ECE36800 Programming Assignment 6

Due Monday, April 27, 2020, 11:59pm

Description

This programming assignment is to be completed on your own. You will implement a program involving perhaps "graph traversals" to find a longest strictly increasing sequence of integers (stored as `short`) in a 2-dimensional table. Consider the following 4×4 table:

4	-2	0	-1
0	0	4	-1
3	2	-2	-2
5	7	9	-2

We label the rows from top to bottom 0 to 3 and the columns from left to right 0 to 3. Let the coordinates of entry i be (r_i, c_i), where r_i is the index of the row that i is in, and c_i is the index of the column that i is in. The $(2,1)$ entry of the table, for example, stores the integer 2 (stored as `short`).

Two entries, i of coordinates (r_i, c_i) and j of coordinates (r_j, c_j), are adjacent if $r_j = r_i - 1$ (j is above i), $r_j = r_i + 1$ (j is below i), $c_j = c_i + 1$ (j is to the right of i), or $c_j = c_i - 1$ (j is to the left of i). For the $(2,1)$ entry, the adjacent neighbor above it is the $(1,1)$ entry, the adjacent neighbor below it is the $(3,1)$ entry, the adjacent neighbor to its right is the $(2,2)$ entry, and the adjacent neighbor to its left is the $(2,0)$ entry. For the boundary entries in the table, they do not have four adjacent neighbors. They have only two adjacent neighbors (for the corner entries) or three adjacent neighbors (for the non-corner boundary entries).

To find a strictly increasing sequence of integers in the table, consecutive elements in the sequence must come from adjacent entries in the table. For example, the entries at $(1,1)$ and $(2,1)$, i.e., $\{0,2\}$, form the consecutive elements of a strictly increasing sequence of integers (in that order) because the $(1,1)$ entry is strictly less than the $(2,1)$ entry. However, the entries at $(1,0)$ and $(1,1)$, i.e., $\{0,0\}$, cannot be consecutive elements in a strictly increasing sequence because the entries are the same.

In this example, the entries $(0,1)$ and $(0,0)$ form a strictly increasing sequence of length 2, where the length is the number of entries in the sequence. The entries $(0,1)$, $(1,1)$, $(2,1)$, and $(3,1)$ form a different strictly increasing sequence of length 4. The *longest* strictly increasing sequence of this example, which is of length 7, is as follows: $(0,1)$, $(1,1)$, $(2,1)$, $(2,0)$, $(3,0)$, $(3,1)$, and $(3,2)$. The numbers in this strictly increasing sequence are $\{-2,0,2,3,5,7,9\}$.

We also introduce the concept of a *maximal* strictly increasing sequence. The sequence of $(0,1)$ and $(0,0)$ is a maximal strictly increasing sequence because we cannot make the sequence longer from its beginning and/or its end. The sequence of $(0,1)$, $(1,1)$, $(2,1)$, and $(3,1)$ is not a maximal strictly increasing sequence because we can make it longer by extending at the end to include $(3,2)$. A longest strictly increasing sequence is also a maximal strictly increasing sequence. However, a maximal strictly increasing sequence is not necessary a longest strictly increasing sequence.

While the longest strictly increasing sequence is unique for this example, there may be multiple longest strictly increasing sequences in other examples. You only have to find one of them (or you may find all of them and report only one of them in the output).

Deliverables

In this assignment, you are required to develop include file(s) and source file(s) that can be compiled with the following command:

```
gcc -std=c99 -pedantic -Wvla -Wall -Wshadow -O3 *.c -o pa6
```

It is recommended that while you are developing your program, you use the "`-g`" flag instead of the "`-O3`" flag for compilation so that you can use a debugger if necessary.

There are two options that the executable `pa6` can accept. The `main` function should simply return `EXIT_FAILURE` if the argument count is incorrect or the options are invalid (see further details below).

Option `"-s"`: Finding a longest strictly increasing sequence

```
./pa6 -s binary_table_file text_table_file sequence_file
```

The option `"-s"` means that you are finding a longest strictly increasing sequence of integers in a 2-dimensional table stored in the (binary) input file `binary_table_file`. The program produces two output files: `text_table_file` stores the input table in text form, and `sequence_file` stores the longest strictly increasing sequence found in binary form.

