Verifying the Worst-Case Time Complexity of Hash Tables

{ Ashley Bruce } University of California, Santa Barbara

UC SANTA BARBARA

Objective and Goals

Hash tables allow for constant time access, additions, and deletions in the best-case and average-case examples. Worst-case time complexity is generally not considered, as it is uncommon for hash tables to be given input that will trigger this. But, if one were to provide a hash table implementation, would it be possible to synthesize input that would trigger the worse-case time complexity?

This experiment looks into this problem and sets out to generate input that satisfies worst-case time complexity. When given an implementation of a hash table, this program should print a counterexample, giving input that makes addition have a linear time complexity.

Future Work

Because this project was conducted within the timeline of this course, there is no future work on this project planned. But, the project could be generalized to look at other examples in the future.

While this project only looked at verifying the worst-case time complexity of hash-tables, increasing the scope could include:

- Looking into verifying the best-case complexity
- Looking into verifying the time complexities of other algorithms

Acknowledgments

This project was carried out for the CS292C class under instruction of Professor Yu Feng.

Challenges

- ***Hash Table Implementation:*** Creating a hash table implementation in Racket that is compatible with symbolic input
- ***Verify Function:*** Determining and writing the functions necessary to verify worst case time complexity
- ***Rosette:*** Utilizing solver-aided tools embedded in the Rosette language to verify the program and generate a counter-example

Solution

By using solver-aided language Rosette within Racket, it is possible to:

- Create a hash-table implementation compatible with symbolic input
- Make an assertion excluding worst-case input from the hash table
- Use the verify tool to check the correctness of this assertion and generate a counterexample

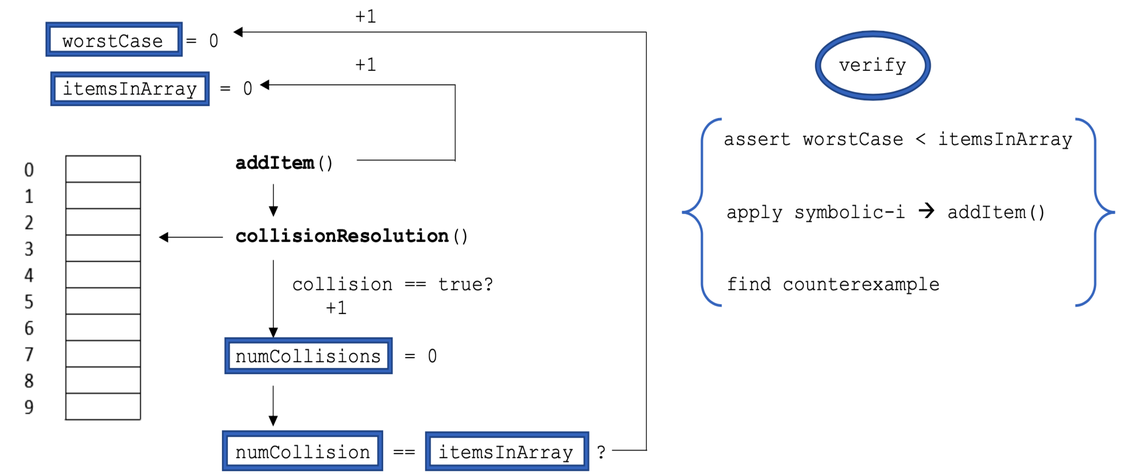
Knowing which input triggers worst-case time-complexity lets us evaluate the hash functions being used. Especially when input is known ahead of time.

References

[1] Landis, Steven. Serval Instruction Counting, GitHub repository. 2020

[2] Gouw, S., Boer, F., Bubel, R., Hahnle, R., Rot, J., Steinhofel, D. Verifying OpenJDK's Sort Method for Generic Collections. In *Journal of Automated Reasoning 2017*

Approach Overview



Approach by Example

```
Integer [] itemsAndWorstCase = {0,0};
// itemsPlaced index 0
// worstCase index 1
```

Global variables

```
#lang rosette

(define itemsInArray 0)
(define worstCase 0)
(define counter 0)


  ; Hash function
  (define (hashFunction key)
    (abs (remainder key arraySize))
    )

  ; Places the key in the index and deals with any collisions
  (define (collisionResolution index key)

      (define num (vector-ref v index))

      (cond
         [(= num key)
          (set! sameVal (addOne sameVal))]
         [(= -1 num)
           (insert index key)
           (set! itemsInArray (addOne itemsInArray))
           (isWorstCase)
           (set! counter 0)
          ]
         [else
           (set! counter (addOne counter))

           (collisionResolution (addOne index) key)
           ]
    ))

  ; Hashes a key to an index
  (define (hash key)
    (collisionResolution (hashFunction key) key)
    )
```

```
private static void collisionResolution(String key, String[] hashArray, Integer [] IAWC) {

    boolean notPlaced = true;
    int counter = 0;
    int hashedIndex = Math.abs(key.hashCode()) % (hashArray.length);

    int itemsInArray = IAWC[0];
    int worstCase = IAWC[1];


    while (notPlaced) {
        int currentIndex = Math.abs((hashedIndex + counter) % hashArray.length);
        if (hashArray[currentIndex] != null) {
            counter++;
        } else {
            hashArray[currentIndex] = key;
            IAWC[0] = ++itemsInArray;
            if (++counter == itemsInArray) {
                IAWC[1] = ++worstCase;
            }
            notPlaced = false;
        }
    }
}
```

Hash Implementations

```
(define (lessThanWorstCase)
  (assert (< (worstCase arraySize)))
  )
```

Verify → Counterexample