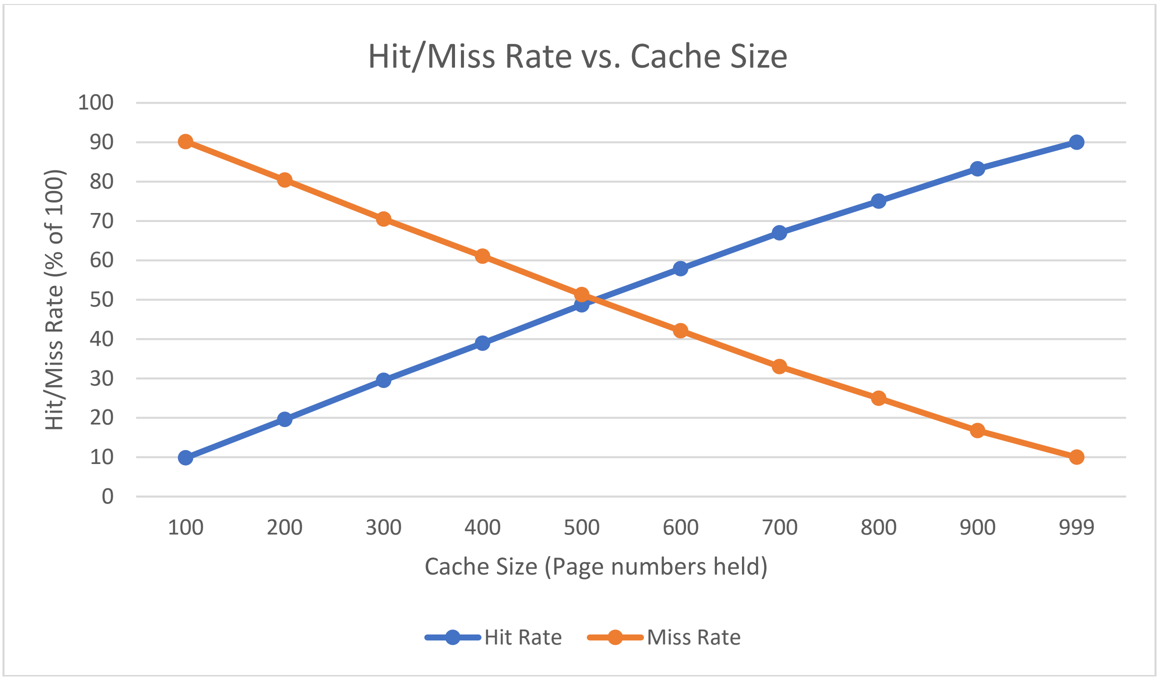
NAME: Raymond Shum
DATE: March 30, 2021
TITLE: Lab 4 – Part 4 – Write-up
DESCRIPTION: This paper discusses the sample results of my implementation of the FIFO page replacement algorithm, a test of the solutions and comparison of the hit rates of different cache sizes.

FIFO Page Replacement Analysis

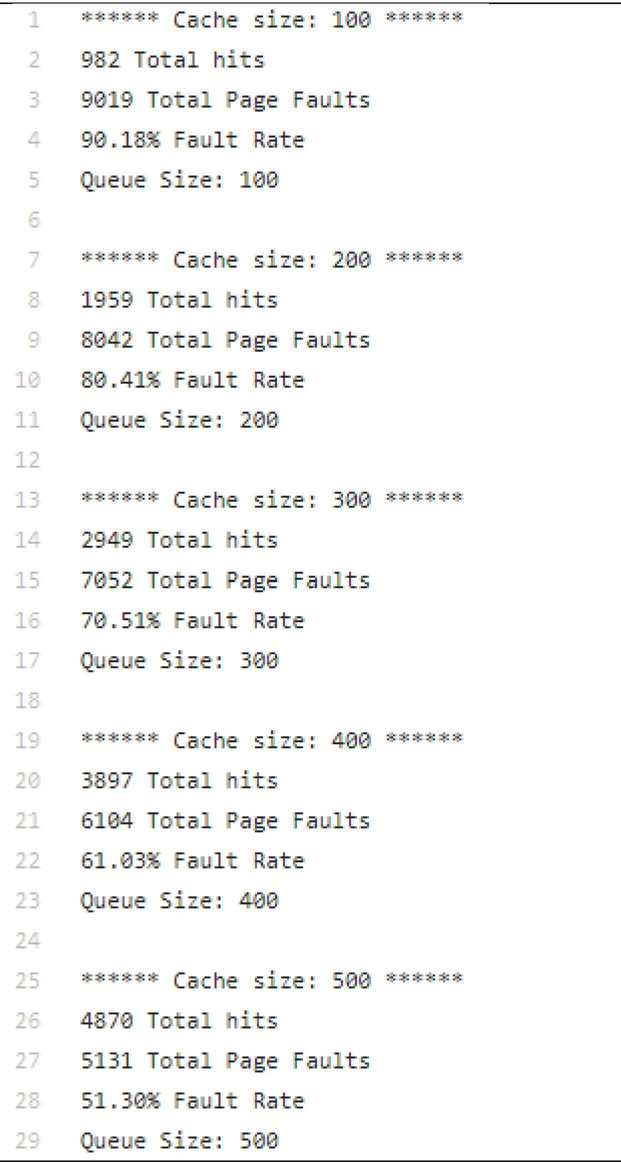
Included in the submission package is fifo.txt, which stores the results of 15 test runs of 1_fifo.o (my implementation of the page replacement algorithm. The test begins at cache size 100, which increments by totals of 100 until the cache size reaches 1500. I have plotted the data on the chart below.



