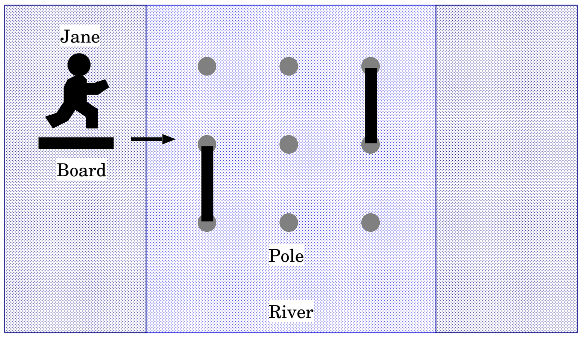
ECE36800 Programming Assignment #5

This assignment covers learning objective 1: An understanding of basic data structures, including stacks, queues, and trees; learning objective 3: An ability to apply appropriate sorting and search algorithms for a given application; learning objective 4: An ability to apply graph theoretic techniques, data structures, and algorithms for problem solving; learning objective 5: An ability to design and implement appropriate data structures and algorithms for engineering applications.

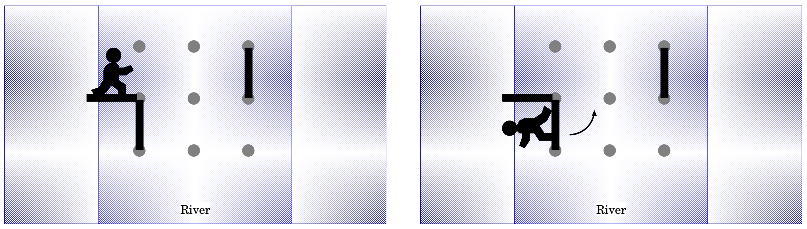
River crossing

This assignment is adapted from a problem made available by Prof. Martin Henz from the Computer Science Department of the National University of Singapore.

Jane wants to cross a river. The river is dotted with $M \times N$ poles to which boards can latch on. Some of the poles already have boards latched on such that the boards all point in the same direction of the river flow (from top to bottom). For $M = 3$ and $N = 3$, the configuration of the boards may look like this.

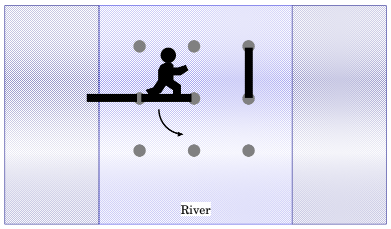


We number the poles from top to bottom 0 to 2 and from left to right 0 to 2, and associate each pole with row-column coordinates. In this example, a board is latched between poles $(1,0)$ and $(2,0)$, and another board is latched between poles $(0,2)$ and $(1,2)$.

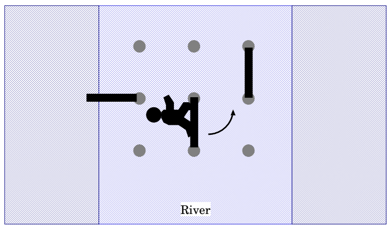


(1) (2)

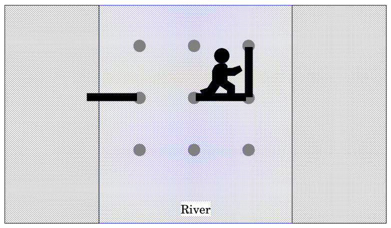
On her side of the river (the left river bank), Jane has a board. In Figure (1), she latches one end of the board on pole $(1,0)$ and keeps the other end of the board on the river bank. She then uses the board as a bridge to reach the board that is latched between poles $(1,0)$ and $(2,0)$, as shown in Figure (2).



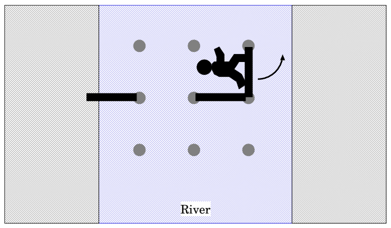
(3)



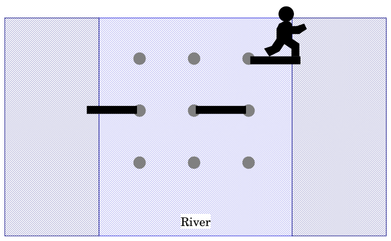
(4)



(5)



(6)



(7)

When Jane is on a board, she can un-latch any of the two ends of the board and turn the board 90 degrees (clockwise or counter-clockwise) such that the board latches on another pole. As shown in Figures (2) and (3), Jane un-latches the board from pole $(2,0)$, turns the board 90 degrees counter-clockwise, and latches the board between poles $(1,0)$ and $(1,1)$.

