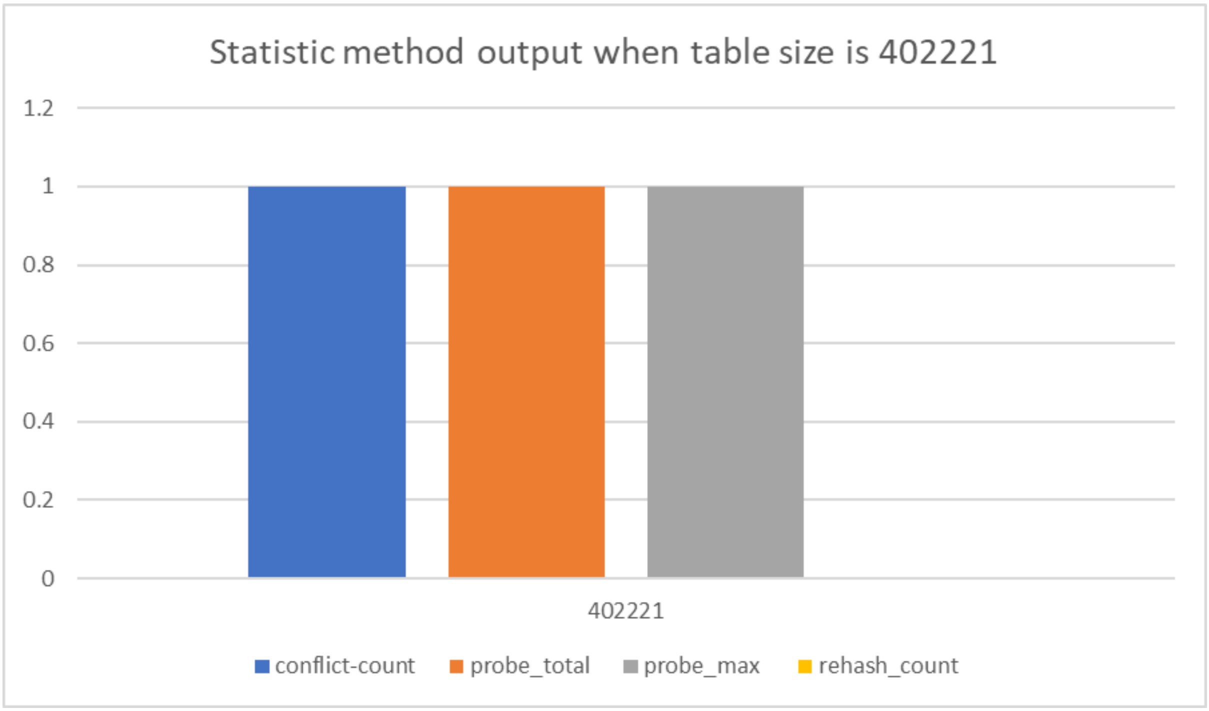
Analysis

Group name: AWSL
Group members: Chew Xin Ning, Foo Kai Yan, Gan Kai Xin, Alicia Chik Wen Quek
Date: 31st October 2022

