Report

NAME: RISHAB GARG

STUDENT NUMBER :796799

Written Tasks:

Part4) Given below shows some stats calculated for part 4:

Below stats has been calculated with table size 4.

TOTAL SIZE OF TABLE	KEYS INSERTED	NUMBER OF COLLISIONS	average probing length	load factor	total probing length	% keys coolisions
16	10	7	3.3	62.5	33	70
128	96	69	6.5833	75	631.9968	71.875
1024	994	746	17.233	97.0703125	17129.602	75.05030181
2047	1091	792	26.5105	53.271	28922.9555	72.5939505
2047	1495	1070	22.42	72.988	33517.9	71.57190635
2047	2034	1512	44.23	99.3616	89963.82	74.33628319
16384	9954	7084	67.3826	60.75439453	670726.4004	71.1673699
32768	19908	14263	97.037	60.75439453	1931812.596	71.644565
32768	29850	21830	79.3881	91.0949707	2369734.785	73.13232831
65536	39785	28625	101.949	60.70709229	4056040.965	71.9492271
65536	49767	35403	122.75	75.93841553	6108899.25	71.13750075
65536	59697	43565	88.0659	91.09039307	5257270.032	72.97686651
65536	63661	47681	109.7845	97.13897705	6988991.055	74.89828938
131072	69654	51342	197.7713	53.14178467	13775562.13	73.71005255
131072	79579	57012	189.6849	60.71395874	15094934.66	71.64201611
131072	99524	70821	173.0474	75.93078613	17222369.44	71.15972027
131072	109460	78426	182.6344	83.51135254	19991161.42	71.64809063
131072	130453	97885	220.56	99.528	28772713.68	75.03468682
262143	139272	102791	355.4738	53.12825443	49507547.07	73.80593371
262143	179085	126593	257.6688	68.31576659	46144617.05	70.68877907
262143	199047	140938	273.5974	75.93	54458741.68	70.80639246
262143	260823	195648	256.6149	99.496	66931068.06	75.0117896

Number of collisions: It signifies the number of keys for which its first addressed is already occupied. E.g. if single key hashed for first time encounter collision, its counted as one.

Length of Probe sequence: It signifies the average number of probing or steps taken while all keys are inserted at the end.

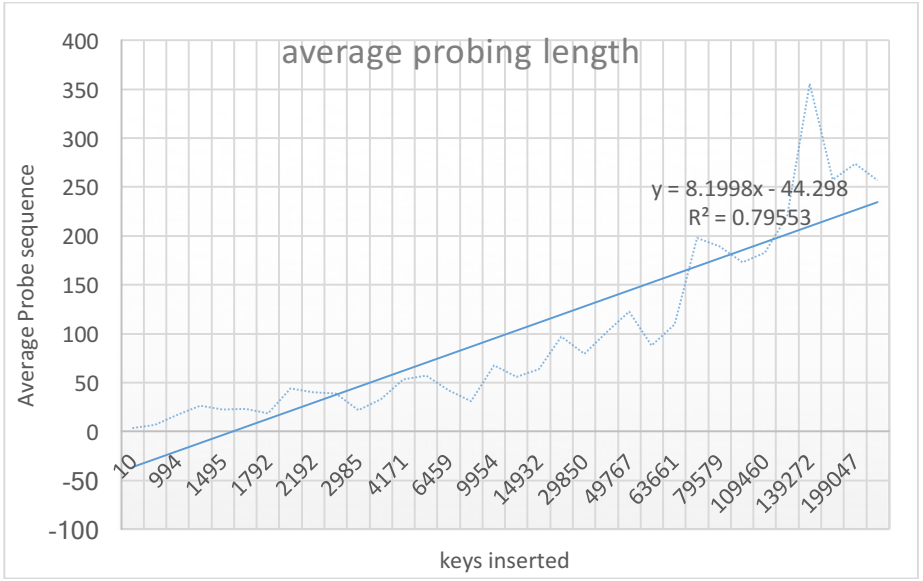
Average probing length: It signifies the total number of probing's made while inserting all keys in total.

