Computer & Systems Engineering Department

Data Structures and Algorithms

Implementing Perfect Hashing [$O(N^2)$ and $O(N)$ space hash tables]

Contributors:

	Name	ID
1	Adel Mahmoud Mohamed Abdelrahman	20010769
2	Amr Ahmed Abdelazim	20011037
3	Marwan Essam Eldin	20011859
4	Mohamed Amr Abdelfattah	20011675
5	Mennat-Allah Mahmoud Saad	19016713

Time complexity of two implementations

Time complexity of the $O(N^2)$ implementation:

Operation	Time Analysis
Insertion	***O (M)*** if rehashing is encountered and ***O (U lg M)*** if not
Deletion	***O (U Lg(M))***
Search	***O (U Lg(M))***

- **Time complexity of the $O(N)$ implementation:**

Operation	Time Analysis
Insertion	***O (U Lg(M) + 2U Lg(M) * m)*** if rehashing is encountered and ***O (U Lg(M) + 2U Lg(M))*** if not
Deletion	***O (U Lg(M) + 2U Lg(M))***
Search	***O (U Lg(M) + 2U Lg(M))***

- **Where (U lg M) is the complexity of getting the index by the hashing matrix such that (h(x) = h. x) and h is a matrix of dimensions (b x u) and x is the binary representation of the (integer)key and its maximum binary length is u = 32 bit.**
- **Where M is the size of the outer hash table and it's fixed, and m is the current size of the 2nd level of each bin and in the worst-case m = M^2 i.e., when all the elements are hashed to the same pin**
- **Note that the time taken for Deletion is constant, but we just need to search first and then delete and the searching complexity depends on the time taken to get the index by (h(x) = h. x).**

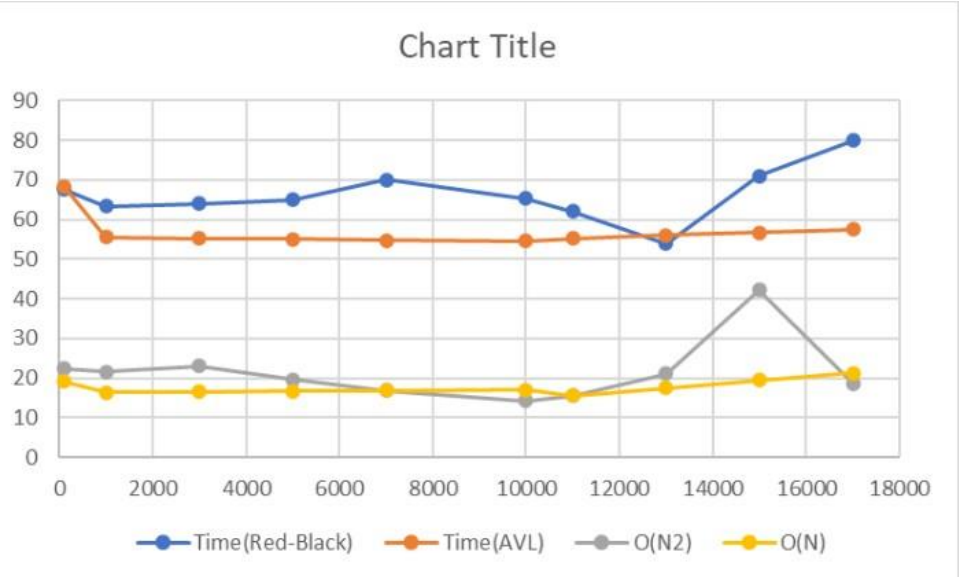
The Space complexity of the two implementations

$O(N^2)$ implementation	***O ($N^2 + Ulg\ M$)*** where ***(U lg M)*** is the size of h, so asymptomatically it's N^2
$O(N)$ implementation	***O (const . N)*** and this constant lies between 3 and 4 which is asymptomatically ***O(N)***

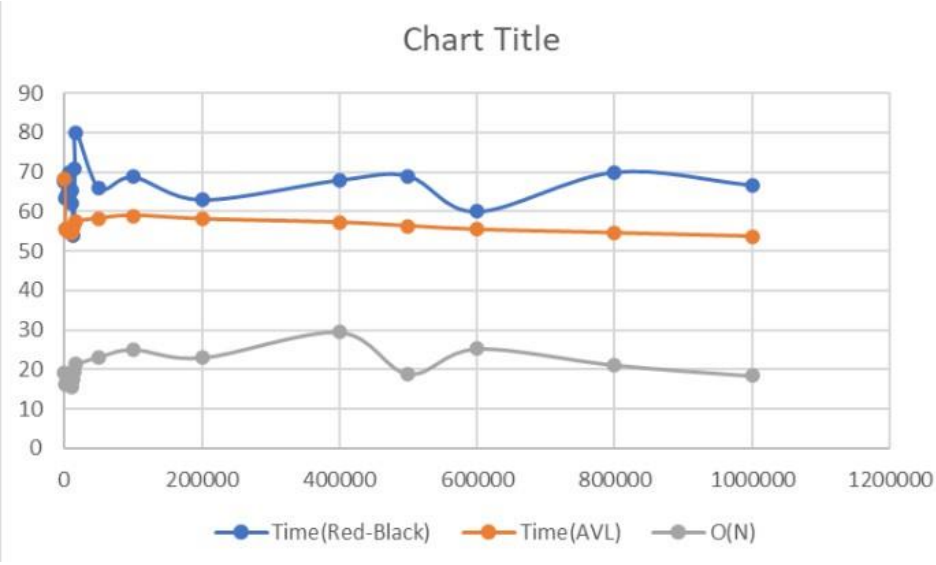
Comparison between the Perfect hashing, AVL, and Red-Black:

