Cory Miljour

Data Structures II – Winter 2018

HW 5

Professor Zimmerman assisted on this assignment by me utilizing in class examples

1. What is the algorithmic complexity of creating/populating a hash table of size N? It looks like the line chart is growing at about 0(n log n) for SeparateChaining and for LinearProbing
2. What is the algorithmic complexity of put the N+1st key into a hash table of size N? Indeterminable. I used the Stopwatch method. Each run had a different increasing and decreasing result. This means it is not possible to see the actual difference using the Stopwatch method due to non specificity on small sample sizes. The way I determined this by creating a N st and then a N + 1 st, taking the difference (testNumPlusOne method in FrequencyCounter.java).
3. What is the algorithmic complexity of get for a key in a hash table of size N? Starting from 100000 entries on, LinearProbing: 0(1), SeparateChaining: O(log n). Below 100000 items LinearProbing and SeparateChaining: 0(n log n)
4. Which Hash Implementation performs better on a large # of puts/gets? I would choose Separate Chaining table because the run time appears to be less on larger creates/puts as exemplified by the charts in my sample size. However, if the sample sizes are much larger than what I used, than I may consider LinearProbing because the rate of growth according to the chart doesn’t seem as great as Separate Chaining.

Testing approach:

Used FrequencyCounter.java as the main testing program.

Disk i/o was a concern when creating/populating a table. This was solved by separating disk i/o build of the table from the actual creation/populating of the data in the table that was used for testing. This was achieved by putting the initial disk i/o build of the table into memory first, and then iterating through the table in memory to create *another* table.

The iteration utilized a couple of loops to test increasing sizes of N\*2 , repeating up to 10x then dividing by elapsed time to get the average run time.

The creation/put loops were isolated from the get loops .

Limitations:

1. Even copying the data from file to memory first, is not an accurate representation of the actual time. There is still a delay copying from table 1 in memory to table 2 in memory. It is faster than file i/o, but not perfect.
2. Stopwatch: This may be unreliable reporting due to computer systems resources being overutilized by other applications at the time of testing. A way around this is to do multiple tests over multiple time periods to get a picture of averages

Analysis:

The run time of gets for both table types is generally constant once it reaches key sizes of 150000. Up to this point, it seems to run linearly. Create/Put tends to run linearly around 100000 items. Get operations are generally fast independent of keys after a certain point (150000). Even with large data sets, the runtime of create/put/get is extremely fast compared to other algorithms.