CS 303 Algorithms — Prof. Lewis
due wed oct 1, 2014

H5.

Yingzhu (Jacqueline) Zhang

Students I have consulte with for this assignment:
1. Diego Lanao
2. Liam Bench
3. Jake Shor
4. Nate Marshalls
5. Maurice Hayward
6. Tyler Reid

1. 3.3.10 **Solution:**

   (See attached paper.)

2. 3.3.11 **Solution:**

   (See attached paper.)

3. 3.4.1

   Solution:

key	E	A	S	Y	Q	U	T	I	O	N
value	5	1	19	25	17	21	20	9	15	14
hash	0	1	4	0	2	1	0	4	0	4

   * We hash by using hash function $11k\%M$, where $M = 5$
   ∴ The table index is:

0	1	2	3	4
E	A	Q		S
Y	U			I
T				N
O				

4. 3.4.6

   Solution:
   Proposition: For a modular hash function with prime M, two keys that are interger differing by 2^p with $p \in N$ have different hash values.
   Proof :
   We prove by contradiction.
   Suppose the proposition above is false, which is, for a modular hash function with prime M, two keys that are interger differing by 2^p with $p \in N$ have the same hash values
   Then,

$k\%M = (k - 2^p)\%M$
For key a, we have:
$k = aM + r, a \in \mathbb{N}$
For key b, we have:
$k - 2^p = bM + r, b \in \mathbb{N}$
$aM + r = bM + r + 2^p$
$(a - b)M = 2^P$

If $(a - b)M = 2^P$, both sides would have to contain M. However, this is not possible for the right side of the equation given that $M \neq 2$. This is because 2^p only gives prime factorization with values of 2 without M.
Following that, the contradiction of the proposition is false.
$\therefore$ The proposition "For a modular hash function with prime M, two keys that are interger differring by 2^p with $p \in N$ have different hash values" is *true*.

5. 3.4.7

Solution:
Proposition: Implementing modular hashing for integer keys with the code $(a \times k)\%M$, where a is an arbitrary fixed prime, will not mix up the bits sufficiently well that we can use nonprime M.
Proof :
We prove this by example.
We first randomly select some number, $25, 37, 48, 91, 145$; and sets a to be a prime number 2, M to be 4 (or 2^2)
Then, the table index would be:

0	1	2	3
48		25	
		37	
		91	
		145	

$\therefore$ As shown above, implementing modular hashing for integer keys with the code $(a \times k)\%M$, where a is an arbitrary fixed prime, will not mix up the bits sufficiently well that we can use nonprime M.

6. 3.4.13

Solution:
Scenario a leads to expected linear running time for a random search hit in a linear-probing hash table.
Explanation :
If all keys hash to the same index, then the ith key inserted requires i times of loopups to be found.
Because the probability of looking up ith key is $1/n$, as it is random,

$\therefore$ All keys hash to the same index results into an expected linear running time.

7. 3.3.15

   Solution:

   $\frac{N(N+1)}{2}$

   Proof :
   To insert N keys into an initially empty table using linear probing with array resizing, we are increasing the size of the array. For instance, we start with $i[0]$ and add to $i[1]$. As we keep incrementing the index, we are also incrementing the number of compares required in such a fashion:

item	1	1	1	...
index	0	1	2	...

   , which results $1, 2, 3, ..., \frac{N(N+1)}{2}$ compares.