Binary table file format

The `binary_table_file` `argv[2]` is an input file in `binary` format. Given a table of m rows and n columns, $0 < m, n \leq$ `SHRT_MAX`, the first two `short`'s stored in `binary_table_file` are m and n, respectively. Here, `SHRT_MAX` is the largest `short` integer and `SHRT_MIN` is the smallest `short` integer (see `limits.h`).

After those two `short`'s, the file is followed by a total of $m \times n$ `short`'s. All $m \times n$ `short`'s are within the range of $[$`SHRT_MIN`, `SHRT_MAX`$]$.

Out of these $m \times n$ `short`'s, the first n `short`'s are in row 0. Out of these n `short`'s, the first `short` is at column 0 and the last `short` is at column $n-1$. The next n `short`'s are in row 1. The last group of n `short`'s are in row $m-1$.

The total size of the file is $(2+mn) \times$ `sizeof(short)` bytes. You may assume that all input files are all correct.

In general, the sample files given to you are named as `m_n.b` where `m` is the number of rows and `n` is the number of columns. However, you should always determine the true dimensions of a given table from the first two `short`'s of the input file. The file `4_4.b` provided to you for this assignment contains the example given earlier.

Suppose we run `pa6` with the following command

```
./pa6 -s 4_4.b 4_4.t 4_4.s
```

The file `4_4.t` provided to you stores the 2-dimensional table in text form. The file `4_4.s` provided to you stores the longest sequence in binary form.

Text table file format

The first output file `argv[3]` is simply a conversion of the binary table file to a text form. Again, we assume that there are m rows and n columns in the binary file. The first line is printed with the format `"%hd %hd\n"`, where the first `short` is the number of rows, m, and the second `short` is the number of columns, n.

After that, there should be m lines. Each line should print n `short`'s, where each `short` is printed with the format `"%hd"`. There should be a space (`' '`) character between a pair of consecutive `short`'s. There should be a newline character (`'\n'`) immediately after the last `short` in each line (row). There should not be a space character after the last `short` in a line.

Sequence file format

The second output file `argv[4]` stores the longest strictly increasing sequence information in binary form.

The file first stores the length of the longest strictly increasing sequence as an `int`. Note that we need an `int` to store the length of longest strictly increasing sequence. With the smallest integer being `SHRT_MIN` and the largest integer being `SHRT_MAX`, the longest strictly increasing sequence can be as long as `SHRT_MAX` $-$ `SHRT_MIN` $+1$, which cannot be stored as a `short`.

Let l be the length of the longest strictly increasing sequence, the output file next contains coordinates of the l entries that form the sequence. Each entry is stored as a pair of `shorts`, the (row, column) coordinates of the entry, with the row coordinate followed by the column coordinate.

The total size of the sequence file is $(\texttt{sizeof(int)} + 2l \times \texttt{sizeof(short)})$ bytes.

Return value of `main` function

If the given input file cannot be opened, your program should terminate and return `EXIT_FAILURE`.

Now, assume that the given input file can be opened. Since the given input file is binary, it is certainly feasible to find the longest strictly increasing sequence by manipulating the file. However, for efficiency reason, we expect you to allocate memory to read in the file and to use appropriate data structures to determine the longest strictly increasing sequence. If in the process of determining the longest strictly increasing sequence, your program encounters a failure in memory allocation or a failure in writing to the output files, your program should gracefully exit and return `EXIT_FAILURE`.

Of course, your program should return `EXIT_SUCCESS` otherwise.

We will test your program with valid input files of reasonable sizes. Therefore, it is unlikely that you will have to return `EXIT_FAILURE`.

Option `"-e"`: Evaluating longest strictly increasing sequence

```
./pa6 -e binary_table_file sequence_file
```

The option "-e" means that you are evaluating a sequence specified in the sequence_file (argv[3]) with respect to the binary_table_file (argv[2]). Both files are input to your program.

Your program should output four integers with the format "%d,%d,%d,%d\n", where the first integer indicates the validity of the binary table file, the second integer indicates the validity of the sequence file, the third integer indicates whether the sequence file corresponds to a strictly increasing sequence, and the fourth integer indicates whether the sequence corresponds to a maximal strictly increasing sequence.

The binary_table_file is of the same format as the input file of the "-s" option. The sequence_file is of the same format as the second output file of the "-s" option.

When we run your program, you may assume that we will provide you input files with correct formats. However, this option is more meant for you to evaluate the output of the "-s" option of your program.