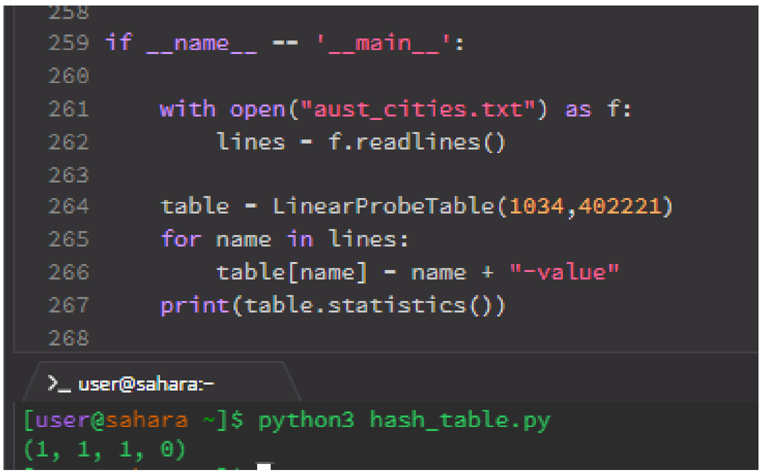
Using aust_cities.txt, consisting of 1034 string keys of Australian cities as the fake data, and considering table size of 402,221, with expected size of 1034, the expected output of the statistics method is to be less than 10 for conflict-count, probe_total and probe_max, while 0 for rehash_count. This is due to the large table size used, which leads to fewer collisions and therefore fewer conflicts. Moreover, using prime base in hash function avoids common factors, which then result in sparse tables that minimise clustering and collisions. With that, the total distance probed and the length of the longest probe chain throughout the execution of the code will also be low, as linear probing only occurs when there is collision and thus conflict. Furthermore, as the table size is huge the frequency of having to rehash is extremely low.

The hash function was tested with the fake data and table size of 402,221. With an expected size of 1034, the resultant output of the statistics method for conflict-count, probe_total, probe_max and rehash_count was 1, 1, 1, 0 respectively as displayed on the bar graph below which then correlated with the expected output of the statistics method.



The bar chart above shows the frequency of conflict count, total number of probing, maximum probing and rehash count in the hashing algorithm. Based on the statistic displayed when table size is 402221, it has a low number of conflicts and linear probing due to the sparse hash table being used. There is only one probing in total which indicates that the majority of the keys are mapped to a unique position without having collision. Therefore, there is no probing needed to find a new empty position for the key. Overall, this table size is suitable for hashing as it has a low chance of collision.

The code shown below is used to get the statistics method output:

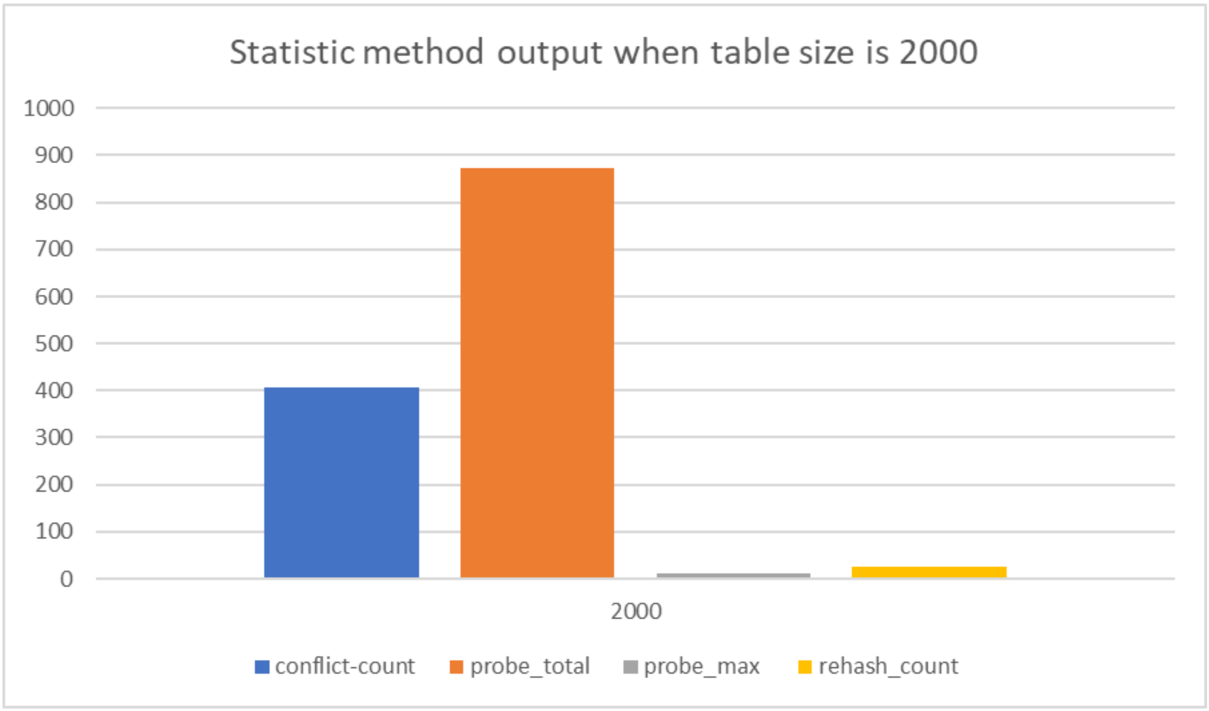


```python
if __name__ == '__main__':

    with open("aust_cities.txt") as f:
        lines = f.readlines()

    table = LinearProbeTable(1034,402221)
    for name in lines:
        table[name] = name + "-value"
    print(table.statistics())

