CS575 Project 2

Due: 11:59pm on March 22

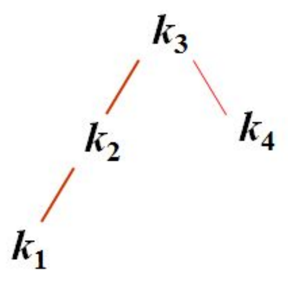
Instructor: KD Kang

1. [30%] **Optimal binary search tree algorithm.** Implement the optimal binary search tree algorithm discussed in class. (No credit will be given if any other algorithm is implemented.) Your program should take an input variable $n > 0$ followed by n float numbers that represent the probabilities for accessing the n keys (see example below). For this problem, the c or cpp file name should be **obst.c or obst.cpp.** Your makefile will generate "**obst.out**". Your command to run the optimal binary search program should be:
   ./obst.out <value of n> <n probabilities>

   Example: ./obst.out 4 0.1 0.2 0.3 0.4
   /* In this example, you have four keys: 1, 2, 3, 4 and each of them has probability of access 0.1, 0.2, 0.3, and 0.4, respectively. */

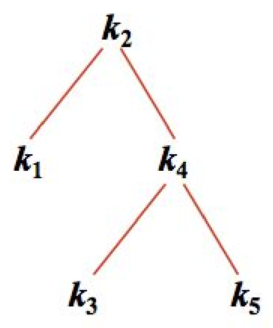
   If your optimal binary search tree looks like the following image, your program should display the result as follows (only textual output is required):

   depth 0: 3, depth 1: 2, 4, depth 2: 1, 0, 0, 0

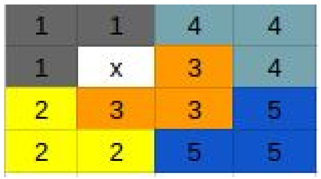


   As another example, if the optimal binary search is the following image, your program should print:

   depth 0: 2, depth 1: 1, 4, depth 2: 0, 0, 3, 5



2. [30%] Implement the tromino tiling algorithm. Your program should take an input positive integer k and the position of the hole as the Linux command line and generate a $2^k * 2^k$ board. For example if your input is 4 then your code should generate 16 x16 board. Show you result as the following image:



1	1	4	4
1	x	3	4
2	3	3	5
2	2	5	5

Here, "digits" represent the regular tile cell and "X" represents a hole.

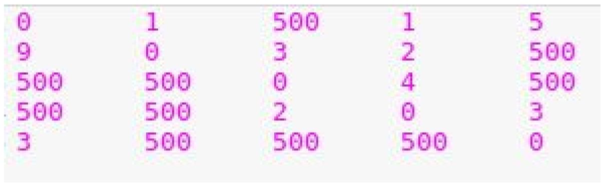
Make sure your program correctly implements the tromino tiling algorithm with $O(n^2)$ time complexity (the divide and conquer algorithm described in class); that is, there must be only one hole on the board and each of all the remaining $2^k * 2^k - 1$ squares on the board must be covered by exactly one square of one tromino tile. No credit will be given if the algorithm is incorrectly implemented, the time complexity of your program is higher than $O(n^2)$, or your program only works for specific k values.

There should be a function named **trominoTile()**, which will be called from your main() function. main() should be able to read the user input. To generate a board and print it, you can create additional function calls. Make sure you put appropriate one or two line comments to each of your function definitions. Inappropriate function declaration and unnecessary function calls will cost your points.

For the tromino problem, the c or cpp file name should be **tromino.c or tromino.cpp.** Your makefile should generate "tromino.out" file. Make sure your makefile will not run/execute the file.

To run tromino program your command should be:
./tromino.out <value of k> <hole position row number> <hole position column number>
example: ./tromino.out 3 4 5

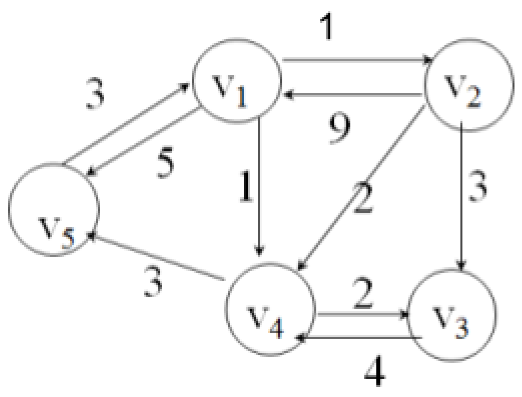
3. [25%] Implement Floyd's algorithm to find the shortest path for each pair of node in the following graph. (Use 500 instead of infinity assuming that the weight of an edge is not bigger than 499.) Print shortest paths for all pairs of nodes. Input adjacency matrix should be stored in a csv file and will be passed to run your code. See the following for a sample adjacency matrix and the corresponding graph.



```
0       1       500     1       5
9       0       3       2       500
500     500     0       4       500
500     500     2       0       3
3       500     500     500     0
```

