Joon Ki Hong

CS 1332 A4

Homework 5 Analysis

**Experiment 1. The effects of size and load factor.**

A general rule of thumb that most people espouse is that the table size should always be prime. This is not always a universally held idea. We will keep a constant key (the default hashCode that you implemented above) and a load factor of 0.75. Insert 1000 randomly selected books (some may be duplicates). Search for 1000 books (some may be duplicates, some may not be in the table).

1. **Does the load factor or size seem to have any influence on the collisions?**
   1. Yes the load factor does seem to have an influence on the number of collisions. With the initial size of the table set to 100, and regrow modifier set to 0, the data presents a somewhat logarithmic increase. When the load factor was set to 0.5, the number of collisions came out to be 542. Whereas when it was set to .75 and .9, the number of collisions came out to be 917 and 1091 respectively. This seems to be the case because the higher the max load factor is set to initially, the greater the possibility for collisions before a regrow since the high max load factor allows for more elements to be added. When the initial size was decreased to 89 with the regrow modifier kept at 0, the number of collisions decreased minimally for all test cases with varying max load factors.
2. **How about searches? What was the average search count? How close was your hash table to O(1) lookups?**
   1. The amount of searches (comparisons) needed to look up elements also increased. For each of the hash tables with .5, .75, and .9 max load factors, the data returned 1388, 1710, and 2540 search probes respectively. Once again this can be attributed to the load factors. Because the increased load factors allow for more elements to be squeezed in before the initial regrow, there exists a larger probability for collisions. Because there are more collisions, that would mean that there are more chained buckets than if the hash table were to have a lower max load factor. Finally, because there are more chained buckets, more search comparisons are necessary to look up buckets and would degenerate the look up time down to O(n). The size of the initial hash table has a minimal but significant effect on the performance. The average search count among the three hash tables was 1879.3. This was about 87% more searches than it would take for it to have an O(1) runtime.

**Experiment 2. The effects of the hash code.**

Use a table size of 89, a mod factor of 1 and a load factor of .75. We now want to try different keys and see if that has any effect on collisions. Insert 1000 randomly selected books from the book list (As in our other simulations). For each test we will use a different key and hashtable. After each run, record the collision counts, and probe count.

1. **Was there any difference in the collision counts of the insertions? Did one hash code appear to be better or worse than the others? What about searching?**

|  |  |  |
| --- | --- | --- |
| **Key/Hash** | **Collisions** | **Probes** |
| **ISBN/DEFAULT** | **733** | **1765** |
| **CONTROL/INT** | **713** | **1743** |
| **ISBN/USER** | **821** | **1746** |
| **TITLE/SIMPLEADD** | **776** | **1775** |
| **CALLNO/SDBM** | **723** | **1710** |
| **AUTHOR/ADLER32** | **730** | **1733** |

* 1. There seems to be some variability between each of the algorithms and different key types. However this difference seems to be somewhat insignificant since the difference in collisions and difference in probes seem to have a small range.

**Extra Credit (All Extra Credit Assignments)**

**Implement a Coalesed Hash Table and repeat the experiments. Was this technique any better/worse than linear probing?**

**Experiment 1. (Coalesced Hash Table) The effects of the size and load factor.**

A general rule of thumb that most people espouse is that the table size should always be prime. This is not always a universally held idea. We will keep a constant key (the default hashCode that you implemented above) and a load factor of 0.75. Insert 1000 randomly selected books (some may be duplicates). Search for 1000 books (some may be duplicates, some may not be in the table).

1. **Does the load factor or size seem to have any influence on the collisions?**
   1. Yes the load factor seems to have a significant effect on the number of collisions. The lower the maximum load factor, the lower the number of collisions met. In the data set, when the coalesced hash table had .5, .75, and .9 maximum load factors, the resulting collisions were 437, 715, and 951. This shows an increasing trend in collisions. When the size was decreased to 89, the number of collisions for when the max load factors were .5, .75, and .9 were 490, 629, and 795 respectively.
2. **How about searches? What was the average search count? How close was your hash table to O(1) lookups?**
   1. For the coalesced hash tables, the number of search probes were kept relatively constant varying from 6 to 14 comparisons. The average search count was 1527. This meant that there were 52.7% more searches than it would take to have a O(1) runtime.

**Experiment 2. The effects of the hash code.**

Use a table size of 89, a mod factor of 1 and a load factor of .75. We now want to try different keys and see if that has any effect on collisions. Insert 1000 randomly selected books from the book list (As in our other simulations). For each test we will use a different key and hashtable. After each run, record the collision counts, and probe count.

1. **Was there any difference in the collision counts of the insertions? Did one hash code appear to be better or worse than the others? What about searching?**

|  |  |  |
| --- | --- | --- |
| **Key/Hash** | **Collisions** | **Probes** |
| **ISBN/DEFAULT** | **623** | **1557** |
| **CONTROL/INT** | **614** | **1557** |
| **ISBN/USER** | **606** | **1485** |
| **TITLE/SIMPLEADD** | **686** | **1634** |
| **CALLNO/SDBM** | **551** | **1307** |
| **AUTHOR/ADLER32** | **573** | **1405** |

* 1. There seems to be some variability but also a slight decreasing trend. It seems that the ADLER32 and the SDBM algorithms have better performance in both collisions and search comparisons.

**Further Analysis:**

Comparing the performance of the coalesced hash table against the linear hash table which when run with a max load factor of .75, initial size of 89 had 2066 collisions and 2255 search probes, it is clear that the coalesced hash table is faster. The coalesced table had 623 collisions and 1557 probes under the same conditions.

**JUnit 4 Tests have been attached and provided with the HW submission.**

**Randomly delete some books during insertion and searching. Does the removal of buckets make performance any worse?**

**a.** The performance of the chaining hash table when books are randomly deleted during insertion and look ups seem to have a negative impact on performance since removals require search comparisons therefore increasing the search probe counts.