With respect to mean time to search:

At small sizes due to $O(N^2)$ limitation.



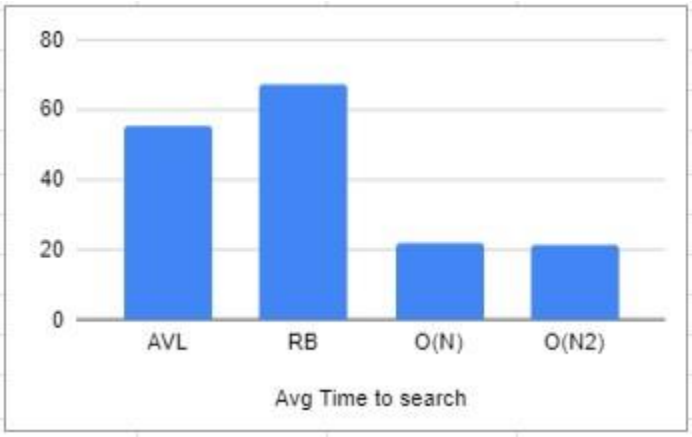
At large sizes.



Size	Time (Red Black)	Time (AVL)	O(N2)	O(N)
100	67.55	68.275	22.4	19.2
1000	63.35	55.525	21.65	16.35
3000	64	55.29375	23.1	16.5125
5000	65	55.0625	19.7	16.675
7000	70	54.83125	16.95	16.8375
10000	65.35	54.6	14.3	17
11000	62	55.32916667	15.65	15.65
13000	54	56.05833333	21.05	17.53
15000	71	56.7875	42.3	19.41
17000	80	57.51666667	18.6	21.29
50000	66	58.24583333	SIZE LIMIT!	23.17
100000	68.775	58.975	SIZE LIMIT!	25.05
200000	63	58.0875	SIZE LIMIT!	23
400000	68	57.2	SIZE LIMIT!	29.4
500000	69	56.3125	SIZE LIMIT!	18.95
600000	60	55.425	SIZE LIMIT!	25.25
800000	70	54.5375	SIZE LIMIT!	21.15
1000000	66.725	53.65	SIZE LIMIT!	18.45

Note that for the perfect hashing of O(N^2) if we tried to make the table size larger than 17,000 then we would get OutOfMemoryError: Java heap space.

Avg Time to search	
AVL	55.5
RB	67.3
O(N)	22.1
O(N2)	21.5



Time comparison for the different operations:

O(N2)	batch insert N2	delete N2	O(N)	batch insert N	delete N
	68326.3	128630.5		108982.2	1897.8
	23578	119484.1		45658.8	448.4
	17472.7	121337		51946.3	1157.9
	17043.9	112058.4		46727.9	903
	18863.7	119129.9		47177.1	322.9
AVG	29056.92	120127.98		60098.46	946

O(N2)	search N2	batch delete N2	O(N)	search N	batch delete N
	38.5	23379.8		22	23503.4
	49.8	22770.1		98	22512.7
	19.9	26577.8		30.8	23859.9
	21.4	18770.4		41.1	27991.1
	21.8	17537.9		19.8	25834
	23.7			23.4	
	20.7			34	
	55.5				
AVG	31.4125	21807.2		38.44285714	24740.22

Collisions Comparison

Run on 15_000 size of the table (N = 15_000).

$O(N^2)$

Words	Time (us)	# collisions	size
7000	41683.8	0	225000000
5000	1146311.7	2	225000000
3000	4262.7	2	225000000
1000	1316.2	2	225000000

O(N)

Words	Time (us)	# collisions	size

7000	61855.4	396	32002
5000	41485.7	846	46600
3000	39465.3	1217	57406
1000	42803.4	1258	61358

Collision of O(N) on the average = zero (collisions / N)

Conclusion:

- **In insertion and deletion, RB, and AVL will beat hash tables at all due to the huge complexity of tables.**
- **Usage of efficient hash table mainly when data is static and doesn't change frequently.**
- **In search, hash tables beat trees as searching in hash table requires (index) only which is O (1).**
- **Due to space, O(N) is the most efficient way to store and search for data.**
- **It is recommended to use trees in case of dynamic data.**
- **It is recommended to use hash tables in case of static data.**