FIT1008

Assignment 3 - Hash Table Analysis

Group Number: T05G04

Group Members:

- ➔ Lee Sing Yuan
- ➔ Benjamin Leong Tjen Ho
- ➔ Lim Jing Kai
- ➔ Loh Zhun Guan

Introduction to Analysis

We used 2 separate methods to analyse the good and bad hash functions, each with their own purpose:

- **`test_comparison_hash_spread()`** under the `TestPotion` class.
  - This is used to compare the spread of hash values after they have been produced by **`good_hash()`** and **`bad_hash()`** functions.
  - It uses 330 5-letter words as the `key` values to be hashed with a `tablesize` value of 11.
  - We check how many times each hash value gets produced and observe the spread of the hash values produced.
    - This is so that we can observe the uniformity of the function.
  - With a perfect hash function each hash value should be produced a total of 30 times each. So, we can offset the values by 30 in order to see the magnitude of difference from the perfect expected value. This allows us to deal with smaller numbers and also makes calculations easier.

- **`test_comparison_hash_statistics()`** under the `TestHashTable` class.
  - This is used to compare the values returned by the `statistics()` method in the LinearProbePotionTable class.
  - This method also uses the same 330 words but this time with a `tablesize` value of 659.
    - This is because in our `__init__()` method in our `LinearProbePotionTable` class, we set the tablesize as the largest prime smaller than 2 times of the max_potions value. Here, the max_potions value is 330, 2 times of that is 660.
  - This method is meant to simulate what would happen if a table with a large size was filled up to a load factor of approximately 0.5.
  - This simulation allows us to see the statistics values produced if we used the `good_hash()` and `bad_hash()` functions (i.e., the `conflict_count`, `probe_total` and `probe_max` values).

test_comparison_hash_spread()

This method produces a list of counts for how many times each hash value (0 to 10) was produced by **good_hash()** and **bad_hash()**, one list for each hash function.

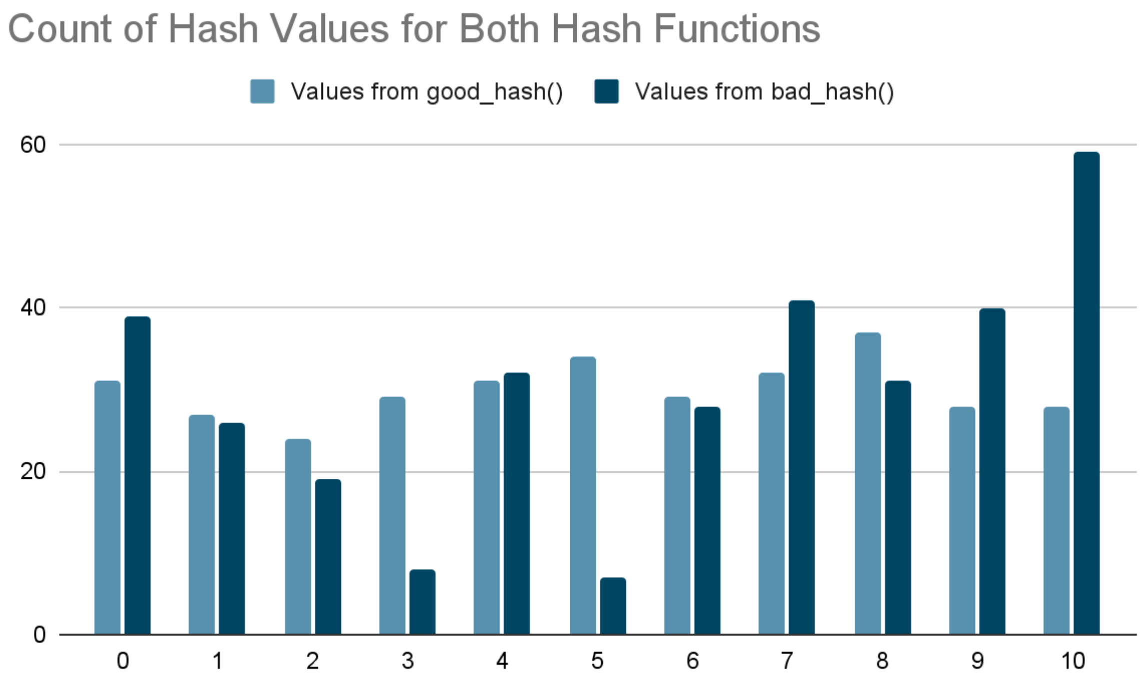
```
                 H    A    S    H         V    A    L    U    E    S
               ----------------------------------------------------------
hash values,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10 |
               ----------------------------------------------------------
good_hash(), 31, 27, 24, 29, 31, 34, 29, 32, 37, 28, 28 |
               ----------------------------------------------------------
bad_hash() , 39, 26, 19,  8, 32,  7, 28, 41, 31, 40, 59 |
               ----------------------------------------------------------
```

This is the output of the function.

