Report on Hashtable Implementation

Rawnak Ahmed Turjo

2205055

The first hash function (Hash1) is djb2 hash function. The cpp code for used function is:

```
unsigned long long hash1(const string &s)
   {
      unsigned long long hashvalue = 5381;
      for (char c : s)
      {
         hashvalue = ((hashvalue << 5) + hashvalue) + c;
      }
      return (hashvalue % capacity);
   }
```

This initializes the hash value to 5381 and updates for each character as hashvalue = (hashvalue * 33) + c, where hash * 33 is (hash << 5) + hash. This is a simple hash function with single line update and very fast due to bit shift and addition. The prime constant 5381 and multiplication by 33 give decent collision resistance for small to medium datasets,reducing clustering in hash tables.

The second hash function is SDBM(Simple Database Manager) hash function which was originally used on the SDBM open-source database system.The c++ code for this function is:

```
unsigned long long hash2(const string &s){

      unsigned long long hashvalue = 0;

      for (char c : s)

                  hashvalue = c + (hashvalue << 6) + (hashvalue << 16) - hashvalue;

        return hashvalue % capacity

 }
```

This function is known for its simplicity and good distribution properties. The single-line update (hashvalue = c + (hashvalue << 6) + (hashvalue << 16) – hashvalue) is minimal, so it is easy to implement,

There are 6 tables for load factor 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9.

Load factor = 0.4

0Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertion	Before Deletion		After Deletion		# of Collisions during insertion	Before Deletion		After Deletion	
		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes
Separate Chaining with balanced BST	714	0.21	N/A	0.21	N/A	734	0.25	N/A	0.18	N/A
Linear Probing with Step Adjustment	1323	0.09	1. 27	0.189	2.53	1417	0.12	1.28	0.18	2.55
Double Hashing	1106	0.13	1.28	0.237	2.17	1164	0.17	1.23	0.22	2.1

Load factor = 0.5

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertion	Before Deletion		After Deletion		# of Collisions during insertion	Before Deletion		After Deletion	
		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes
Separate Chaining with balanced BST	1058	0.20	N/A	0.23	N/A	1099	0.169	N/A	0.20	N/A
Linear Probing with Step Adjustment	2377	0.13	1.46	0.22	3.12	2725	0.09	1.54	0.20	3.19
Double Hashing	1834	0.17	1.33	0.24	2.34	2009	0.13	1.38	0.23	2.51

Load factor = 0.6

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertion	Before Deletion		After Deletion		# of Collisions during insertion	Before Deletion		After Deletion	
		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes
Separate Chaining with balanced BST	1475	0.18	N/A	0.32	N/A	1542	0.13	N/A	0.21	N/A
Linear Probing with Step Adjustment	4190	0.11	1.79	0.28	3.76	4681	0.1	1.85	0.21	4.09
Double Hashing	3057	0.15	1.48	0.34	2.7	3230	0.13	1.52	0.22	2.9

Load factor = 0.7

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertion	Before Deletion		After Deletion		# of Collisions during insertion	Before Deletion		After Deletion	
		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes
Separate Chaining with balanced BST	1972	0.17	N/A	0.23	N/A	2009	0.16	N/A	0.31	N/A
Linear Probing with Step Adjustment	7760	0.12	2.01	0.25	5.58	8263	0.11	2.17	0.28	5.49
Double Hashing	4981	0.16	1.76	0.27	3.5	5145	0.14	1.7	0.325	3.57

Load factor = 0.8

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertion	Before Deletion		After Deletion		# of Collisions during insertion	Before Deletion		After Deletion	
		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes
Separate Chaining with balanced BST	2492	0.16	N/A	0.24	N/A	2541	0.33	N/A	0.28	N/A
Linear Probing with Step Adjustment	14832	0.12	2.8	0.32	10.57	15035	0.27	2.98	0.35	9.21
Double Hashing	8080	0.15	1.87	0.31	4.615	8124	0.26	2.065	0.34	4.54

Load factor = 0.9

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertion	Before Deletion		After Deletion		# of Collisions during insertion	Before Deletion		After Deletion	
		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes		Avg Search Time (µS)	Avg Probes	Avg Search Time (µS)	Avg Probes
Separate Chaining with balanced BST	3038	0.19	N/A	0.27	N/A	3140	0.17	N/A	0.35	N/A
Linear Probing with Step Adjustment	36871	0.18	4.86	0.6	27.13	35905	0.21	6.15	0.8	30.92
Double Hashing	13998	0.18	2.44	0.36	7.2	14215	0.18	2.39	0.53	7.83

For all load factors, the number of collisions during insertion is highest in linear probing. In separate chaining (using red black tree), the collision is lowest because the collision is counted when there is a key-value pair already existing in specified index. On the other hand, in linear probing and double hashing, the collision is counted when the position directed by hash is occupied by other valid key-pair value in every step.

As load factor increased, the number of collisions also increased due to high clustering. Initially when the load factor is low, the average search time is lowest in linear probing and highest in separate chaining because of cache efficiency. Linear probing has array slots in predictable pattern (index + i * S), which aligns well with CPU cache lines. When the

load factor is high, the search times in these three implementations are not very much different.

For average probes, the linear probing method has higher average probes than the double hashing because double hashing ensures lower collision due to better distribution of key-value pairs. After deletion, the count of average probe increased in linear probing and double hashing. The difference between the average probe count after deletion and before deletion increased as load factor increased.