UPPSALA UNIVERSITET

# Algorithms and Datastructures Assignment 3

Alexander Sundquist, Pontus Björklid, Vilmer Olin & Erik Hennings

17 oktober 2022

## 2 Hash Tables

### 2.2 Implementation

See attached files for source code.

### 2.3 Tests

1. Hash function 1 performs very well n this case. It evenly distributes random values between all 11 buckets, which is desirable. The hash function performs well because 37 and 11 are both prime numbers. Prime numbers perform well because they do not contain multiples of numbers, different from prime number itself. This means that when you take *mod* it will not hash to those multiples.

2. Hash function 2 does not perform well in this case, as it maps all random elements to buckets with even indices. The hash function only maps to even buckets, because 30 and 8 are both even numbers. When you take *mod* of any number with this hash function the remainder will always be even, and therefore the function will only map to even numbers.

3. Hash function 3 does not perform well in this case, as it does not map to all slots. It only maps to slots in the set $\{0, 1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\}$. To analyse this behaviour the function `testquadratic` was written. When running the function it is clear that the remainder from $e^2 \mod 23$ goes in a the same repeating pattern only giving remainders from the mentioned set of outputs.

4. Hash function 1 performs very poorly in this case, because it maps all the values, of the set, to bucket 4. It does this because all the values are some multiple, $m$, of 11 plus 1. When the values that are being mapped are some multiple of the modulo it will always map to a distinct bucket.

5. The function `randstring(n)` returns a list of length n with a random combination of the strings 'TA', 'AT', 'GC', 'CG'.

   The python function `ord()` returns an integer representing the Unicode character. The entire statement, `sum(map(ord,e))`, of hash function 4, will return the summation of all the integers returned by the `ord()`-function.

   The values returned by `ord()` for the possible letters from `randstring(n)` are displayed in table 1. The summation of the possible combinations of the letters are displayed in table 2.

   From table 2 it is clear that the values for AT and TA give the same value, as well as for CG and GC. This implies that `randstring(n)`, which returns a string with 10 sets of these combinations, can only return 11 unique valued strings. This behaviour is displayed in table 3. Because, the summation in the hash function only results in 11 unique values the hash function will only map to 11 slots, which is consistent with the tests. Therefore it does not perform that well in this case.

   Table 1: letters and their value following the use of `ord()` on the letter.

| e | ord(e) |
|---|---|
| A | 65 |
| T | 84 |
| G | 71 |
| C | 67 |

Table 2: The different combinations that `randstring(n)` can output and the combinations' value following the use of `ord()`.

| Number | ord(e) |
|---|---|
| AT | 149 |
| TA | 149 |
| GC | 138 |
| CG | 138 |

Table 3: the number of combinations `randstring(10)` can output that provide the same value.

| Number of AT/TA | Number of CG/GC | Summation value |
|---|---|---|
| 0 | 10 | 1380 |
| 1 | 9 | 1391 |
| 2 | 8 | 1402 |
| 3 | 7 | 1413 |
| 4 | 6 | 1424 |
| 5 | 5 | 1435 |
| 6 | 4 | 1446 |
| 7 | 3 | 1457 |
| 8 | 2 | 1468 |
| 9 | 1 | 1479 |
| 10 | 0 | 1490 |

# 3 Binary Search Trees

This part of the assignment was not completed.

# 4 Work distribution

We worked on the assignment together. We sat in our group and coded on different computers while discussing the problems together.