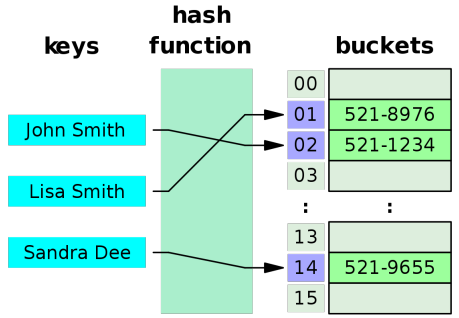


Hashing
Lecture 10

COSC 242 – Algorithms and Data Structures

Today’s outline

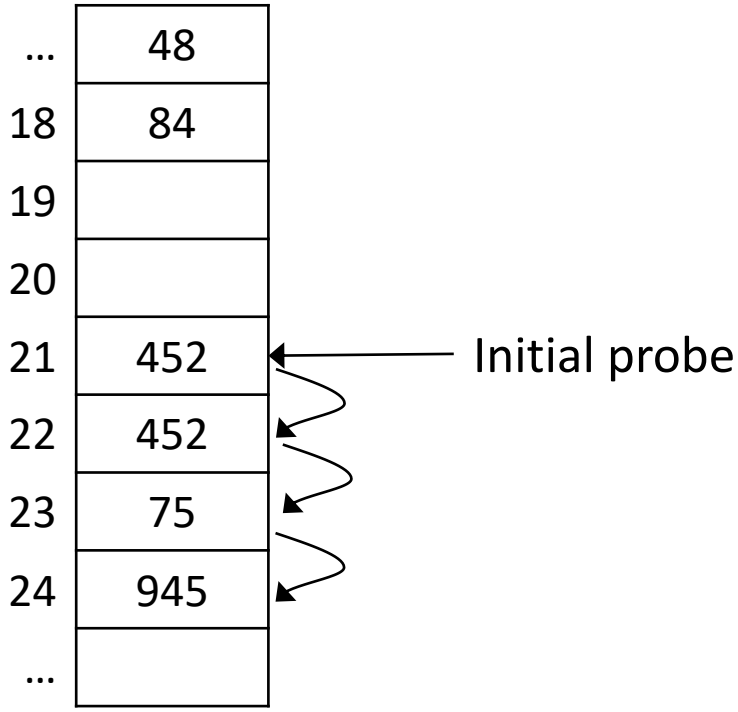
1. Double hashing
2. Chaining
3. Universal hashing

Today’s outline

1. Double hashing
2. Chaining
3. Universal hashing

Issues with linear probing

Linear probing suffers from **primary clustering**. This is the situation of long runs of occupied sequences.

