# Assignment 6: Hash Tables - Analysis Document

Muyuan Zhang

2022-12-07

1. Explain the hashing function you used for `BadHashFunctor`. Be sure to discuss why you expected it to perform badly (i.e., result in many collisions).

- `BadHashFunctor` implements `lose lose` algorithm appeared in *K&R (1st ed)*. It is extremely simple and returns the summation of all bytes in the string.
- It does not consider the character order in the input string, so `hash("ab")` would return the same value as `hash("ba")`, which will cause many collisions.

2. Explain the hashing function you used for `MediocreHashFunctor`. Be sure to discuss why you expected it to perform moderately (i.e., result in some collisions).

- `MediocreHashFunctor` implements `K&R` algorithm appeared in *K&R (2nd ed)* and is slightly modified. It's basically the same as `djb2` algorithm, but the initial value of `hash` is set to 0 and each time `hash` is multiplied by 10.
- The performance of `K&R` algorithm is much better than `lose lose` algorithm for it returns different values for `hash("ab")` and `hash("ba")`. However, the multiplication factor is set to 10 instead of a prime number, which will cause more collisions than djb2 algorithm.

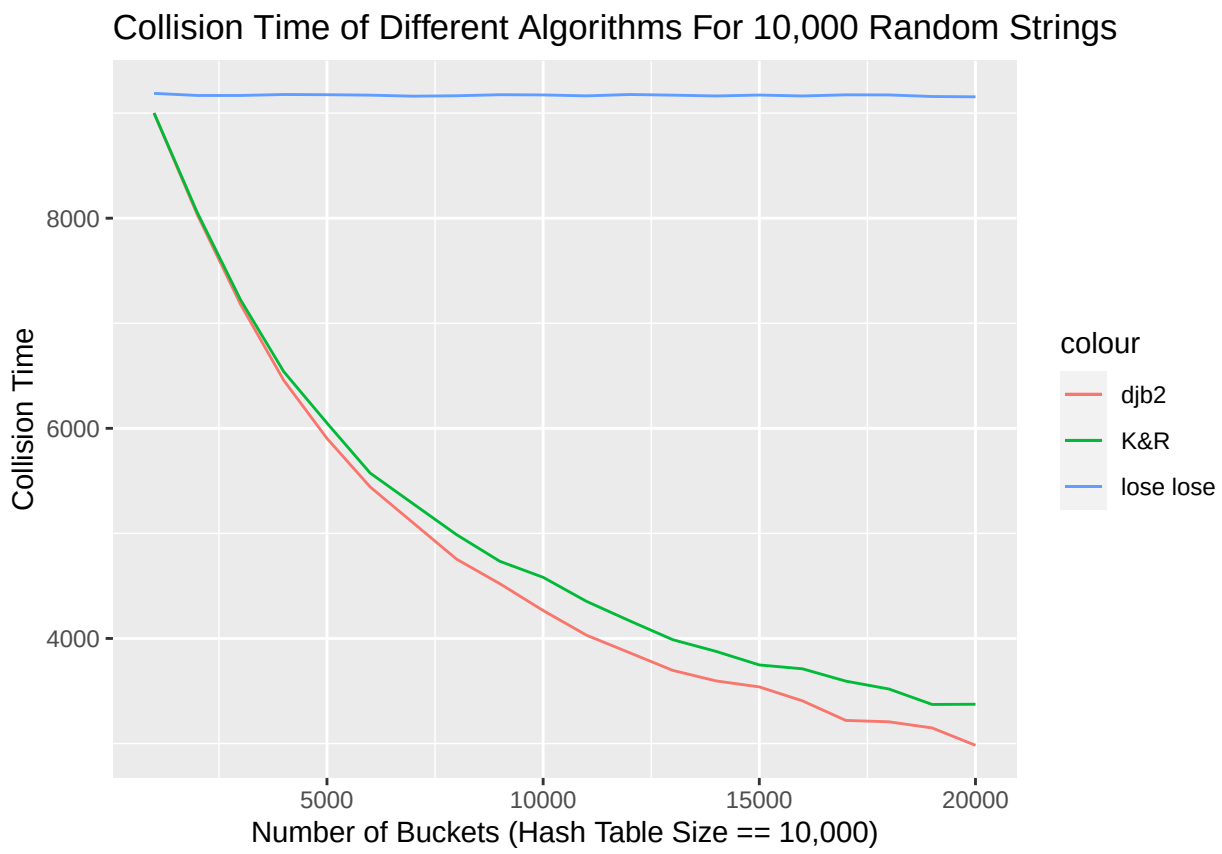
3. Explain the hashing function you used for `GoodHashFunctor`. Be sure to discuss why you expected it to perform well (i.e., result in few or no collisions).