She then un-latches the board from pole from $(1,0)$, turns the board 90 degrees counter-clockwise, and latches the board between poles $(1,1)$ and $(2,1)$, as shown in Figure (4).

Now, she un-latches the board from pole $(2,1)$, turns the board 90 degrees counter-clockwise, and latches the board between poles $(1,1)$ and $(1,2)$, as shown in Figure (5).

She then moves onto the pre-existing board that is latched between poles $(0,2)$ and $(1,2)$, shown in Figure (6).

She un-latches the board from pole $(1,2)$, turns it 90 degrees counter-clockwise, such that one end of the board is at pole $(0,2)$ and the other end is on the right river bank. She is now able to reach the right river bank, as shown in Figure (7).

Since turning a board is quite tiresome, she wants to cross the river in such a way that she has to turn boards by 90 degrees (clockwise or counter-clockwise) as few times as possible. In this particular configuration of poles and boards, Jane requires a minimum of 4 board turns to cross the river.

While this example shows that Jane only has to move from left to right, it is possible that Jane may have to move from right to left in order to minimize the number of board turns required to cross the river.

Solution representation

We can describe Jane's solution (to cross the river with the fewest board turns) as a sequence of board positions. We assume that the left bank is at column -1 and the right bank is at column N. Moreover, we assume the left bank has M imaginary poles and the right bank also has M imaginary poles.

We represent a board position by the two poles the board is latched to. Moreover, a vertical board is represented by $(r,c)(r+1,c)$, with $0 \leq r \leq M-1$, and $0 \leq c \leq N-1$. Note that the first row-coordinate is always 1 less than the second row-coordinate. Jane could be moving from bottom to top or top to bottom on the board.

A horizontal board is represented by $(r,c)(r,c+1)$, with $0 \leq r \leq M-1$, $-1 \leq c \leq N-1$. Note that the first column-coordinate is always 1 less than the second column-coordinate. Jane could be moving from left to right or right to left on the board.

Given this representation, the following shows the solution presented in Figures (1)–(7):

```
(1,-1)(1,0)
(1,0)(2,0)
(1,0)(1,1)
(1,1)(2,1)
(1,1)(1,2)
(0,2)(1,2)
(0,2)(0,3)
```

Each of the lines tells us a board position, printed with the format

```
"(%d,%d)(%d,%d)\n"
```

The first board position must be a horizontal board that starts at column -1 (the left bank). **No other board positions in a solution (with the fewest turns) should start at column -1.**

The board positions in two adjacent lines must have one and only one common pair of coordinates.

The last board position must be a horizontal board that ends at column N (the right bank). **No other board positions in a solution (with the fewest turns) should end at column N.**

It is possible that Jane can visit a board position many times without making any turns. For example, in Figure (6), Jane could move between board positions (1,1)(1,2) and (0,2)(1,2) many times without making any turns. However, for this assignment, we require that **every board position appears at most once in a solution**.

Of course, there are other ways Jane can reach the right bank with the same number of 4 board turns. For example, we list three other possible solutions as follows:

```
(2,-1)(2,0)
(1,0)(2,0)
(1,0)(1,1)
(1,1)(2,1)
(1,1)(1,2)
```

```
(0,2)(1,2)
(0,2)(0,3)

(1,-1)(1,0)
(1,0)(2,0)
(1,0)(1,1)
(1,1)(2,1)
(1,1)(1,2)
(0,2)(1,2)
(1,2)(1,3)

(2,-1)(2,0)
(1,0)(2,0)
(1,0)(1,1)
(1,1)(2,1)
(1,1)(1,2)
(0,2)(1,2)
(1,2)(1,3)
```

Note that we did not list all possible solutions.