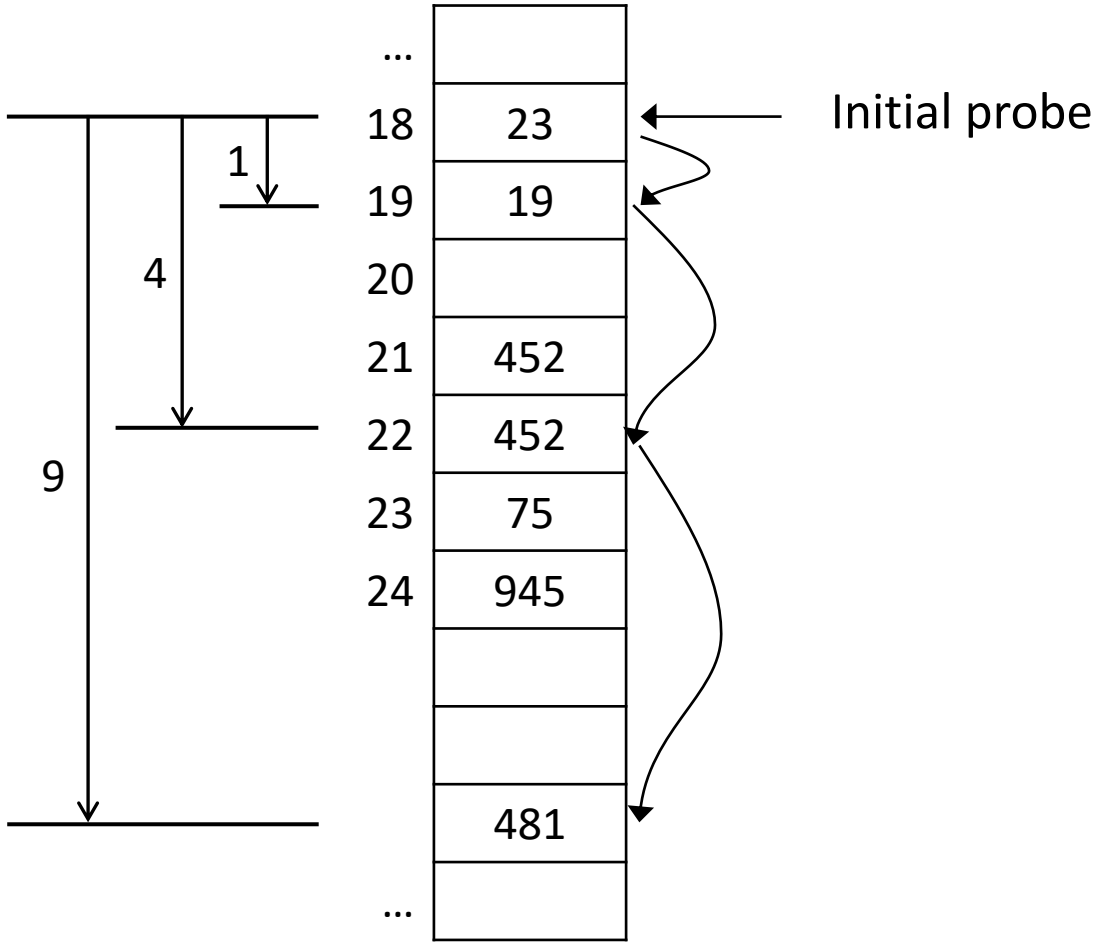


Issues with quadratic probing

Quadratic probing suffers from a milder form of clustering, called **secondary clustering**. Rather than probing sequential positions, it attempts to probe successively further away locations from the previous collision.

But, if two keys contain the same hash address, they will follow the same path (see example at end of L09). Secondary clustering has a lower performance cost than primary clustering, but still not ideal.

Secondary clustering



Double hashing

The drawback of linear and quadratic probing is that collision resolution strategies follow the same path from a collision point regardless of key value.

A better solution is **double hashing:**

$$h(k,i) = \big(h_1(k) + i \cdot h_2(k)\big)\%m$$

- i = the number of collisions so far
- h_1 = division hashing
- m = table size
- h_2 = *a secondary hash function*

Double hashing

$$h(k,i) = \big(h_1(k) + i \cdot h_2(k)\big)\%m$$

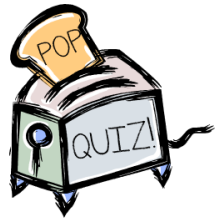
Double hashing offers one of the best methods available for open addressing, as the permutations have many of the characteristics of randomly chosen permutations.

Notice that the initial probe goes to position T[h_1(k)], as i = 0.

h_2 is often something like $h_2(k) = 1 + k\%(m - 1)$.

Pop quiz 1

$$h(k,i) = \big(h_1(k) + i \cdot h_2(k)\big)\%m$$



Why is it helpful to have *m* as a prime number?

Another question

$$h(k,i) = \left(h_1(k) + i \cdot h_2(k)\right)\%m$$

$h_2(k)$ is often of the form: $1 + k\%(m - 1)$.

Why don't we just use $h_2(k) = k\%(m - 1)$?

Another question

$$h(k,i) = \left(h_1(k) + i \cdot h_2(k)\right)\%m$$

$h_2(k)$ is often of the form: 1 + k%(m - 1).

Why don't we just use $h_2(k) = k\%(m - 1)$?

Example

- m = 33, k = 7

$(h_1(k) + i*(k \% m-1)) \% m$	**$(h_1(k) + i*(1 + k \% m-1)) \% m$**
7 14 21 28 2 9 16 23 30 4 11 18 25 32 …	7 15 23 31 6 14 22 30 5 13 21 29 4 12 …