- `GoodHashFunctor` implements `djb2` algorithm. The initial value of `hash` is 5381 and each time `hash` is multiplied by 33.
- The magic constant 5381 is an odd number and prime number, and is relatively prime to 33, which enables the function to generate a quite random distribution of hash values given a realistic set of input strings.

4. Design and conduct an experiment to assess the quality and efficiency of each of your three hash functions. Briefly explain the design of your experiment. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plot(s), as well as, your interpretation of the plots is important. A recommendation for this experiment is to create two plots: one that shows the number of collisions incurred by each hash function for a variety of hash table sizes, and one that shows the actual running time required by various operations using each hash function for a variety of hash table sizes.

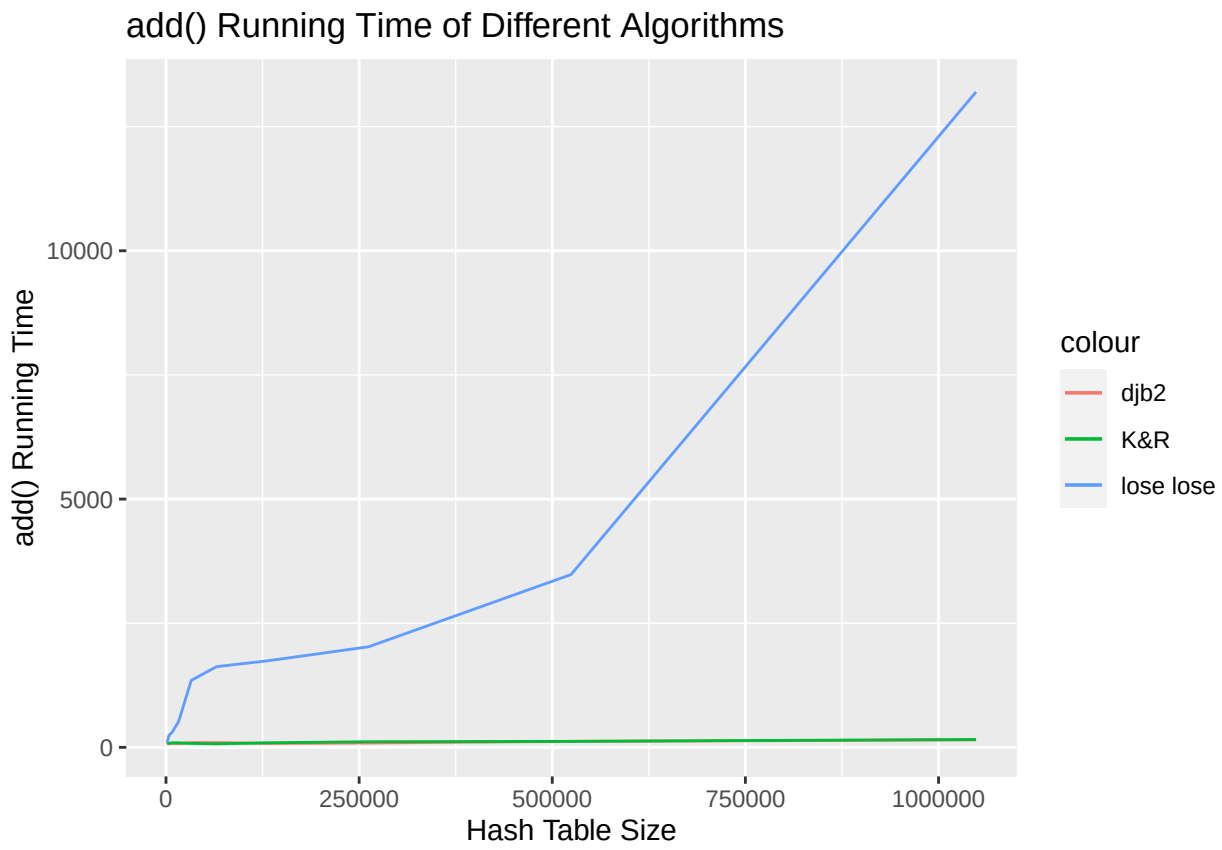
- Generate 10,000 random strings and hash them using different algorithms. Change the size of hash table, which is the number of buckets, and count the collision times for each algorithm. Plot the number of collisions incurred by each hash function for a variety of hash table sizes.

```
library(ggplot2)
capacity <- c(1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000,
              11000, 12000, 13000, 14000, 15000, 16000, 17000, 18000,
              19000, 20000)
djb2_collisions <- c(9000, 8034, 7181, 6457, 5904, 5442, 5098, 4754, 4521,
                     4266, 4032, 3864, 3696, 3596, 3539, 3406, 3220, 3207,
                     3148, 2983)
KnR_collisions <- c(9001, 8054, 7224, 6540, 6050, 5574, 5280, 4987, 4734,
                    4582, 4354, 4168, 3988, 3877, 3748, 3711, 3594, 3519,
                    3372, 3374)
loselose_collisions <- c(9188, 9168, 9168, 9177, 9175, 9171, 9161, 9165,
                         9175, 9173, 9164, 9177, 9171, 9163, 9172, 9163,
                         9174, 9173, 9158, 9155)
collisiondata <- data.frame(capacity, djb2_collisions, KnR_collisions,
                            loselose_collisions)
ggplot() + geom_line(data = collisiondata, aes(x = capacity,
    y = djb2_collisions, colour = "djb2")) + geom_point() +
    geom_line(data = collisiondata, aes(x = capacity,
    y = KnR_collisions, colour = "K&R")) + geom_point() +
    geom_line(data = collisiondata, aes(x = capacity,
    y = loselose_collisions, colour = "lose lose")) + geom_point() + labs(
    x = "Number of Buckets (Hash Table Size == 10,000)",
    y = "Collision Time",
    title = "Collision Time of Different Algorithms For 10,000 Random Strings")
```