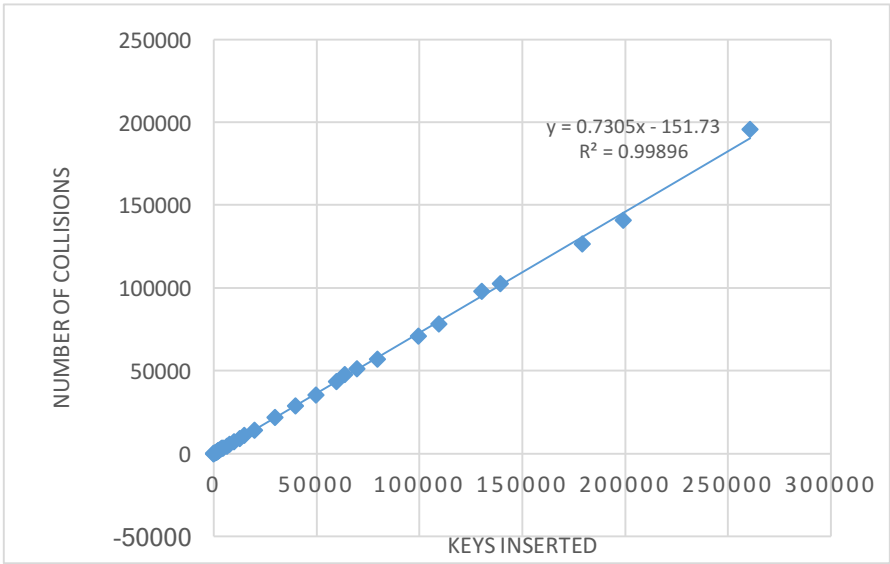
Load factor: It counts the %age of keys inserted in table of size Total size of table.

Highlighted table: It signifies that if we look next row of highlighted row we need to double the table size.

As seen from above table, keys are inserted into table until load factor goes to 100%. After inserting some more keys (as mentioned in highlighted region) we need to double the table size and thus it increases the average probing or total probing even if we increase small number of keys. I concluded it is due to the table size initialization. It starts from 4 and thus keeps on doubling at factor of 2. If we double the table and rehash the keys with %table size, we observe that

Table size is multiple of 2 and thus its hash value distribution for keys is coming quite high in some interval giving high number of steps or probing's to occur to insert the keys. Thus it increases as soon as table gets double.
If we guess in advance the number of keys to get inserted, we may set a combatively high prime number as initial table size, which gives less number of average probing's when table gets double as there will be more space to allocate and less collisions, thus keys will get insert fast.
If table size gets double and we try to insert more keys into it, it's likely for average probing to decrease until some number of keys gets inserted until table gets double again. It is due to less number of probing's as table is half empty until table gets double again and more number of keys inserted. Thus as table gets nearly to 90% load factor, its again high number of probing's then before as few keys gets high probing's done to insert into table.
Summary: Initial table size is set to 4. If table size gets double its multiple of 2 and when we try to rehash all the values again to some intervals. As a result, its high chance to have same hash value again thus its average probing is increase and then it starts to decrease until some keys have inserted and then it increases again due to high number of keys inserted already, giving high chance to have same hash value.
%age of collisions are still nearly same as we insert more number of keys as table is just getting double. It clearly shows that hash value of keys is still same colliding at same rate even if we increase the table size. Thus it holds the statement that number of probing's will also increase as we get nearly same rate of collision, thus if we insert new elements after doubling the table, we have more empty space in our table which decrease the %age of collision rate to some extend and again starts increasing. Total number of collisions are increasing as we are inserting more keys and it age increases as load factor of table goes quite high.





As we may see that the probing increases to some extent when table is got doubled and then it starts decreasing till load factor reaches around 80%. After that, average probing again increases. Sudden jump in average probing sequence signifies the doubling of table when keys are inserted and slow decrease and increase signifies the keys are inserted and table is not getting double.
Collisions are increasing as keys are inserting more and more. Thus if table size is increased then also number of collisions are increasing which gives idea that hash value is nearly same even if we double. slope is constant of 2nd figure. It signifies the collisions rate is same.

Part 5: Keys Per Bucket

Table size	keys per bucket	Total keys inserted	Average key per bucket	Clock ticks	Depth
32	1	10	0.909	40	5
16	2	10	1.667	33	4
8	4	10	2.5	31	3
4	8	10	3.33	29	2
2048	1	100	0.763	484	11
512	2	100	1.429	235	9
128	4	100	2.778	130	7
32	8	100	6.25	120	5
1048576	1	996	0.682	178471	20
262144	2	996	1.372	43050	18
2048	4	996	2.822	1413	11
512	8	996	5.413	1132	9
2097152	1	9942	0.681	414411	21
524288	2	9942	1.371	114858	19
32768	4	9942	2.763	16905	15
8192	8	9942	5.514	10812	13

Depth: It signifies the depth of table
Keys per bucket: It's the bucket size of table bucket
Average keys per bucket: It gives the average number of keys found if we pick any bucket at random.
Total keys inserted: It signifies the total distinct keys inserted into extendible hash table.

