CS494/594 -- Lab 0

- CS494/594
- Fall, 2021
- James S. Plank
- This file: **http://web.eecs.utk.edu/~jplank/plank/classes/cs494/494/labs/Lab-0**
- Lab Directory: **/home/jplank/cs494/labs/Lab-0**

To be clear on what you should submit:

You should submit a gzipped tar file which contains two files in it:

1. **AliceGame.cpp**: which defines the **AliceGame** class and implements the **findMinimumValue()** method.
2. **BoardFolding.cpp**: which defines the **BoardFolding** class and implements the **howMany()** method.

Submit on Canvas.

Part 1

Do the 250 point problem from D1 in Topcoder SRM 639 (AliceGame). There is a writeup in: https://community.topcoder.com/stat?c=problem_statement&pm=13490&rd=16082. However, you need to be able to handle $x+y < 2^{63}$. To hand this in, submit a file that implements the appropriate class and method. The TA will compile this with his own testing **main()** to grade it.

You can compare your code with the executable **Alice** in the lab directory, which takes the two numbers on the command line. For example:

```
UNIX> cd /home/plank/cs494/labs/Lab-0
UNIX> Alice 17 8
3
UNIX> Alice 5000 5000
30
UNIX> Alice 500000000000000000 500000000000000000
292893220
UNIX> time Alice 500000000000000000 500000000000000000
292893220
0.000u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
UNIX>
```

If you'd like some good examples to try, see **Alice-Test.sh**. **Alice-Test.sh** should run in well under 1 second (mine runs in about 0.14 seconds).

Also, the PDF of my presentation on this problem is in http://web.eecs.utk.edu/~jplank/plank/classes/cs494/494/notes/Alice/index.html.

Part 2

Do the 500 point problem from D1 in Topcoder SRM 639 (BoardFolding). There is a writeup in:

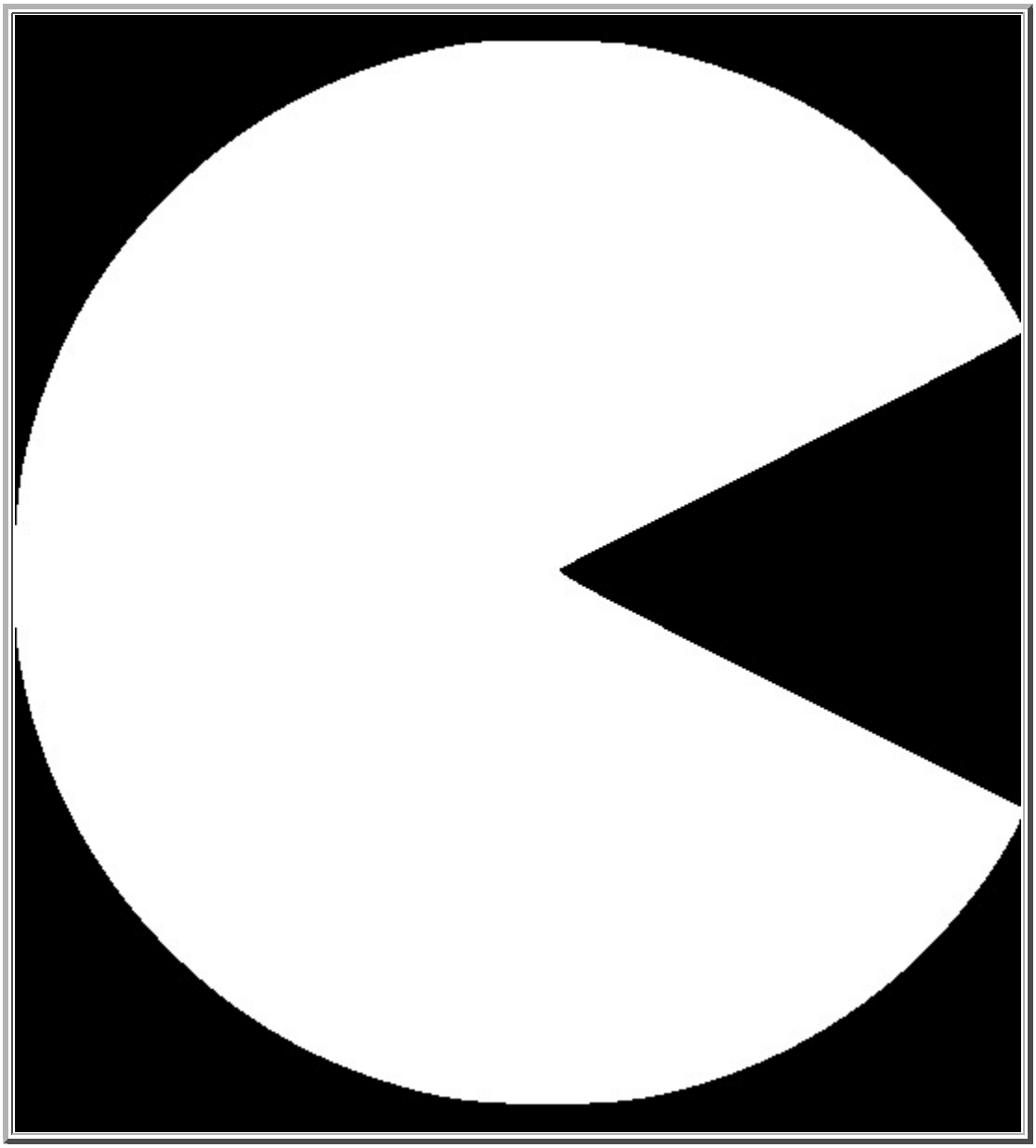
https://community.topcoder.com/stat?c=problem_statement&pm=13457&rd=16082. However, you need to be able to handle $x+y < 2^{63}$. However, the constraints on N and M are now that they are <= 3000. For this to work, you'll need to make sure that **StartingPlaces** is indeed $O(N^2)$ in the worst case. You should also convert the vector of strings into a vector of integers, as described in the presentation. Finally, you should make the method return a **long long** rather than an integer.