- Run the program and record the time for `add()` on 11 set sizes: $2^{10}$, $2^{11}$, $2^{12}$... $2^{19}$, $2^{20}$.
- Time `add()` for each size of Hash Table.
- Plot these times against the Hash Table size.

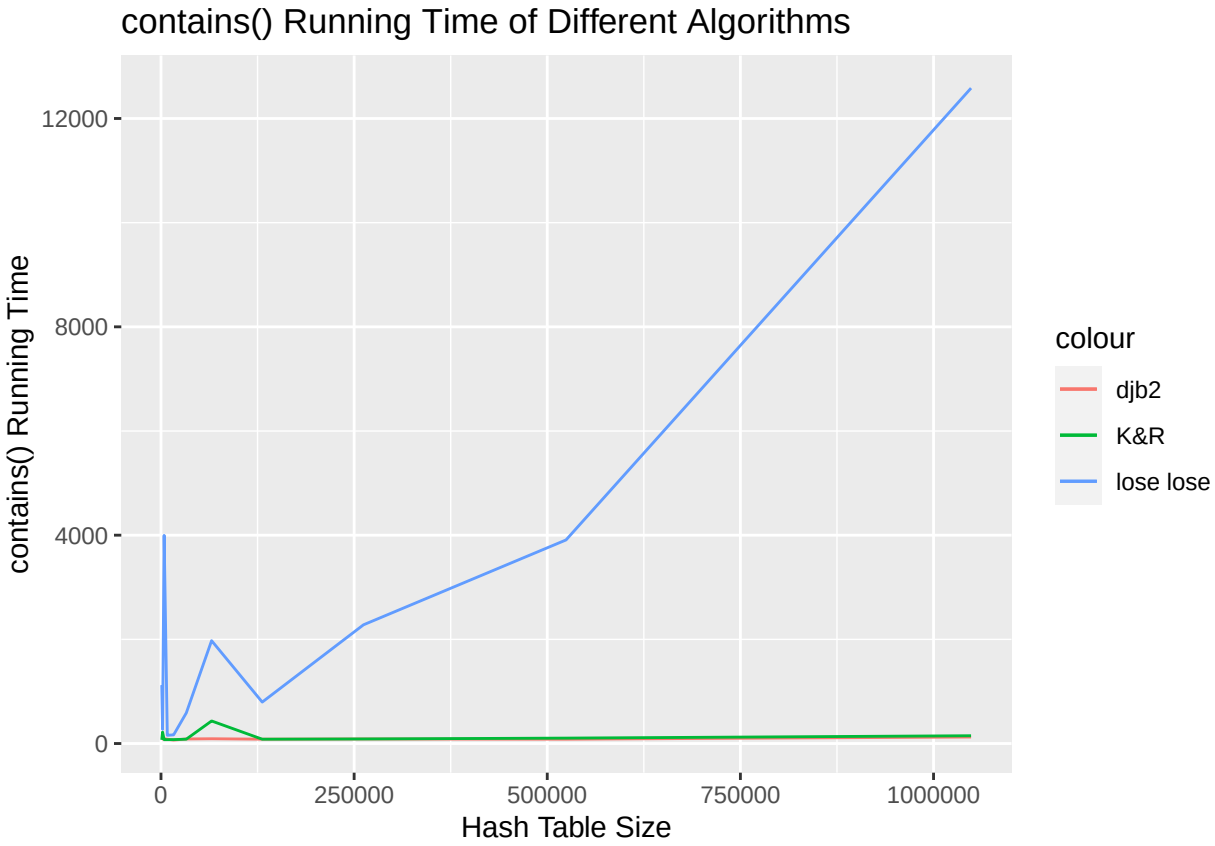
```
library(ggplot2)
size <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
          524288, 1048576)
djb2_addtime <- c(75.7, 79.2, 79.3, 79.2, 74.7, 95.7, 95.8, 79.0, 87.6,
                  120.6, 150.1)
KnR_addtime <- c(82.7, 87.5, 79.5, 91.7, 90.9, 79.2, 70.8, 91.6, 112.6,
                 120.7, 158.2)
loselose_addtime <- c(104.1, 129.1, 241.8, 308.3, 520.6, 1349.9, 1625.2,
                      1741.8, 2024.9, 3479.1, 13200.3)
addtimedata <- data.frame(size, djb2_addtime, KnR_addtime, loselose_addtime)
ggplot() + geom_line(data = addtimedata, aes(x = size,
    y = djb2_addtime, colour = "djb2")) + geom_point() +
    geom_line(data = addtimedata, aes(x = size,
    y = KnR_addtime, colour = "K&R")) + geom_point() +
    geom_line(data = addtimedata, aes(x = size,
    y = loselose_addtime, colour = "lose lose")) + geom_point() + labs(
    x = "Hash Table Size",
    y = "add() Running Time",
    title = "add() Running Time of Different Algorithms")
```