As per my discussion with Ibrahim, we noticed that the miss rate decreased as cache size increased (up to a maximum of 999, determined by my hardware). This was because this data set held repeating numbers, stored in a non sequential fashion. Regarding our discussion about Belady's Anomaly, if the numbers were stored in a linear, iterative order, then we could expect 100% misses until the cache size met or exceeded the number of unique page numbers in accesses.txt.

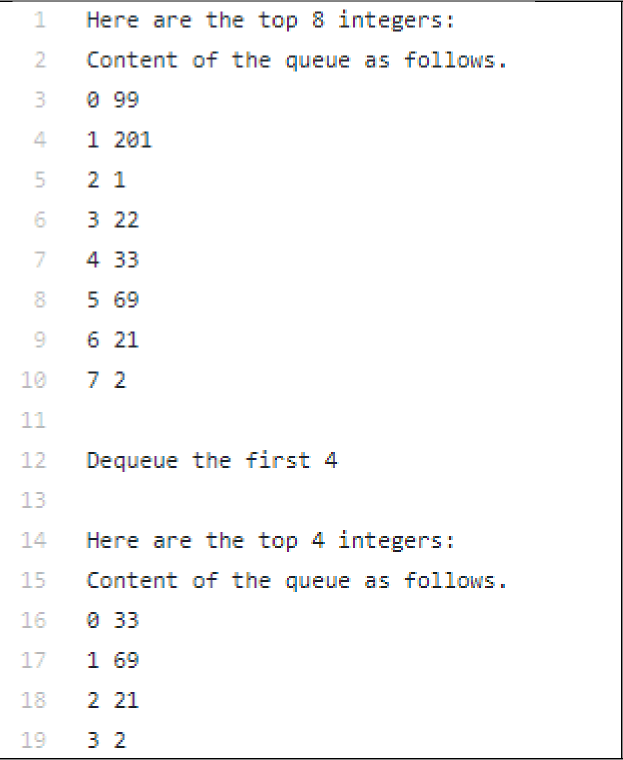
Test Results

Testing was performed using two source files: queue_test.c and 1_fifo.c. Output from the respective runs are stored in 3_testrun.txt and fifo.txt. I have included a portion of the sample data below.



```
****** Cache size: 100 ******
982 Total hits
9019 Total Page Faults
90.18% Fault Rate
Queue Size: 100

****** Cache size: 200 ******
1959 Total hits
8042 Total Page Faults
80.41% Fault Rate
Queue Size: 200

****** Cache size: 300 ******
2949 Total hits
7052 Total Page Faults
70.51% Fault Rate
Queue Size: 300

****** Cache size: 400 ******
3897 Total hits
6104 Total Page Faults
61.03% Fault Rate
Queue Size: 400

****** Cache size: 500 ******
4870 Total hits
5131 Total Page Faults
51.30% Fault Rate
Queue Size: 500
```

This is sample output from fifo.txt. The assignment only requests that we include miss rate (Fault Rate). However, comparing the additional statistics against accesses.txt. shows that the program functions properly.



```
Here are the top 8 integers:
Content of the queue as follows.
0 99
1 201
2 1
3 22
4 33
5 69
6 21
7 2

Dequeue the first 4

Here are the top 4 integers:
Content of the queue as follows.
0 33
1 69
2 21
3 2
```

This is the sample data from 3_testrun.txt, which runs the queue_test.o program. It shows that the node.c and queue.c source code runs as expected.

Implementation

I implemented the page replacement algorithm in the following way:

1. Take an incoming integer (page number) and compare it against the existing cache.
2. If the cache is empty, then enqueue the new page at the head of the cache and iterate the fault counter.
3. If the cache is not full (implicitly also not empty) and the page has not been found in the cache, then enqueue the page as the newest entry and iterate the fault counter.
4. If the cache is full and the page is not found in the cache, then evict the oldest page and enqueue the new page as the newest entry. Then iterate the fault counter.
5. If a page is found in the cache, then iterate the hit counter. This step is unnecessary but was used for testing purposes.

The program was implemented this way to meet the requirements of the FIFO page replacement algorithm, which states that the oldest entry should be evicted once the cache is full.