(Let d = count - 30)

Hash Value		0	1	2	3	4	5	6	7	8	9	10
GOOD	**Count**	31	27	24	29	31	34	29	32	37	28	28
	d	1	-3	-6	-1	1	4	-1	2	7	-2	-2
	d^2	1	9	36	1	1	16	1	4	49	4	4
BAD	**Count**	39	26	19	8	32	7	28	41	31	40	59
	d	9	-4	-11	-22	2	-23	-2	11	1	10	29
	d^2	81	16	121	484	4	529	4	121	2	81	841

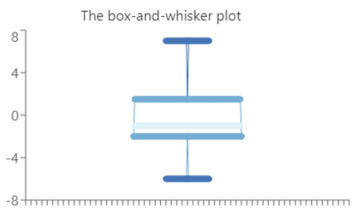
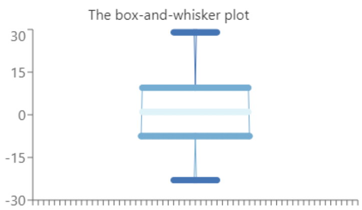
This table is a table representation of the lists returned by this analysis method. It also includes a new variable, d, which is just the count with 30 subtracted from it as well as d^2 which is used in the next table.



And this is a graphical representation of the two count values from the table above with the hash values as the x-axis. We can observe a much more uniform distribution for good_hash() compared to bad_hash() from the chart above.

To observe the spread of the hash values produced, we need to find the median as well as the variance (and hence, standard deviation) of the values for each method. But before we calculate the values, each count value is subtracted by 30 so that the counts are compared with the ideal count value.

The table below shows these values as well as some extra information for illustration purposes.

Information	**good_hash()**	**bad_hash()**
Box-and-Whisker plot	 Source	 Source
Number of hash values	11	11
min(Count)	24	7

max(Count)	37	59
Range	37 - 24 = 13	59 - 7 = 52
$Median_{count}$	29	31
Σd	0	0
Σd^2	126	2284
Mean, $\bar{x}_d$	0	0
Variance, σ_d^2	11.454545…	207.636363…
Standard deviation, σ_d	3.384456…	14.40959…

As we can see from the table above, the medians of both functions are 29 and 31, both differ from 30 by a magnitude of 1. However, the range of values for good_hash() is only 13 whereas that of bad_hash() is 4 times larger, at 52. Hence, it is clear to see that bad_hash() is not reliable when it comes to producing values in a uniform manner and good_hash() is relatively much better.

The variance and standard deviation for bad_hash() is approximately 207.6 and 14.4, which is much larger than that of the good_hash() function, i.e., 11.5 and 3.38. Hence, the bad_hash() function produces hash values with a much larger spread than that of good_hash().

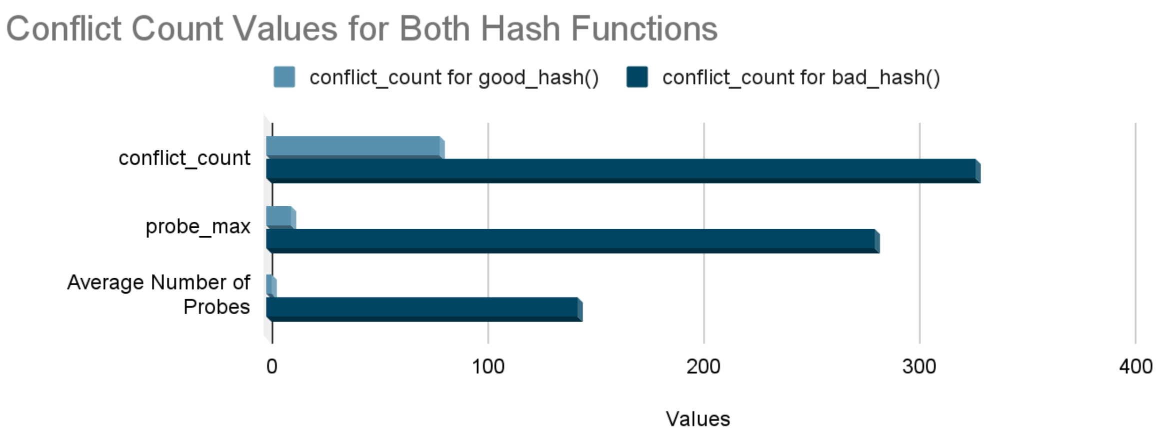
Note: The box-and-whisker plot for good_hash() seems less uniformed compared to bad_hash() but this is because the range of values for bad_hash() is much larger than good_hash() and hence, it appears to be more uniform