- Run the program and record the time for `contains()` on 11 set sizes: $2^{10}$, $2^{11}$, $2^{12}$... $2^{19}$, $2^{20}$.

- Time `contains()` for each size of Hash Table.
- Plot these times against the Hash Table size.

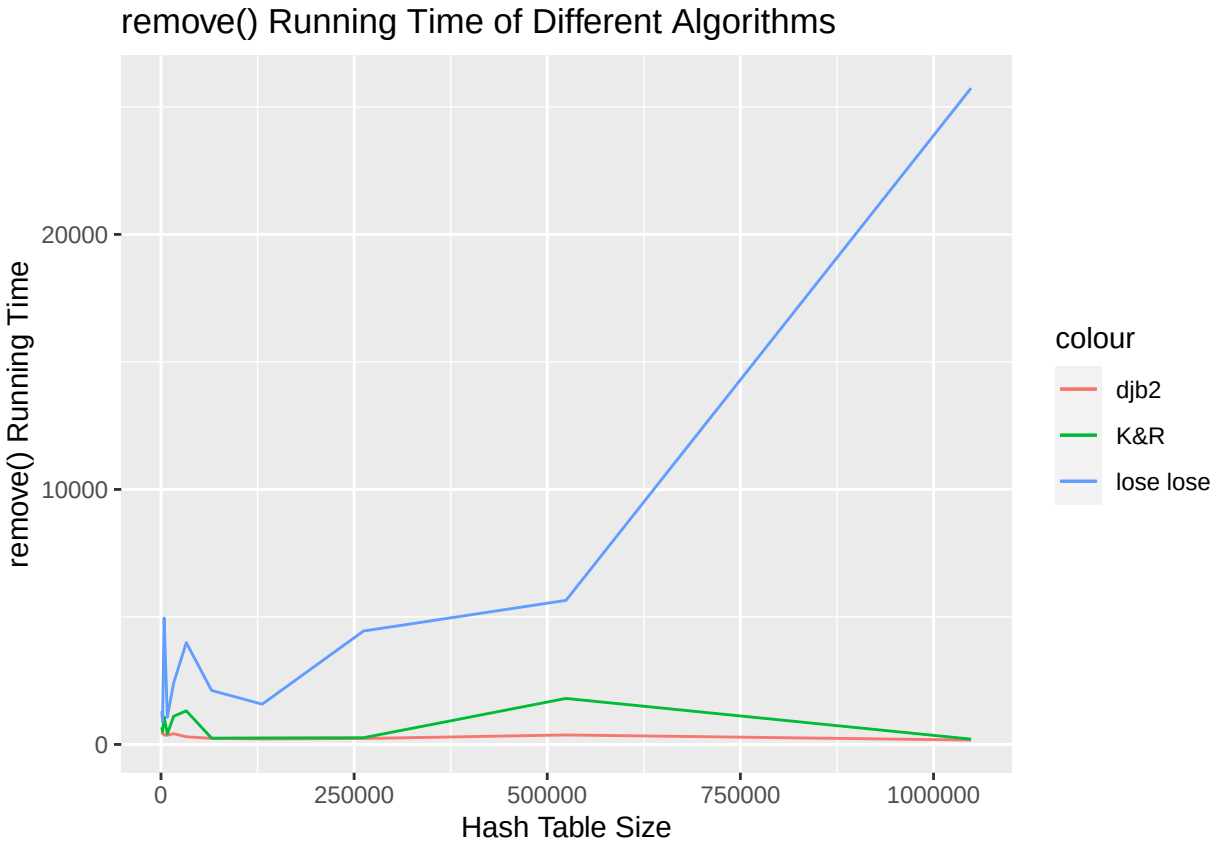
```
library(ggplot2)
size <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
          524288, 1048576)
djb2_containstime <- c(80.1, 88.1, 91.8, 83.2, 66.4, 87.6, 91.6, 83.4,
                       87.8, 83.4, 125.1)
KnR_containstime <- c(70.7, 220.8, 75.0, 74.7, 75.1, 83.3, 433.3, 83.0,
                      87.6, 104.1, 149.8)
loselose_containstime <- c(1120.8, 275.0, 3995.9, 158.4, 166.7, 587.5,
                           1975.0, 795.7, 2279.4, 3908.3, 12583.3)
containstimedata <- data.frame(size, djb2_containstime, KnR_containstime,
                               loselose_containstime)
ggplot() + geom_line(data = containstimedata, aes(x = size,
    y = djb2_containstime, colour = "djb2")) + geom_point() +
    geom_line(data = containstimedata, aes(x = size,
    y = KnR_containstime, colour = "K&R")) + geom_point() +
    geom_line(data = containstimedata, aes(x = size,
    y = loselose_containstime, colour = "lose lose")) + geom_point() + labs(
    x = "Hash Table Size",
    y = "contains() Running Time",
    title = "contains() Running Time of Different Algorithms")
```




- Run the program and record the time for `remove()` on 11 set sizes: $2^{10}$, $2^{11}$, $2^{12}$... $2^{19}$, $2^{20}$.

- Time `remove()` for each size of Hash Table.
- Plot these times against the Hash Table size.

```
library(ggplot2)
