64 bits of memory. Here is a figure showing the components of a double floating number in IEEE-754.



In this problem, we will consider double precision representations, where each number is represented using 64 bits (i.e. a sequence of 64 zeros (false) and/or ones (true)). You will write a function to convert this `logical` array to its double floating point equivalent. We index the bits from left to right: the left-most bit is the 1^{st} bit and the right-most bit is the 64^{th} bit. Instead of utilizing each bit as the coefficient of a power of 2, floats allocate bits to three different parts: the sign indicator, s, which says whether a number is positive or negative; characteristic or exponent, e, which is the power of 2; and the fraction, f, which is the coefficient of the exponent. To be more specific, the number represented by a sequence of 64 bits can be calculated using the following formula:

$$n = (-1)^s \cdot 2^{e-1023} \cdot (1+f)$$

where n represents the decimal format of the floating number.

For example: 0 01111111111 0100000000000000000000000000000000000000000000000000. Its exponent decimal is

$$s = 0$$

$$e = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6 + 1 \cdot 2^7 + 1 \cdot 2^8 + 1 \cdot 2^9 = 1023$$

$$f = 1 \cdot \frac{1}{2^2}$$

$$(-1)^0 \cdot 2^{1023-1023} \cdot (1+0.25) = 1.25$$

However, there are special cases where the above formula does not necessarily apply. **Remember to program these cases into your function.**

- if $e = 0$ and $f \neq 0$, $n = (-1)^s \cdot 2^{1-1023} \cdot (1+f)$
- if $e = 0$ and $f = 0$, $n = 0$
- if $e = 2047$ and $f = 0$, $n = (-1)^s \infty$
- if $e = 2047$ and $f \neq 0$, $n = NaN$(Not a Number)

5 Floating Point Precision vs Fixed Precision

`function [double_stats, int64_stats] = comparePrecision(binary)`		
Input	Type	Description
`binary`	`1x64 logical`	Logical array of length 64 made of only zeros (false) and ones (true), representing a single precision floating point number.
Output	Type	Description
`double_stats`	`1x3 double`	The value, precision, and maximum representable number in `double` format, listed in that order.
`int64_stats`	`1x3 int64`	The value, precision, and maximum representable number in `int64` format, listed in that order.

Each numerical data type in Matlab has a finite amount of different numbers it can represent, associated with the number of bits it uses in your computer's memory. Thus, rather than there being a continuous number line that we would expect in mathematical theory, for numbers stored in a computer, the number line is a set of discrete values that can be represented with gaps in between them where no numbers can be stored. The distance(absolute difference) between these representable numbers is often referred to as machine precision and represents the accuracy with which we can store numbers and perform mathematical operations on a computer. For this problem, we will explore how the machine precision varies for two different data formats: `double` and `int64`, both of which take up 64 bits (and thus can represent the same quantity of distinct numerical values).

In this problem, you are given the binary representation of a double precision number. You should take this data and do the following once for the `double` data format and once for the `int64` data format.

1. Calculate the numerical value, getting the value of the `double` from the binary data and getting the value of the `int64` from the `double` data.
2. Find the machine precision to which this number is represented in `int64` or `double` data format.
3. Find the largest positive (non-infinite) numerical value that can be represented in `int64` or `double` data format.

Tips

- Again, note that the output type should be `double` and `int64`.
- **Restriction**: You should **not** use the Matlab built-in functions `eps` or `realmax` for this problem.

6 Memory Consumption

There is a quiz in bcourse. Please finish it before the due date.

Submission Guidelines

When you are ready to test one or more of your function(s), download the `Lab05Tester.m` script from bCourses and run it in the same directory as your function(s). This will run a series of test cases for your function(s) and display the outputs. You should check these results with the example outputs displayed. Note that the series of tests that the script will run is not exhaustive and that it is up to you to make sure that your function fully satisfies the requirements of the problem. If you want to run all the test cases for one function, select that part of the tester script and push the `Run Section` button. If you wish to only check a single test case, you may do so by highlighting the appropriate lines in Matlab and then pressing F9 (on Windows and linux) and `shift + fn + F7` (on a Mac) to run the selected text in the Command Window.

The script will also pack your function into a zip file for final submission. Your function file names and headers should match the examples given in the problem statement. If they do not match, the tester script will not be able to find them and will issue a warning. If you do not correct this problem, your missing function(s) will not be able to be graded and **you will receive 0 points**, so be sure to check carefully! Make sure to upload the `Lab05.zip` file to bcourses before the deadline. You may resubmit your zip file by re-uploading to bcourses before the deadline, but please be aware that only the most recent zip submission is considered for grading. That said, it is critical that you include your most recent version of **all** of your functions in each new zip submission.

Your final zip file for this assignment should contain the following:

- `searchGrade.m`
- `cheeseBoard.m`
- `berkelize.m`
- `myDoubleFromBinary.m`
- `comparePrecision.m`
- `scores_lab04.mat`