CSCI-1200 Data Structures — Fall 2017
Homework 6 — Hashi Recursion

In this homework we will solve Hashi puzzles, an island and bridge graph construction game by Conceptis Puzzles that has appeared in Games magazine. You can play online versions of this game here:

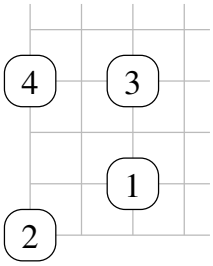
http://www.conceptispuzzles.com/index.aspx?uri=puzzle/hashi

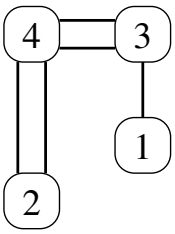
You may not search for, study, or use any code related to existing solvers for this puzzle game. *Please carefully read the entire assignment and study the examples before beginning your implementation.*

Hashi - How to Play

Your program will accept one, two, three command line arguments. The first argument is the name of a puzzle board file similar to the file on the left. Each row contains information for an island in the puzzle. The first two numbers are the x and y coordinates. *It may be helpful to grab some graph paper.* The third number for each island is the total number of bridges connecting this island to the neighboring islands. The islands may be in any order and the file may include blank lines. The middle image is a diagram of the initial Hashi puzzle for this input file.

```
0 0 2
0 3 4
2 3 3
2 1 1
```





We will connect the islands in our puzzle with horizontal and/or vertical bridges. The bridges must be straight and may not cross other bridges or touch other islands. A pair of islands may be connected by at most two direct parallel bridges. This example Hashi puzzle has only one solution (drawn on the right). Your task for the core of this homework is to find a solution to the puzzle – a set of edges/bridges (if it exists).

An additional rule of the original Hashi puzzle game is that the islands must be *connected*. A valid connected solution must allow residents of each island to move to all other islands by following a path of one or more bridges. If the optional `--connected` command line argument is provided, your code should return a solution that is connected (if a connected solution exists).

Finally, if the `--find_all_solutions` optional command line argument is provided, your program should find all solutions (if requested). This option may be used with or without the `--connected` option.

Output Formatting

Your program will print the solution(s) to `std::cout`. A sample full credit output to the puzzle above is shown on the right. Each solution begins with the string "`Solution:`", followed by the coordinates of the edges/bridges that make up the solution. The endpoints of the edge may be swapped, and the edges may be listed in any order. If two bridges connect a pair of islands, the edge will be listed twice. After listing the edges, we display an ASCII art representation of the islands and bridges.

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

 4===3
 #   |
 #   |
 #   |
 #   1
 #
 2
```

If the optional argument `--find_all_solutions` is specified, your program should output all valid, unique solutions (in any order) and then also print at the bottom the number of solutions found, e.g., "`Found 3 solution(s)`". If the puzzle has no solutions, your program should print "`No solutions`". Note that we define solution uniqueness by the resulting graph drawn, not the order of the edges or endpoints of an edge.

Additional Requirements: Recursion & Order Notation

We provide starter code to read the input puzzle file and print a solved or unsolved Hashi puzzle board. You may use or modify any or all of this provided code.

You must use recursion in a non-trivial way in your solution to this homework. As always, we recommend you work on this program in logical steps. Partial credit will be awarded for each component of the assignment. *IMPORTANT NOTE: This problem is computationally expensive, even for medium-sized puzzles! Be sure to create your own simple test cases as you debug your program.*

Once you have finished your implementation, analyze the performance of your algorithm using order notation. What important variables control the complexity of a particular problem? The dimensions of the board (w and h)? The number of nodes/islands (n) and edges/bridges (e)? In your `README.txt` file write a concise paragraph (< 200 words) justifying your answer. Include a table summarizing the running time and number of solutions found by your program on each of the provided examples. *Note: It's ok if your program can't solve the biggest puzzles in a reasonable amount of time.*

You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.

Homework 6 Hashi Contest Rules

- All students are required to submit their program to the contest. Extra credit will be awarded for programs that have a strong performance in the contest.
- Contest submissions are a separate homework submission. Contest submissions are due Thursday Oct 26th at 11:59pm. You may not use late days for the contest. (The regular homework deadline is Thursday Oct 19th at 11:59pm and late days are allowed for the regular homework submissions.)
- You may submit the same code for both the regular homework submission and the contest. Or you may make a small or significant change for the contest.
- Contest submissions *do not* need to use recursion.
- We will compile your code with optimizations enabled (`g++ -O3 *.cpp`) and run all submitted entries on the homework server.
- Programs must be single-threaded and single-process.
- We will run your program by *redirecting* `std::cout` to a file and measure performance with the UNIX `time` command. For example:

```
time hashi.out puzzle2.txt --find_all_solutions > out_puzzle2_all.txt
```

- You may want to use a *C++ code profiler* to measure the efficiency of your program and identify the portions of your code that consume most of the running time. A profiler can confirm your suspicions about what is slow, uncover unexpected problems, and focus your optimization efforts on the most inefficient portions of the code.
- We will be testing with and without the optional command line arguments `--find_all_solutions` and `--connected` and will highlight the most correct and the fastest programs.
- You may submit up to two interesting new test cases for possible inclusion in the contest. Name these tests `smithj_1.txt` and `smithj_2.txt` (where `smithj` is your RCS username). Extra credit will be awarded for interesting test cases that are used in the contest. Caution: Don't make the test cases so difficult that your program cannot solve them in a reasonable amount of time!
- In your `README_contest.txt` file, describe the optimizations you implemented for the contest, describe your new test cases, and summarize the performance of your program on all test cases.