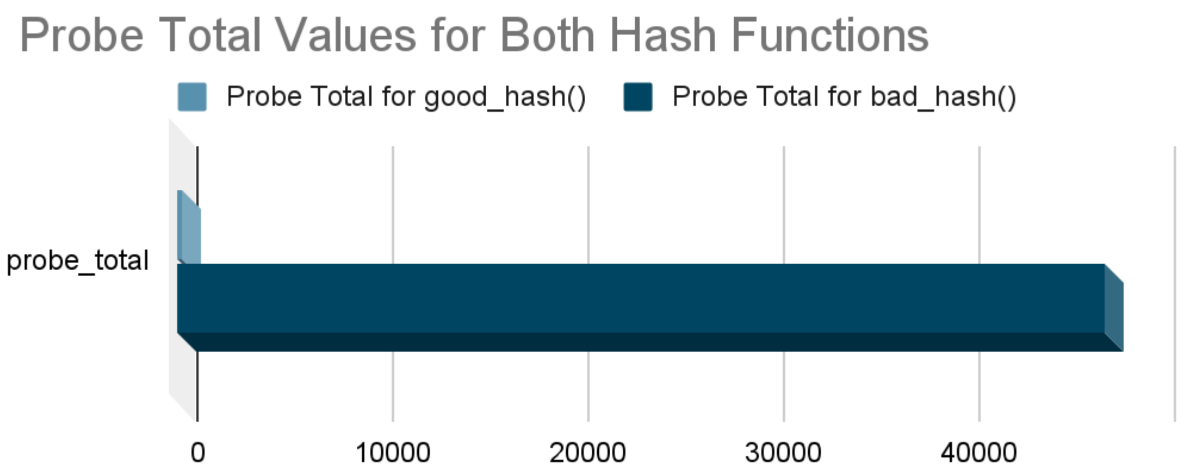
`test_comparison_hash_statistics()`

This method returns the statistics values after inserting all the elements into the LinearProbePotionTables. In the table below, Average number of probes is simply probe_total divided by conflict_count

Hashing Functions	conflict_count	probe_total	probe_max	Average Number of Probes
good_hash()	80	181	11	2.26
bad_hash()	328	47357	282	144.38

This table is a table representation of the values returned by this analysis method





These are graphical representations of the table above

As we can see from both the table and graphical representations of the statistical values returned from the function, when using bad_hash(), the amount of conflicts and probings are much, much higher compared to if good_hash() is used. Therefore, it is much better to use good_hash() if we are aiming for minimal conflicts and probing.

Note:
The reason probe_total has its own graphical representation is because if we represented all values in the same graph, probe_total for bad_hash() will absolutely dwarf the other values as it is 150 to 4300 times larger than all other values!

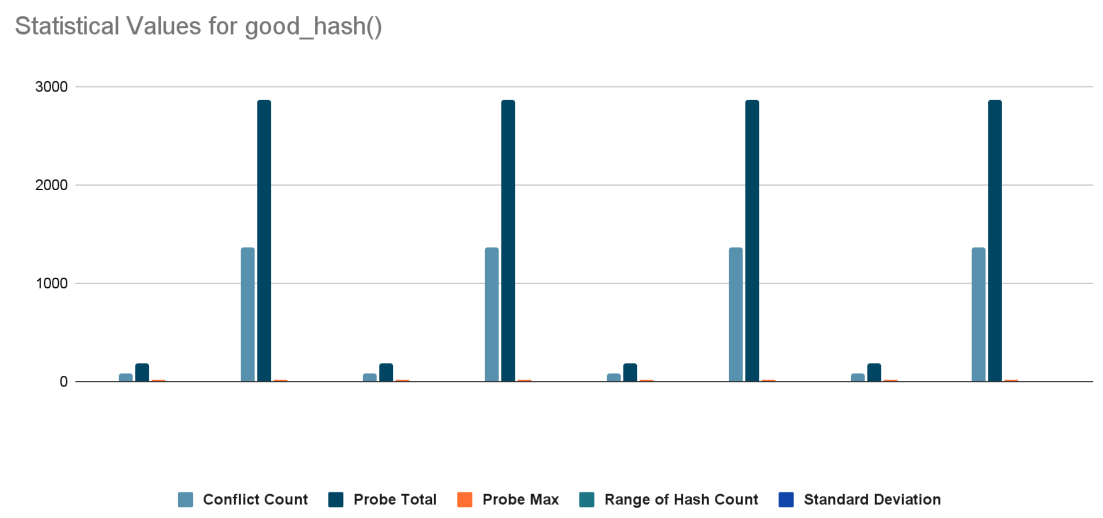
Extra Data Comparisons:

This section is simply to further prove that good_hash() is a much more reliable hashing function compared to bad_hash(). As shown in the section above, it is hard to illustrate the probe_total values alongside the others. Hence, in this section, we will show the Average Probes per Conflict in place of that

