DSA I Lab 8: Hash Tables

Christine Baca [cab8xd]

[1] University of Virginia, Charlottesville VA 22904, USA

Abstract. The experiment investigated implementation and runtimes of hash tables in Java. The first experiment compares the no. of collisions and runtimes of two collision resolution strategies: separate chaining via linked lists and separate chaining using array lists. The second experiment optimizes the world solver program by implementing the both hash table structures from experiment one and compares the resulting runtimes. The first experiment's results demonstrate that the ArrayList and LinkedList separate chaining methods had similar values for both insert runtime and collisions. The second experiment's results show that the ArrayList chaining ran faster than the LinkedList chaining, likely due to the simplicity the ArrayList add method in comparison the Linked List's complex re-referencing processes for adding nodes and the ArrayList's significantly better compatibility with cache.

Keywords: hash tables, hash functions, separate chaining, collisions, Java, linked lists, hash nodes, runtimes, array lists (vectors)

The data structure of hash tables implements an array abstract data type that maps keys and values by the use of a hash function that indexes into different slots or buckets in the array based on the given key value. The lab explored the efficiency of hash table collision resolutions and runtimes of different word scramble algorithms.

1.1 Experiment One: Collision Resolution

The first experiment tested the efficiency of two different collision resolution strategies. Both strategies implement separate chaining but with different data structures. The first strategy stores items with the same keys and values in 'buckets' through the implementation of linked lists. The second strategy acts the same but employs separate chaining with ArrayLists. The number of collisions is predicted to be similar but the runtime of the ArrayList should prove slightly faster as arrays fit better into cache, making the process of adding into the array is simpler than the linked list add implementation. Below shows the test code used to run trials.

```
//Experiment One Testing Code:
hash.HashTable<Integer, Integer> stud = new
hash.HashTable<Integer, Integer>();
hash.HashTable<Integer, Integer> stud = new
hash.HashTable<Integer, Integer>();

System.out.print("Inserting...");
      for (int i = 1; i < NUM_TESTS + 1; i++) {
                  t = (int) System.currentTimeMillis();
                  for (int j = 0; j < NUM_EL; j++) {

                        Integer key = (int) (Math.random() *
                        NUM_TESTS);
                        Integer value = (int) (Math.random() *
                        NUM_TESTS)
                        stud.insert(key, value);
            }

time = ((int) System.currentTimeMillis()) - t;
System.out.println("----Trial " + i);
System.out.println("Collisions " + stud.getCollisions() + "Time:
" + time);
System.out.println();

System.out.println("DONE");
            }
```

1.2 Experiment Two: Word Scramble Solver Optimization

The second experiment compares the runtimes of two solvers of a word scramble. The first solver uses the hash table's contain methods to detect words. The second strategy uses the hash table structures from experiment one to run the word scramble program and further compare implementations of the linked list and array list separate chaining strategies. The lab predicts that the array list implementation will run faster than the linked list because the array list will operate a faster find due to the greater efficiency of traversing array lists indices than traversing the node references in linked lists. Below shows the test code used to run trials.

```
//Experiment Two Testing Code:
for (int trials = 10; trials > 0; trials--){
      int t = (int) System.currentTimeMillis();
      String word;
      for (int len = 3; len <= 23; len++) {
      for (Direction d :Direction.values()) {
      for (int r = 0; r < grid.length; r++) {
      for (int c = 0; c < grid[r].length; c++){
            if (wl.contains(getWordInGrid(grid, r, c, d, len))) {
                  if ((((wl.retrieve(getWordInGrid(grid, r, c, d,
                  len)).length())) == len)) {
                        p.add(wl.retrieve(getWordInGrid(grid, r, c,
                        d, len)));
                        dir.add(d);
                        row2.add(r);
                        col2.add(c);
            }
      }}}}}}
      Iterator<String> itp = p.iterator();
      int time = ((int) System.currentTimeMillis()) - t;
```

2 Experiment:

The procedure for the experiment runs through numerous trials of their respective algorithms and data sets, recording the runtime of the sorts. The average and standard deviation of the trials will be calculated and recorded. The data will be analyzed based on the trial's runtimes, collisions (experiment one only), averages, and standard deviations. The experiment table lists the number of trials, runs, and additional information regarding the experimental procedure.