```

```
[user@sahara ~]$ python3 hash_table.py
(1, 1, 1, 0)
```

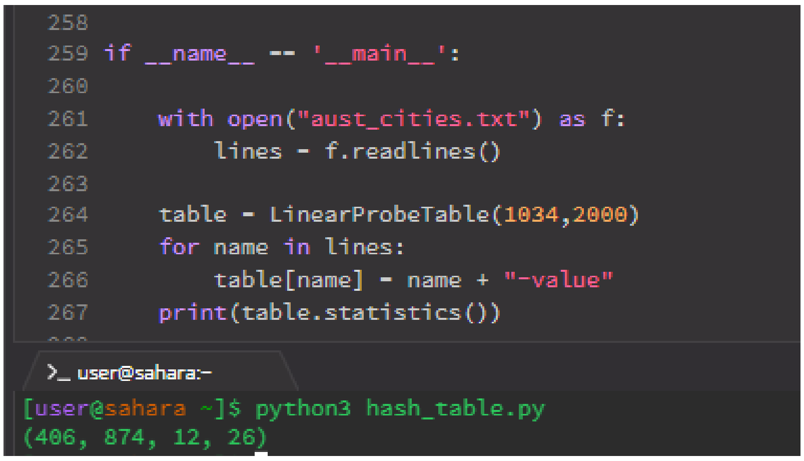
While using the same set of fake data as well as the expected size, but with smaller table size of 2000, the expected output of the statistics method to be more than 100 for conflict-count and probe_total, while less than 50 for probe_max and rehash_count. This is due to the smaller table size used as compared to the previous table size. Thus, leading to higher number of collisions and therefore greater number of conflicts. As linear probing occurs when there is collision, the total distance probed and the length of the longest probe chain throughout the execution of the code will also be higher. Additionally, as the table size is smaller the frequency of having to rehash is higher.

During testing of the hash function with the fake data and table size of 2000, with expected size of 1034, the resultant output of the statistics method for conflict-count, probe_total, probe_max and rehash_count was 406, 874, 12, 26 respectively as display on the bar graph below. Thus, correlating with the expected output of the statistics method.



By comparing the statistics when table size is 2000, it can be obviously seen that smaller table size has a higher conflict count and number of linear probing. As a smaller number of empty positions is available for the keys, there is a higher possibility for different keys to be mapped to identical positions and cause collision. Therefore linear probing was performed to find another new empty slot for the key. However as the total number of empty spaces is small and the majority of the position is filled by other keys after performing multiple hashing, and resulting clusters in the table. Hence, the number of probes would increase.

