Computer & Systems Engineering Department

Data Structures and Algorithms

Assignment 2: Shortest Paths Algorithms

Name: Ahmed Khalil Ahmed Elzainy ID: 20010087
Name: Abdelrahman Mohamed Mohamed ID: 20010851
Name: Mohamed Elsayed Amin ID: 20011502
Name: Mahmoud Attia Mohamed ID: 20011810

I. Time Analysis:

	HASH_N2
insert	O(M)
search	O($u\ log_2(M)$)
delete	O($u\ log_2(M)$)
batchInsert	O(N*M) .
batchDelete	O(N*$ulog_2(M)$) where N is the size of the list

- Expected insertion time depends on the probability of collisions which is 1/M its O(M*(1/M)) = O(1)
- M is the hash table size
- u*log2(M) (u*b) is the time to calculate the universal hash function.

	HASH_N
insert	O($n_j^2 * log_2(n_j^2)$)
search	O($ulog_2(M) + log_2(n_j^2)$)
delete	O($u\ log_2(M) + log_2(n_j^2)$)
batchInsert(k)	O($n_j^2 * log_2(n_j^2) * k$)
batchDelete(k)	O(($ulog_2(M)$ + log2(n_j^2)) * k)

- M is the hash table size.
- n_j^2 is the size of inner table.
- k is the number of inserted or deleted elements.
- log2(M) (u*b) is the time to calculate the universal hash function.

II. Space Analysis:

HASH_N2	HASH_N
O(N2+ $log_2(M)$) = O(N2) Where $log_2(M)$is the space of the hashing matrix.	O(const*N + $log_2(M)$) = O(N) Where $log_2(M)$is the space of the hashing matrix.

III. Comparison between perfect hashing & AVL & RB trees mean search time (ns):

size	HASH_N	HASH_N2	AVL	RB
10	7440	10150	2472	2434
100	3663	3707	4200	2612
1000	869	1400	3717	2969
5000	937	1324	5330	4824
10000	1475	1971	6142	7292
20000	1058	1471	6301	7227
10^5	1186		7041	9070
5*(10^5)	1270		9244	10987
10^6	1308		7988	10205
Mean Time	**2134**	**3337.17**	**5826.11**	**6402.22**

- Hashing with O(N) and O(N^2) take less search time than trees search on average.

IV. Verify the consumed space:

- O(N2) Space:
  - In the HASH_N2 the space used is for the hash table which is O(M) = O(N2) plus the space used by the universal hash matrix which is O(u*log2(M)) so the space used is O(M+ u*log2(M)) which is O(M) which is O(N2).
  - And this is verified by tests in the next section.

- O(N) Space:
  - we need to prove that:

$$E\left[\sum_{j\equiv 0}^{m-1} n_j^2\right] < 4n$$

where n_j is the number of keys hashing to slot j .
We start with the following identity, which holds for any nonnegative integer a:

$$a^2 = a + 2\binom{a}{2}.$$

We have

$$\begin{aligned}
&\mathrm{E}\left[\sum_{j=0}^{m-1} n_j^2\right] \\
&\quad = \mathrm{E}\left[\sum_{j=0}^{m-1} \left(n_j + 2\binom{n_j}{2}\right)\right] \\
&\quad = \mathrm{E}\left[\sum_{j=0}^{m-1} n_j\right] + 2\,\mathrm{E}\left[\sum_{j=0}^{m-1} \binom{n_j}{2}\right] \\
&\quad = \mathrm{E}[n] + 2\,\mathrm{E}\left[\sum_{j=0}^{m-1} \binom{n_j}{2}\right] \\
&\quad = n + 2\,\mathrm{E}\left[\sum_{j=0}^{m-1} \binom{n_j}{2}\right]
\end{aligned}$$