2.1 Experiment Table:

	Exp. 1	**Exp. 2**
No. of Trials	10	10
Number of Elements inserted in Hash Table	10,000	62,500
Dictionary txt Used	NA	250x250.txt
Word txt Used	NA	words.txt

3 Results

3.1 Table Results

Experiment One: Separate Chaining Runtimes & Collisions

Runtimes:

Trials	Linked List	ArrayList / Vectors
1	579	525
2	610	509
3	525	528
4	524	520
5	526	518
6	524	512
7	569	513
8	524	512
9	525	517
10	525	513
Average	520	520
Standard Deviation	31.28	6.147

Collisions:

Trials	Linked List	ArrayList / Vectors
1	9999990	9999990
2	10000000	9999990
3	10000000	9999990
4	10000000	9999990
5	10000000	9999990
6	10000000	9999990
7	10000000	9999990
8	10000000	9999990

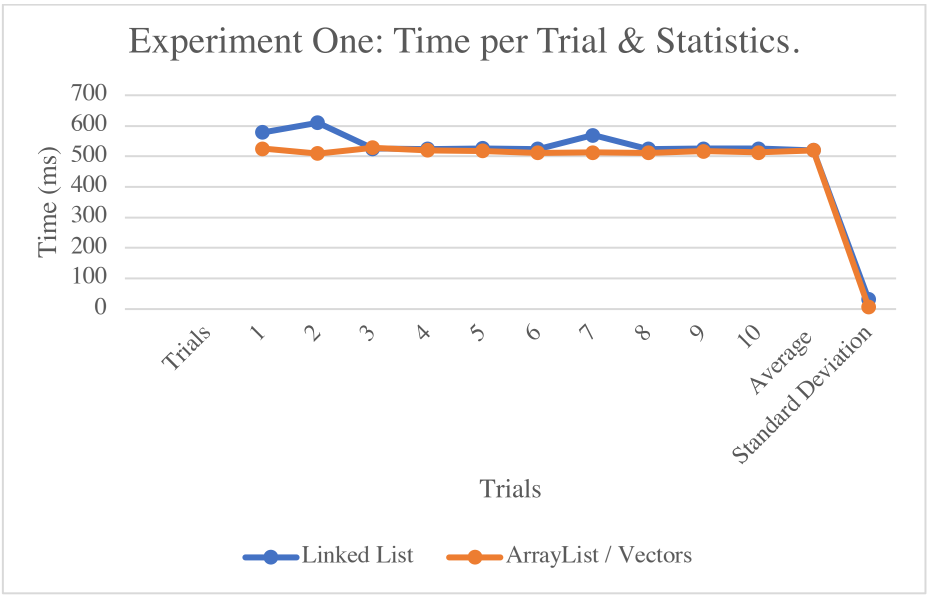
9	10000000	9999990
10	10000000	9999990
Average	10000000	9999990
Standard Deviation	3.02	0

Experiment Two: Word Scramble Runtimes

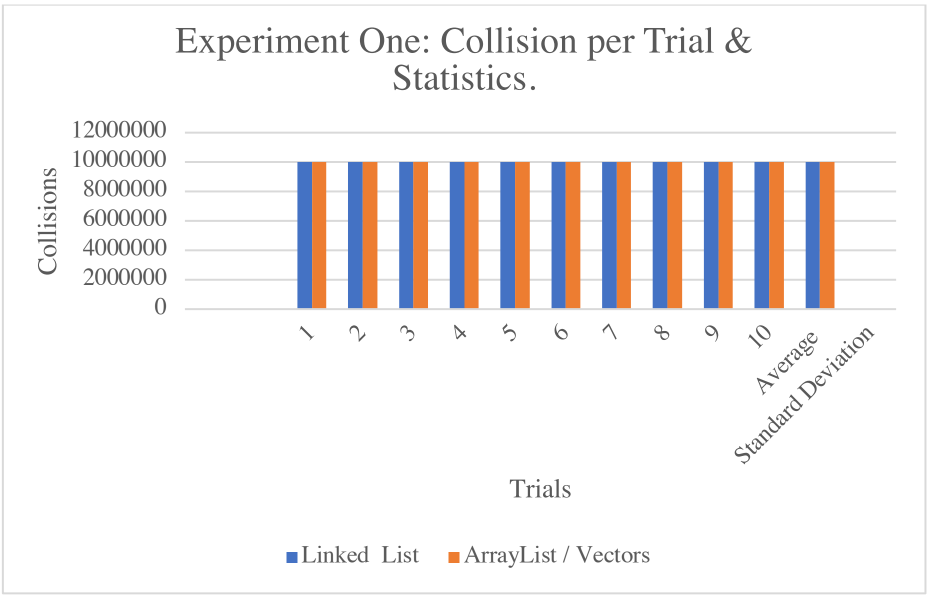
Trials	Linked List	ArrayList / Vector
1	8519	6706
2	6663	5910
3	8350	5938
4	7224	6324
5	7043	5898
6	7205	5882
7	6552	6019
8	6750	5968
9	6723	6170
10	6656	5925
Average	7168.5	6074
Standard Deviation	708.0353491	262.6434