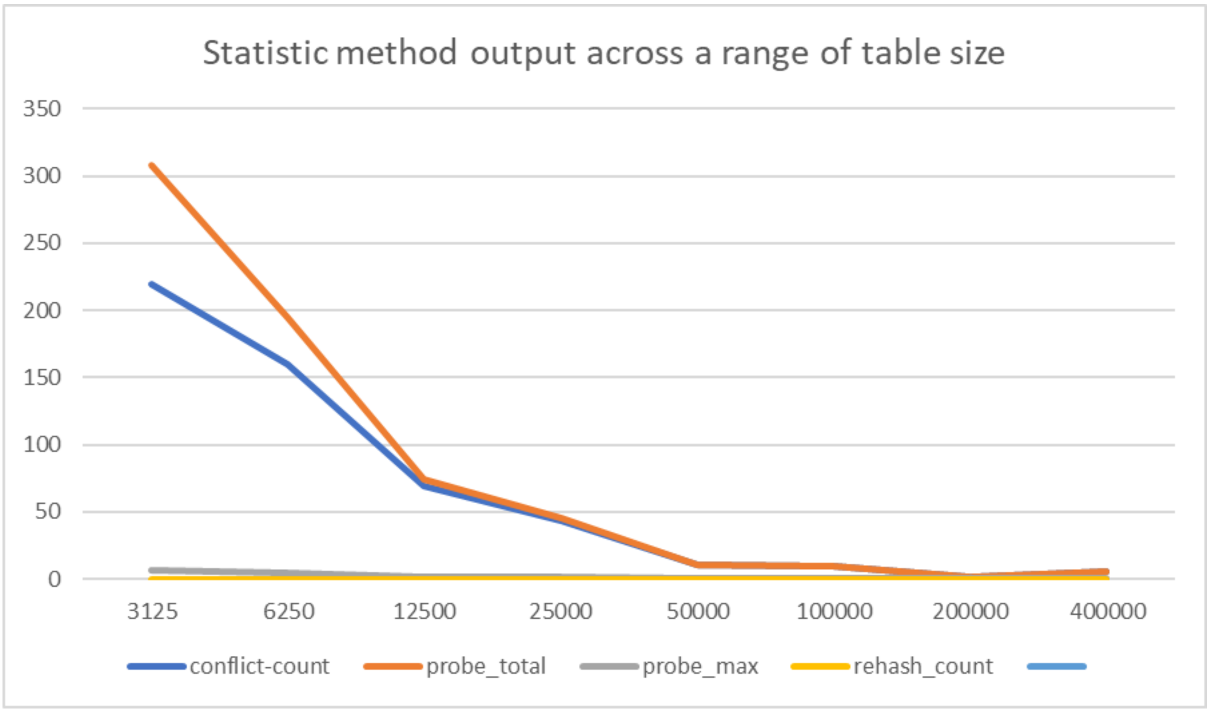
The code shown below is used to get the statistics method output:



```python
if __name__ == '__main__':

    with open("aust_cities.txt") as f:
        lines = f.readlines()

    table = LinearProbeTable(1034,2000)
    for name in lines:
        table[name] = name + "-value"
    print(table.statistics())
```

```
>_ user@sahara:~
[user@sahara ~]$ python3 hash_table.py
(406, 874, 12, 26)
```

The graph below shows the overall relationship of conflict-count, probe_total, probe_max and rehash_count across a range of table sizes using the same fake data sets and expected size.



In conclusion, the graph displayed above shows the trend that when table size increases, conflict-count, probe_total, probe_max and rehash_count decreases due to a higher number of keys that can be mapped to a unique position without conflict and collision decreases. X-axis of the line graph above is table size whereas the y-axis of the line graph above is frequency or can be known as count. It can be seen from the graph displayed above that when the value of table size is approximately 12500, the conflict-count and probe_total is of 70 whereas probe_max and rehash_count is close to 0 but when the table size is approximately 50000, the conflict-count and probe_total is of 10 whereas probe_max and rehash_count is still very close to 0. This obviously shows that conflict-count and probe_total decreases when table size increases as there was a decrease of 60 from 70 to 10. At the start of the 2 coloured lines that represent probe_max and rehash_count, it has already shown a decline of frequency from table size 3125 to 12500 which further shows that probe_max and rehash_count decreases when table size increases. Overall, it can be concluded from the line graph above that table size has a logarithmic relationship with conflict-count, probe_total, probe_max and rehash_count.