size <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
          524288, 1048576)
djb2_removetime <- c(391.6, 449.9, 366.9, 362.6, 420.7, 300.1, 237.3, 216.8,
                     233.5, 370.9, 170.8)
KnR_removetime <- c(666.6, 500.1, 1079.2, 404.1, 1104.3, 1316.6, 245.7, 254.1,
                    262.4, 1804.2, 208.4)
loselose_removetime <- c(1300.4, 883.4, 4958.4, 1066.7, 2404.3, 3991.8, 2116.7,
                         1579.1, 4445.8, 5649.7, 25737.5)
removetimedata <- data.frame(size, djb2_removetime, KnR_removetime,
                             loselose_removetime)
ggplot() + geom_line(data = removetimedata, aes(x = size,
    y = djb2_removetime, colour = "djb2")) + geom_point() +
    geom_line(data = removetimedata, aes(x = size,
    y = KnR_removetime, colour = "K&R")) + geom_point() +
    geom_line(data = removetimedata, aes(x = size,
    y = loselose_removetime, colour = "lose lose")) + geom_point() + labs(
    x = "Hash Table Size",
    y = "remove() Running Time",
    title = "remove() Running Time of Different Algorithms")
```




5. What is the cost of each of your three hash functions (in Big-O notation)? Note that the problem size

(N) for your hash functions is the length of the String, and has nothing to do with the hash table itself. Did each of your hash functions perform as you expected (i.e., do they result in the expected number of collisions)?

| Algorithm | add() | contains() | remove() |
|---|---|---|---|
| djb2 | O(1) | O(1) | O(1) |
| K&R | O(1) | O(1) | O(1) |
| lose lose | O(N) | O(N) | O(N) |

- Yes, each hash functions perform as I expected.