Deliverables

In this assignment, you have to develop your own include file and source files, which can be compiled with the following command:

```
gcc -std=c99 -pedantic -Wvla -Wall -Wshadow -O3 *.c -o pa5
```

All declarations and definition of data structures and functions must reside in the include file or source file. The executable `pa5` would be invoked as follows:

```
./pa5 input_file output_file1 output_file2
```

The executable loads the configuration of the poles and boards from `input_file` and produces two output files. The first output file corresponds to a solution that allows Jane to cross the river with the fewest number of board turns. The second output file stores the number of board turns required when Jane starts the crossing at different rows.

As shown earlier, there are different ways to achieve the the same fewest number of turns. All you have to do is produce one of those for the first output file.

Input file format

The input file, in text form, has the following format. The first line contains two numbers printed with the following format:

`"%d %d\n"`

Two integers are separated by a space and the line is terminated with a newline character. The first number is the number of rows of poles, and the second number is the number of columns of poles.

Let M be the number of rows and N be the number of columns of poles. Subsequently, the input file has $M-1$ lines. Each of the $M-1$ lines contains N characters (not counting the newline character at the end of the line), which are 0 or 1, and these N characters describe the board configuration between two adjacent rows of poles. The character 1 means there is a board and the character 0 means there is no board. Thus the second line of the input file describes the board configuration between the top two rows of poles, and so on, and the last input line describes the board configuration between the bottom two rows of poles.

The first of the N characters in a line describes the leftmost board configuration between the leftmost two poles of two adjacent rows of poles. The rightmost character in a line describes the rightmost board configuration between the rightmost two poles of the same two rows of poles.

The input file for the starting configuration of the given example contains:

```
3 3
001
100
```

For the purpose of this programming assignment, you may assume that $M \geq 2$ and $N \geq 1$. Moreover, you may assume that both M and N are no greater than `SHRT_MAX`.

Format of first output file

Please see the description on the solution representation. Although there could be many possible solutions, you should print only one of those solutions.

Format of second output file

To produce the second output file, you have to know the number of board turns required for Jane to cross the river when she starts at row 0, row 1, ..., and row $M-1$, respectively. For the example described earlier, when Jane starts at row 0, the number of board turns required for river crossing is 5. When Jane starts at row 1, the number of board turns required for river crossing is 4. When Jane starts at row 2, the number of board turns required for river crossing is also 4.

The following shows the second output file that stores these information:

```
5
4
4
```

There are M lines, with each line being printed with the format

`"%d\n"`

Starting from row 0 until row $M-1$, we print the number of board turns required for river crossing when Jane starts at the respective row.

Electronic Submission

The assignment requires the submission (electronically) of the C-code (source and include files) through Brightspace. You should create and submit a zip file called `pa5.zip`, which contains the `.h` and `.c` files. Your zip file should not contain a folder.

```
zip pa5.zip *.c *.h
```

You should submit `pa5.zip` to Brightspace.

If you want to use a `Makefile` for your assignment, please include the `Makefile` in the zip file. In that case, you can create the zip file as follows:

```
zip pa5.zip *.c *.h Makefile
```

If the zip file that you submit contains a `Makefile`, we use that file to make your executable (by typing "make pa5" at the command line to create the executable called `pa5`).

Grading

The assignment will be graded based on the two output files produced by your program. The first output file accounts for 100 points and the second output file accounts for 50 bonus points. Even if you do not wish to earn the bonus points, your program should still account for two output files.

With the bonus points, you can earn a maximum of 550 points from all programming assignments. The base points of the PA component (500 points) account for 45 points in the HW+PA score and the HW+PA+EX score. With the bonus points, you can earn a maximum of 49.5 points for the PA component for both the HW+PA score and HW+PA+EX score.

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 leaks or memory errors reported by `valgrind` will result in a 50-point penalty.

Be aware that we set a time-limit for each test case based on the size of the test case. If your program does not complete its execution before the time limit for a test case, it is deemed to have failed the test case.

What you are given

You are given some sample input files (in `pa5_examples.zip`) for you to partially test your program. The input files are `river.in[0-3]`, and output files are `river.in[0-3].out1` and `river.in[0-3].out2`.

The example provided in this description can be found in `river.in0`, and the corresponding output files are `river.in0.out1` and `river.in0.out2`.