Analysis of Statistic Method inside Hash Table

From the statistics method, it will produce these values:

1. conflict_count: total number of conflicts (two or more values have the same key value)
2. probe_total: total probe_chain throughout the hash table
3. probe_max: longest probe chain
4. rehash_count: how many rehashing has been done if total element in the hash table > half of the table size

Input 10 values into the hash table simultaneously:

Eva, Amy, Tim, Ron, Jan, Kim, Dot, Ann, Jim, Jon

Hash key result:

- Eva 12

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
												Eva						

- Amy 8

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
								Amy				Eva						

- Tim 8, Tim will be placed at 9, probe length = 1 [8]. It is conflict and collision

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
								Amy	Tim			Eva						

- Ron 6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
						Ron		Amy	Tim			Eva						

- Jan 17

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
						Ron		Amy	Tim			Eva					Jan	

- Kim 18

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
						Ron		Amy	Tim			Eva					Jan	Kim

- Dot 11

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
						Ron		Amy	Tim		Dot	Eva					Jan	Kim

- Ann 8, Ann will be placed at 10, probe length = 2 [8,9]. It is conflict and collision

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
						Ron		Amy	Tim	Ann	Dot	Eva					Jan	Kim

- Jim 17, Jim will be placed at 0, probe length = 2 [17,18]. It is conflict and collision

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Jim						Ron		Amy	Tim	Ann	Dot	Eva					Jan	Kim

- Jon 17, Jon will be placed at 1, probe length = 3 [17,18,0]. It is conflict and collision

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Jim	Jon					Ron		Amy	Tim	Ann	Dot	Eva					Jan	Kim

The result of the hash table is:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Jim	Jon					Ron		Amy	Tim	Ann	Dot	Eva					Jan	Kim

From this case, we have:

4 conflict counts when:

1. we insert Tim with key 8, because we already had Amy on key 8
2. we insert Ann with key 8, because we already had Tim on key 8
3. we insert Jim with key 17, because we already had Jan on key 17
4. we insert Jon with key 17, because we already had Jim on key 17

probe_total:

1. Tim 8 -> 1 (Amy 8 is filled, Tim will be at 9)
2. Ann 8 -> 2 (Amy 8 is filled, Tim 9 is filled, Ann will be at 10)
3. Jim 17 -> 2 (Jan 17 is filled, Kim 18 is filled, Jim will be at 0)
4. Jon 17 -> 3 (Jan 17 is filled, Kim 18 is filled, Jim 0 is filled, Jon will be at 1)

So, the probe total is 6 (1 + 2 + 2 + 3)

probe_max:

➢ From the previous result, we noticed that Jon runs the biggest probing which is 3.

rehash_count:

➢ When the hash table consists values more than half of the table size which in this case > then (0.5 * 19 which is 9.5), we will rehash the table.
➢ In this case, when we already have 9 values inside the table, the above condition is still false because 9 > 9.5 is false, so when we insert the tenth value, it hasn't rehashed the table. It also means that in this scenario, the rehash_count is still 0.
➢ However, if we insert a new value as the 11th value, the table will rehash itself which makes the rehash_count into one.

As evidenced by the above example, this hash function is suitably effective for its use case - which is short strings of text denoting game objects. Conflicts are minimised - occurring only with very similar input strings. Given the wide variety of materials - and when they do occur, they are efficiently handled with minimal probing. This can be seen above with the probe total of 6 to handle 4 conflicts. Extrapolating this behaviour indicates that with a larger (and therefore more varied) set of inputs, accompanied by a larger table, the proportion of conflicts and the probing needed to resolve them will reduce drastically.