For context:

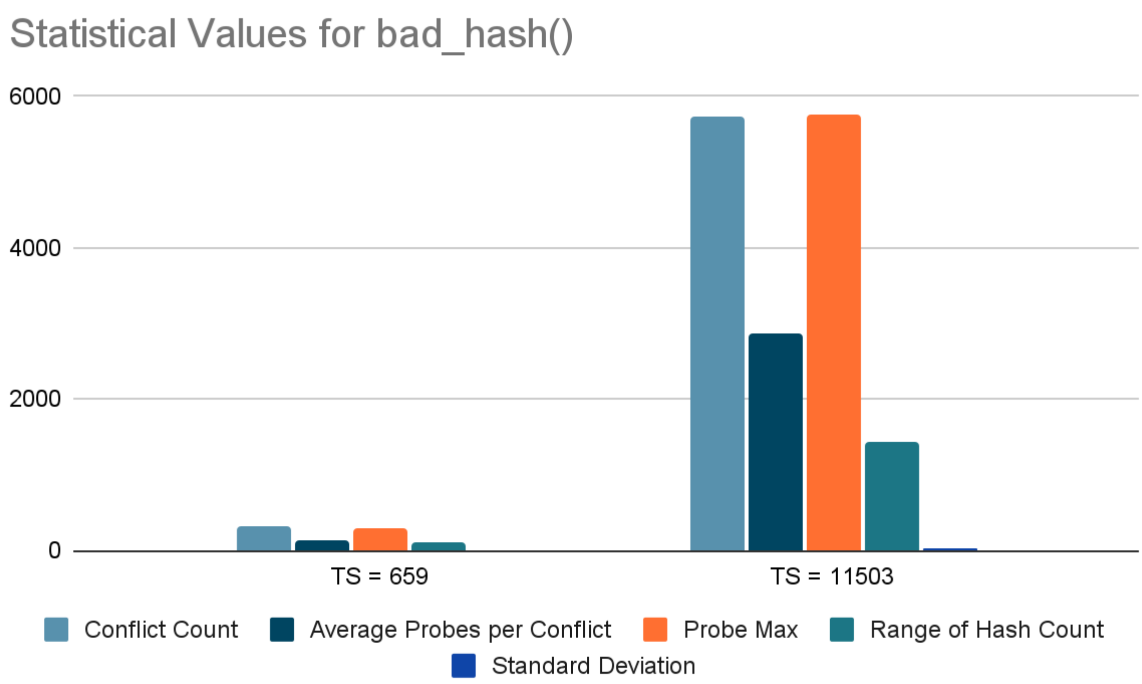
- The good_hash() function uses 2 variables *a* and *b* when hashing the key values and creating the returning hash value.
- In the different comparisons below ,we will be altering the values of *b*, tablesize and the number of elements inserted (Input Size).

b / **Tablesize, Input Size**	**12** *(Small Composite)*	**23** *(Small Prime)*	**27183** *(Large Composite)*	**27179** *(Large Prime)*
659, 330	Test Case 2	Test Case 5	Test Case 8	Test Case 11
11503, 5757	Test Case 3	Test Case 6	Test Case 9	Test Case 12



good_hash()	b = 12		b = 23		b = 27183		b = 27179	
Tablesize, Input Size	**659, 330**	**11503, 5757**	**659, 330**	**11503, 5757**	**659, 330**	**11503, 5757**	**659, 330**	**11503, 5757**
Conflict Count	80	1364	80	1364	80	1364	80	1364
Probe Total	181	2867	181	2867	181	2867	181	2867
Probe Max	11	20	11	20	11	20	11	20
Range	6	8	8	8	8	8	8	8
Standard Deviation	1.36	1.417	1.365	1.406	1.413	1.391	1.467	1.394

As we can see from the values above, the statistical values returned each time remained the same no matter the tablesize, so long as the load factor is approximately equal. This means that our good_hash() function is consistent despite changes in the tablesize value.



bad_hash()		
Tablesize, Input Size	**659, 330**	**11503, 5757**
Conflict Count	328	5742
Probe Total	47357	16446184
Probe Max	282	5753
Range	96	1448
Standard Deviation	6.735	26.25

Sources:

- The 5757 5-letter words are from a .txt file from github: https://github.com/charlesreid1/five-letter-words/blob/master/sgb-words.txt
- The 330 5-letter words are from an Google Sheets file which contains Wordle answers up until 22nd of May 2022 (meaning there are no spoilers here): https://docs.google.com/spreadsheets/d/1nl_kSpnsmY-qvez2OHvMmtIYfM7tZhP5embgmedMRxE/edit?usp=sharing
- Box-and-Whisker plot Calculator: https://www.omnicalculator.com/statistics/box-plot