To evaluate the summation $\sum_{j=0}^{m-1} \binom{n_j}{2}$, we observe that it is just the total number of collisions. By the properties of universal hashing, the expected value of this summation is at most

$$\binom{n}{2}\frac{1}{m} = \frac{n(n-1)}{2m} = \frac{n-1}{2},$$

since $m = n$. Thus,

$$\text{then } E\left[\sum_{j\equiv 0}^{m-1} n_j^2\right] < n + 2 * \frac{n-1}{2}$$

$$< 4 * n$$

- And this is verified after insertion by function:

```
1 usage   Ahmedkhalilelzainy
private boolean checkCorrectness(){
    boolean flag = true;
    int sigmaN = 0, sigmaNSquared = 0;
    for(int i = 0; i< entrySizes.length; i++){
        sigmaN += entrySizes[i];
        sigmaNSquared += entrySizes[i]*entrySizes[i];
    }
    return sigmaNSquared<4*sigmaN;
}
```

and if the total space exceed this expected space we make total rehash:

```
private void TotalRehash(){
    do {
        Tables = new Long[totalNumKeys][];
        entrySizes = new int[totalNumKeys];
        L1HashEdit();
        insertionDirector(allElements.toArray(new Long[0]));
        numOfOuterTableCollisions++;
    }
    while(!checkCorrectness());
}
```

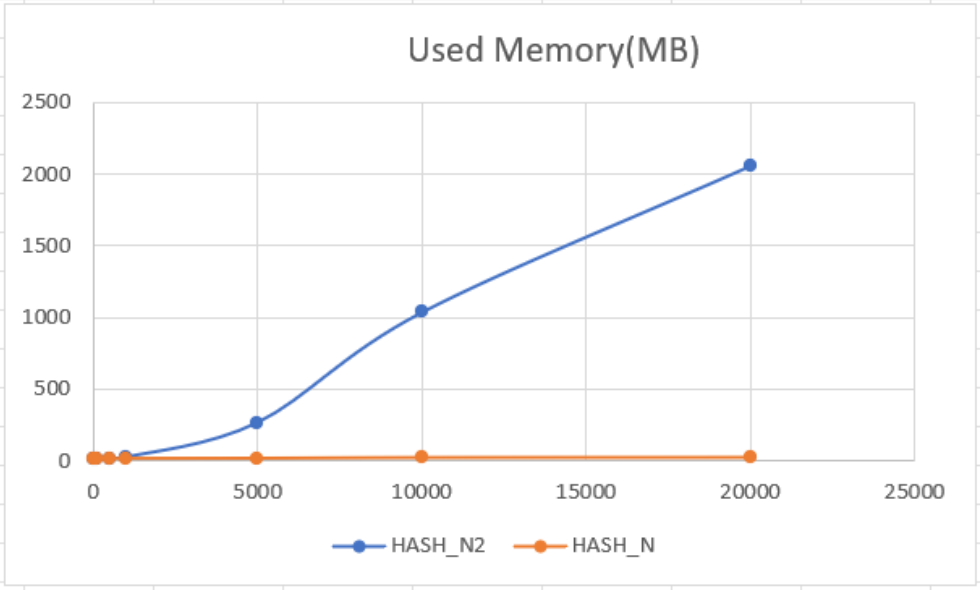
V. Comparison between HASH_N2 & HASH_N:

- **Space (number of hash table slots):**

	HASH_N2	**HASH_N**
10	128	14
100	16384	176
500	262144	994
1000	1048576	2108
5000	33554432	8042
10^4	134217728	16172
2*(10^4)	536870912	32016
	O(N2)	O(N)

- **Memory used in(MB)**

size	HASH_N2	HASH_N
10	14	14
100	15	15
500	17	16
1000	22	18
5000	265	19
10000	1036	23
20000	2062	24



- **Number of rebuilds:**

	HASH_N2	**HASH_N**
10	0	4
100	0	37
500	1	204
1000	0	442
5000	0	1546
10^4	0	2958
2*(10^4)	0	5986

- **Notes:**
  - Space used by the HASH_N of course is less than that of HASH_N2 so it's more efficient in terms of space.
  - The search time of HASH_N is less than that of HASH_N2 so it's more efficient.
  - On average trees search time is more than that of hash tables. So, hash tables are more efficient in searching than trees.
  - Generally, hash tables are better in the case of a static dictionary and searching on it (and that's how perfect hashing really works) but trees are better in the case of a dynamic dictionary.