If the binary_table_file cannot be opened, the first integer should be -1. If it can be opened, but of the wrong format, the first integer should be 0. If it can be opened and is of the correct format, the first integer should be 1.

If the sequence_file cannot be opened, the second integer should be -1. If it can be opened, but of the wrong format, the second integer should be 0. If it can be opened and is of the correct format, the second integer should be 1. Note that a sequence file that is of the correct format may still be an invalid sequence. That would be handled by the third and fourth integers.

The third and fourth integers are meaningful only if the first two integers are both 1. If the sequence_file corresponds to a strictly increasing sequence, the third integer is 1; otherwise, it is 0. A sequence provided in the sequence_file may not be valid for several reasons: an entry does not have valid coordinates, two consecutive entries are not adjacent neighbors, two consecutive entries are not strictly increasing.

If the sequence_file corresponds to a maximal strictly increasing sequence, the fourth integer is 1; otherwise, it is 0. Of course, the sequence can only be a maximal strictly increasing sequence if it is a strictly increasing sequence. Keep in mind that we check for maximality based on whether we can extend the sequence at the beginning and/or at the end. *You should not consider the case of splitting a given sequence and inserting additional entries between the split sequences to join them.*

The main function should return EXIT_SUCCESS only if the input files are valid (i.e., the first two integers of the terminal output are 1) and you are able to obtain the necessary memory to perform your evaluation. Even when the evaluation of the sequence is negative, you still return EXIT_SUCCESS because you have successfully evaluated the sequence. Your program should return EXIT_FAILURE when the first two integers of the terminal output are not 1 or when you could not obtain the necessary memory to perform evaluation.

Electronic Submission

The project requires the submission (electronically) of the C-code (source and include files) through Blackboard. You should create and submit a zip file called pa6.zip, which contains the .h and .c files. Your zip file should not contain a folder.

```
zip pa6.zip *.c *.h
```

You should submit `pa6.zip` to Blackboard.

If you want to use a makefile for your assignment, please include the makefile in the zip file. If the zip file that you submit contains a makefile, we use that file to make your executable (by typing "make pa6" at the command line to create the executable called `pa6`).

Grading

The assignment will be graded based on the two tasks performed by your program. The first task of determining the longest strictly increasing sequence accounts for 70 points and the second task of evaluating a sequence accounts for 30 points. Of the 70 points, the first output file accounts for 10 points and the second output file accounts for 60 points.

It is important that all the files that have been opened are closed and all the memory that have been allocated are freed before the program exits. Memory errors or any errors reported by `valgrind` will result in a 50-point penalty.

What you are given

You are given 4 sample binary table files (`4_4.b`, `4_5.b`, `5_5.b`, `10_10.b`) and the corresponding text table files (`4_4.t`, `4_5.t`, `5_5.t`, `10_10.t`) and sequence files (`4_4.s`, `4_5.s`, `5_5.s`, and `10_10.s`). It is possible that a binary table file may contain multiple longest strictly increasing sequences. Your program should not try to match the sequence exactly. However, the length should match.

To help you to understand the sequence files, we also provide the text version of these files (`4_4.st`, `4_5.st`, `5_5.st`, and `10_10.st`). In each of these files, the first line stores the number of entries in the sequence (printed with the format `"%d\n"`). Each subsequent line stores the (row, column) coordinates of an entry (printed with the format `"%hd %hd\n"`).

For the binary input file `4_4.b`, we also provide three other sequence files `4_4_00.s`, `4_4_10.s`, and `4_4_11.s`, of which `4_4_00.s` does not contain a valid strictly increasing sequence, `4_4_10.s` contains a valid strictly increasing sequence that is not maximal, and `4_4_11.s` contains a maximal strictly increasing sequence. We do not provide the text version of these sequence files.

Additional information

You may want to write a program that allows you to convert from a text table file into a binary table file. This allows you to easily create other examples to test your programs. Similarly, you may want to create programs to convert from a binary sequence file to a text sequence file and vice versa for testing.

You should write the evaluation part first. This part allows you to test your construction part. (As a good practice, you think of how to evaluate/test your program before you write your program). However, *note that a maximal strictly increasing sequence is not necessary a longest strictly increasing sequence.*

Is there a graph in this problem? If there is, is this a directed or undirected graph? Is there a special property about this graph that allows you to perform the task efficiently?

Check out the blackboard website for any updates to these instructions.