**Mason Shepherd**

**Homework 7: due March 15th 11:59PM.**

R-10.6 [10 points] Draw the 11-entry hash table that results from using the hash function, *h*(*i*) = (3*i*+5) mod 11, to hash the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, assuming collisions are handled by chaining.

Ans:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| ↓ | ↓ |  |  |  | ↓ |  |  | ↓ | ↓ | ↓ |
| 13 | 39 |  |  |  | 11 |  |  | 23 | 5 | 20 |
|  | 94 |  |  |  | 88 |  |  | 12 | 16 |  |
|  |  |  |  |  | 44 |  |  |  |  |  |

R-10.7 [10 points] What is the result of the previous exercise, assuming collisions are handled by linear probing?

Ans:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | 94 | 39 | 16 | 5 | 44 | 88 | 11 | 12 | 23 | 20 |

R-10.9 [10 points] What is the result of Exercise R-10.6 when collisions are handled by double hashing using the secondary hash function *h*′(*k*) = 7 − (*k* mod 7)?

Ans:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | 94 | 23 | 88 | 39 | 44 | 11 | 5 | 12 | 16 | 20 |

k = 5 is sent through an infinite loop w/ this double hash method which will only probe the non-odd index positions [0, 2, 4, 6, 8, 10] … since the only remaining position is odd, it would never be mapped to A[7] unless an exception was thrown for too many iterations of the double hash.

R-10.10 [5 points] What is the worst-case time for putting *n* entries in an initially empty hash table, with collisions resolved by chaining? What is the best case?

Ans:

Worst-case: O(n) // all entries are mapped to same bucket

Best-case: O(1) // load factor is *n*/*N*. No collisions.

Worst-case: O(n2) // all N map positions & n list positions

Best-case: O(n) // uniformly distributed hash table N pos.

R-10.17 [5 points] Explain why a hash table is not suited to implement a sorted map.

Ans:

If it is already sorted, there already exist a few ways to search through the map efficiently without hashing, which would yield little to no advantages for the amount of work required for such an implementation.

Hash table uses hash functions to evenly distribute entries in pseudo-random order…

R-10.23 [10 points] Draw the result after performing the following series of operations on the skip list shown in Figure 10.13: remove(38), put(48, *x*), put(24, *y*), remove(55). Use an actual coin flip to generate random bits as needed (and report your sequence of flips).

Ans:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S5 | **-∞** |  |  |  |  |  |  |  |  |  |  | **+∞** |
| S4 | **-∞** |  | **17** |  |  |  |  |  |  |  |  | **+∞** |
| S3 | **-∞** |  | **17** |  | **(24,y)** |  |  | **42** |  |  |  | **+∞** |
| S2 | **-∞** |  | **17** |  | **(24,y)** | **31** |  | **42** |  | **(48,x)** |  | **+∞** |
| S­1 | **-∞** | **12** | **17** |  | **(24,y)** | **31** |  | **42** | **44** | **(48,x)** |  | **+∞** |
| S0 | **-∞** | **12** | **17** | **20** | **(24,y)** | **31** | **39** | **42** | **44** | **(48,x)** | **50** | **+∞** |

Coin flip for put(48, x): heads, heads, heads, tails

Coin flip for put(24, y): heads, heads, tails

**Projects**

[10 points]Implementa program that uses a hash table to remove duplicates in an array of numbers. You can utilize the hash table that was already implemented in Java (or other programming languages if you don’t use Java) - it is not required to implement your own hash table. Compile and test your code. Turn in your complete Java code as well as your test data and the output of your program.

**Grading:**

Programs compile 50%

Programs give correct output: 50%