1. “please work.”: Hybrid Perceptron vs. GShare

   The github for this work is:

   https://github.com/yishaiSilver/Hybrid-Perceptron-vs-GShare-Branch-Predictor

2.

   I first implemented the perceptron presented by Jimenez and Lin (https://www.cs.utexas.edu/~lin/papers/hpca01.pdf), but I was having trouble finding size parameters that would give me reasonable performance while also staying within the memory constraints. As such, I decided to see if complimenting it with a gshare predictor would yield better results, and to my enjoyment, they did.

   My perceptron predictor takes in the 7 most least significant bits of the program counter and uses them to index an array of perceptrons. As such, there are $2^7 = 128$ perceptrons in my predictor. Each perceptron has 23 weights, $\mathbf{P}_{1, ..., 23}$, with each one being represented by a 2’s complement byte. This means that each weight can have any integer value from -128 to 127 and that each perceptron takes up 23 bytes. As input, the perceptron takes in the 22 most recent entries of global history, $\mathbf{G}_{1, ..., 22}$. It should be noted that the branch history is represented in the form of bits, but if a bit is 0 (not taken), the respective value of $\mathbf{G}_i$ is -1, and not 0. if The 23rd input is a constant bias term of 1, **b**.

   My gshare predictor uses 9 bits of the most recent history to index an array of 2-bit saturating counters.

   To choose between the perceptron predictor and the gshare predictor, I use a choice pattern history table (PHT), that takes in the 9 most least significant bits in the program counter and uses them to access an array of 2-bit saturating counters. This PHT is updated in the same manner as that of the tournament predictor: no changes when both predictors predict the same thing, but the 2-bit saturating counter is moved toward the correct prediction if the predictors predict different things.

3. Size Analysis
    a. Tournament Predictor:
        i. 4096 2-bit saturating counters for the choice PHT
        ii. 4096 2-bit saturating counters for the global predictor
        iii. 1024 x 10-bit local history table
        iv. 1024 x 3-bit saturating counters
        v. 12-bit global history

        Put together, we get: 4096*2 + 4096*2 + 1024 * 10 + 1024 * 3 + 12 = **29,708 bits**

    b. Custom Predictor:
        i. 128 23-byte perceptrons, or 22632 bits
        ii. 512 2-bit saturating counters for the choice PHT
        iii. 512 2-bit saturating counters for the gshare predictor
        iv. 9-bit global history

        Put together, we get: 22632 + 1024 + 9 = **24,689 bits**

    c. Yes, I probably could have used more memory for my custom predictor, but I didn't realize this until after the deadline to submit. Regardless, I think the fact that it gives better performance than my tournament predictor points to the fact that it is, indeed, a good algorithm.

4. Performance Analysis
    a. Tournament Predictor

	Misprediction Rate
fp_1.bz2	0.985
fp_2.bz2	2.735
int_1.bz2	10.740
int_2.bz2	0.395
mm_1.bz2	3.682
mm_2.bz2	8.188
Average	4.454

    b. Custom Predictor

	Misprediction Rate
fp_1.bz2	0.820
fp_2.bz2	1.087
int_1.bz2	18.945
int_2.bz2	0.466
mm_1.bz2	4.261
mm_2.bz2	9.111
Average	5.78166667

Given the performance above, I can't help but wonder why gradescope told me that my custom predictor outperformed the tournament predictor, unless the versions used for grading were much worse than my own. Regardless, changing the gshare predictor so that it uses 12 bits for the history (as opposed to 9) increases the total bit count to 22632 + 4096*2+ 1024 + 12 = **31,860 bits** and gives the following performance: (following page). We can see that it beats my tournament implementation on *almost* all counts. I'm confident that it could be all if given more time to explore the design space.

c. Custom Predictor

	Misprediction Rate
fp_1.bz2	0.815
fp_2.bz2	1.085
int_1.bz2	14.229
int_2.bz2	0.350
mm_1.bz2	3.674
mm_2.bz2	8.504
Average	4.77616667