If we get collisions and suppose we have keys per bucket 1, we either need to split the table or double it. It depends if depth of table equals to depth that hash address bucket, we need to double the table which increase in more number of clock ticks, but if we would have more number of keys per bucket, we could put that key into that empty slot available which comparatively is faster as there is no splitting and doubling.
If we have multiple keys, there is possibility that while splitting there is high probability of values to get distributes then to rehash to same value.
E.g. if we have 2 keys per bucket and 4 keys per bucket. If we get on the case where our table hash address is full and global depth is not equal to depth of hash value bucket. We try to rehash values again and probability of rehashing 4 values at same slot is less than probability of rehashing 2 keys at same slot. Thus it helps to avoid chance of doubling the table when keys per bucket are more. It hence reduces the clock time also.
As seen from stats. It's clearly seen that if keys per bucket are less and we are inserting same keys to different keys per bucket, the size of table is less if keys per bucket are more. It's because it avoids table from doubling due to factors mentioned above and hence reduces the clock time also.

BONUS

KEYS INSERT	XUCKOO					XUCKOON		
	size of T1, T2	Load facotr T1	Lad factor t2	CPU TIME	size of T1,T2	Load facotr T1	Lad factor t2	CPU TIME
100	256 128	20.703	36.719	3833	8 8	93.75	62.5	1491
497	1024 1024	25.586	22.949	22797	64 32	58.594	76.953	9136
993	8192 4096	6.726	10.791	47603	128 128	53.613	43.359	18874
4971	32768 16384	8.194	13.953	260347	1024 512	35.62	50.122	101203
9955	65536 32768	8.287	13.806	534194	2048 1024	35.358	50.806	223038
14932	65536 65536	12.318	10.466	905396	2048 2048	51.251	39.886	330079
29855	131072 131072	12.376	10.402	1848375	8192 4096	25.493	40.12	660644
49776	524288 262144	5.145	8.698	3403776	8192 8192	44.545	31.407	1134941
79619	1048576 524288	4.113	6.961	5540644	16384 8192	35.541	50.407	1862559
99508	2097152 1048576	2.573	4.345	7302190	16384 16384	44.119	31.799	2311539
149229	2097152 1048576	3.858	6.516	11236583	32768 16384	32.7	48.453	3596425
895676	8388608 8388608	5.787	4.89	86754462	262144 131072	24.159	37.1	22511485

CPU time here refers to CPU ticks.
Xuckoon analyses are done with bucket size equals 8.
As we may observe that in inserting same number of keys, CPU time of XUCKOO is always coming out to be greater than XUCKOON.
While considering Xuckoon , I always tried to split or double the table and immediately check cuckoo again, if it can insert element into empty bucket if all my keys get rehashed to same bucket and while performing cuckoo again, there is chance to insert element back in that empty slot or I try to replace the keys in cuckoo with my helper function called remove_better_key, which removes the key with lease number of values in bucket into another table and thus it increase more possibility to get my values hashed more without doubling table again. While performing XUCKOO I have only one key per bucket, thus even if I get same rehash value chance to fill into that empty bucket is quite low and hence I am doubling time man times as compared to XUCKOON. As a result, xuckoo table is doubling at higher rate than xuckoon and thus taking more CPU ticks each time.
We may conclude that by increasing number of keys per bucket we are increasing the possibility to insert keys fast without doubling and even if keys are doubling, probability to rehash many values at same bucket is less than with one key at same bucket. Moreover, if we have split there are high chance to get value filled in the empty place as best key (mentioned above) from bucket is removed.
As seen from stats CPU time of XUCKOO is always greater than XUCKOON on insertion of same number of keys. Its due to more doubling of tables in XUCKOO due to above factors. Load factor of XUCKOON is quite relatively high than XUCKOO. If we try to insert nearly one million keys, there is huge difference in table depth and CPU time also. Table depth for Xuckoo went to 2^23 while XUCKOON table depth went to 2^18 which signifies there can be more keys inserted easily.