The CSV file content will look like below:

```
0,1,500,1,5
9,0,3,2,500
500,500,0,4,500
500,500,2,0,3
3,500,500,500,0
```



Make sure your program correctly implements Floyd's algorithm. Make sure you put appropriate one or two line comments to each of your function definitions. Inappropriate function declaration and unnecessary function calls will cost your points.

For the Floyd's problem, the c or cpp file name should be **floyd.c or floyd.cpp.** Your makefile should generate "**floyd.out**" file. Make sure your makefile will not run/execute the file.

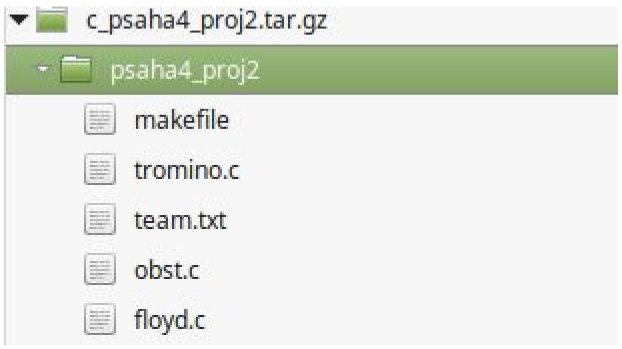
To run the **floyd** program, your command should be:
./floyd.out <input file name>

4. [10%] Elegant, compact, and easy to read code with meaningful comments.

5. [5%] Exactly follow file naming, directory naming, packaging, and submission instructions given next.

Submission Guidelines:

Please submit a tar.gz file with the below naming convention.
<user name>_project2.tar.gz
Make sure you include all the source code files as mentioned below:



Your makefile should be able to compile both the source code and generate .out files. make sure you makefile will not run the executable files.

You should run and test your code in **remote.cs.binghamton.edu** and make sure it compiles and runs there without any issues. You code will be evaluated in the same environment. If code does not run on remote server the TA will not run it on any other environments. Wrong directory structure and presence of unnecessary files will cost you points. Do not create unnecessary output files that are not asked to create.

Important Policies:

- **Project Group:** Up to two students can work as a team; however, each team should work independently of the other teams. Make sure you don't look into others' code -- each team should write its code on its own. Copying from the Internet or any other online/offline sources is strictly prohibited.
- **Plagiarism:** To detect software plagiarism, everybody's code will be cross-checked using automated tools. If automated tools find doing so, **you will receive zero point for this project, your final letter grade will be decreased by one level, and you have to fill in and sign the official form that will be kept by the Binghamton University.**
- Your code will be compiled and executed. **If your code does not compile or produce any runtime errors such as segmentation faults or bus errors, you will get zero.** Before submitting your code, run it in a Linux machine (e.g., a remote machine) to remove any such errors. You can use "**valgrind**" (http://valgrind.org/) to debug memory errors, such as segmentation faults or bus errors. Note that **robust programming** is essential for developing high quality software. If you are curious about robust programming, check this out: http://nob.cs.ucdavis.edu/bishop/secprog/robust.html
- **Assistance: You are supposed to figure out how to implement this project. The instructor and TA will be more than happy to explain fundamental algorithms covered in class; however, they will not teach you how to implement them or help you to make any kind of decision. Neither will they read or debug your code.** By the same

token, the instructor and TA will not take a look at an emailed code. If you have questions about general C/C++ programming or Linux, use Google search, which will be much faster than asking the TA. (Don't copy from the Internet though.) Contact the **TA Pankaj Saha** (psaha4@binghamton.edu) and meet during the office hours for any required clarification.

Asked Questions:

Q1: if row and column no. start from 0 then will it be fine ??
⇒ row and column number will start from ZERO

Q2: will the content of the CSV file be comma separated?
⇒Yes. Added a sample file for reference.