3.2 Graph Results

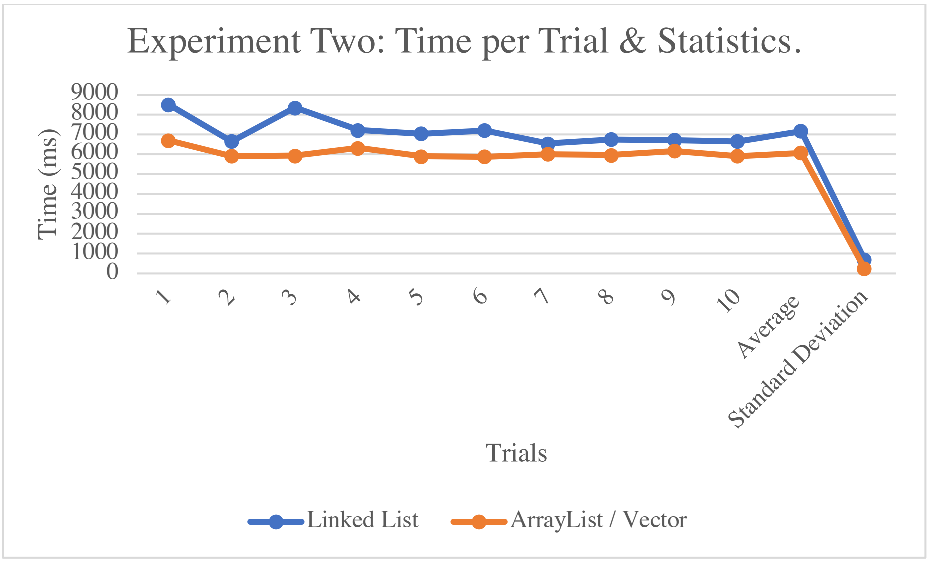
Experiment One: Separate Chaining Runtimes & Collisions
Runtimes:


Experiment One: Time per Trial & Statistics.
Time (ms)
700
600
500
400
300
200
100
0
Trials
1
2
3
4
5
6
7
8
9
10
Average
Standard Deviation
Trials
Linked List
ArrayList / Vectors

Collisions:


Experiment One: Collision per Trial & Statistics.
Collisions
12000000
10000000
8000000
6000000
4000000
2000000
0
1
2
3
4
5
6
7
8
9
10
Average
Standard Deviation
Trials
Linked List
ArrayList / Vectors

Experiment Two: Word Scramble Runtimes



4 Conclusion

The experiment tested the different implementations of separate chaining in hash tables. Linked list and array list separate chaining were compared in two sub-experiments. The first experiment compared the insert runtime and collisions of two hash table structures that implemented the different separate chaining strategies. The average runtimes of the two implementations proved the same; although, the linked list hash table had a much greater deviation than that of the array list's. The collision averages were roughly the same. The second experiment tested the hash table's runtimes in a word scramble program that employed the data structure's find and contain methods. The array list proved to solve the word scramble faster than the linked list with a significantly smaller standard deviation. This conclusion may be due to the array list's higher compatibility with cache and its ability to more quickly traverse its structure in search of relevant data (in contrast with traversing a linked list's node references). Errors include faulty programming that may have skewed data results. Further experiments could include implementations of probing in comparison with separate chaining or further experiments of separate chaining with binary trees.