I have three executables in the lab directory, **Board-Fast**, **Board-Medium** and **Board-Slow**. Both take the uncompressed board on standard input. The slow version is the one I submitted to topcoder, which doesn't optimize, and has **StartingPlaces** be $O(n^3)$. The medium one has **StartingPlaces** be $O(n^2)$, but doesn't convert the board to ints. The fast one works like the medium one, only it now converts the board to ints.

I have some big boards in the files **BF*.txt**. The most challenging is **BF7.txt**.

```
UNIX> wc BF7.txt
   3000    3000 9003000 BF7.txt
UNIX> time Board-Medium < BF7.txt
8994001
6.920u 0.096s 0:07.03 99.7%     0+0k 17640+0io 1pf+0w
UNIX> time Board-Fast < BF7.txt
8994001
0.780u 0.044s 0:00.83 98.7%     0+0k 72+0io 1pf+0w
UNIX>
```

Also, in 2015, Adam Disney, who TA'd the class, created a bunch of example boards. These are in the directory **/home/jplank/cs494/labs/Lab-0/Disney_Boards**. The files *xxx.cli* hold **N** and **M**, and the files *xxx.txt* hold the compressed board. As Adam said last year, he left some "Easter Eggs" in the boards, so if you're curious, take a look, because they are pretty amusing. For example, here's the input board for file 010.cli/010.txt, where I converted the 0's and 1's into a PGM file, and then to a JPG:



He also wrote the program **compress.cpp**, which turns 0's and 1's into the compressed format of the problem.

The PDF of my presentation on this problem is in http://web.eecs.utk.edu/~jplank/plank/classes/cs494/494/notes/Board-Folding/BoardFolding.pdf.

Submitting

You should submit a gzipped tar file with two cpp files (AliceGame.cpp and BoardFolding.cpp). These should *not* have mains, but just the class/method implementations.

Your code should have comments and be written so that it reads easily.

You'll note -- no grading script and no pre-written **main()**. You'll have to write your own **main()** for testing, and our TA needs to learn how to write some good shell scripts.... You can use topcoder to help you test, though.