i^{th} slot is a multiple of k

i^{th} slot **not** a multiple of k

Quadratic vs double hashing

Insert (12, 13, 43, 52, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

$H(k, i) = (k\%m + i^2) \% m$

Total collisions = 0

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

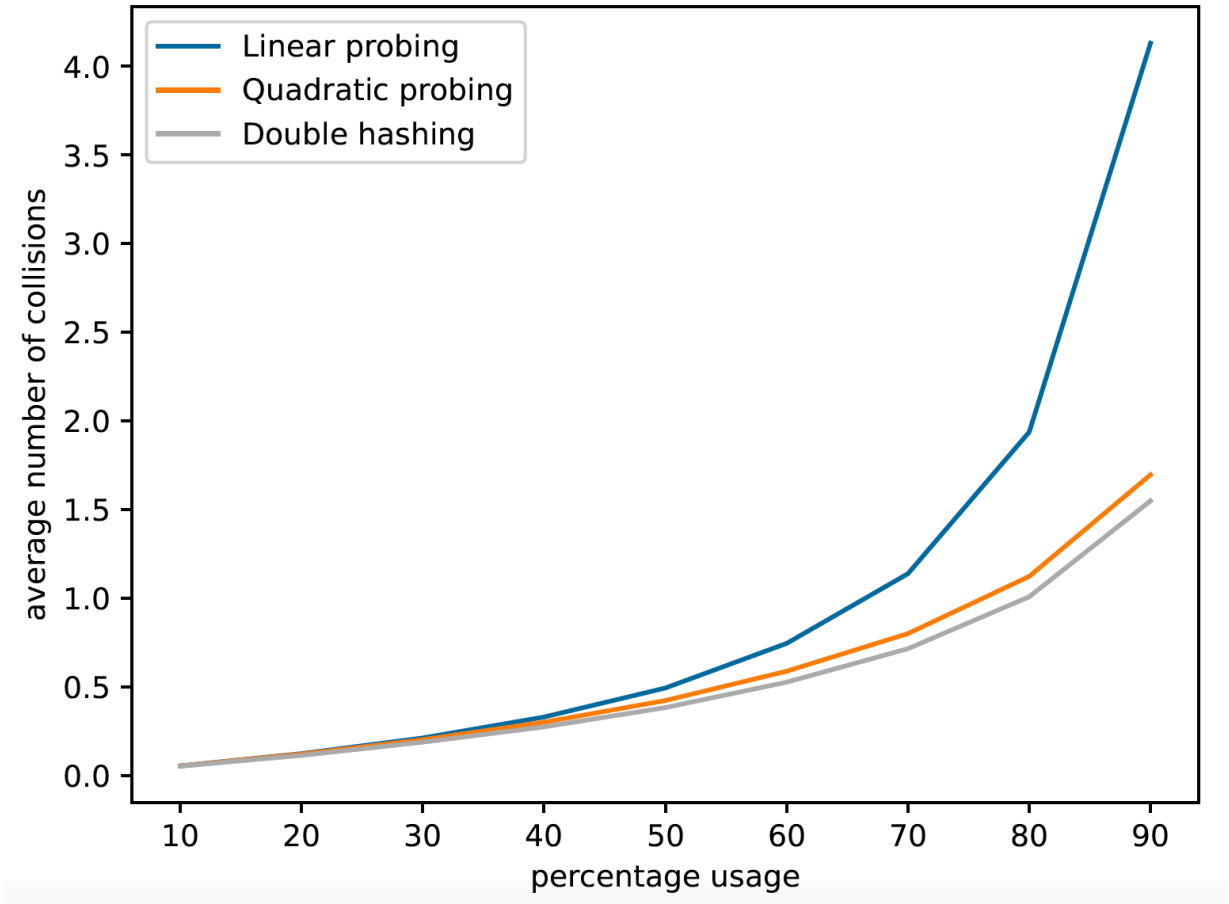
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

double

$H(k, i) = (k\%m + i(1+k\%(m-1)) \% m$

Total collisions = 0

Linear vs quadratic vs double



*Note, in our example m=10 for simpler calculations, but performance is better when m is prime.

Today’s outline

1. Double hashing
2. Chaining
3. Universal hashing

Full tables

What happens as a hash table gets full?

As clusters get large, gaps become fewer and the number of collisions for insertion and for search becomes larger, breaking down the O(1) property.

What if you want to get rid of an item? Beware of deleting a key on a search path that has possible collisions. We have to use *lazy* deletion.

Lazy deletion

Deletion from an open-address hash table is difficult. When we delete a key from slot *i*, we cannot simply mark that slot as empty by storing NIL in it.

If we did, we might be unable to retrieve any key *k* during whose insertion we had probed slot *i* and found it occupied.

With lazy deletion, we mark the slot by storing the special value DELETED instead of NIL.

Full tables

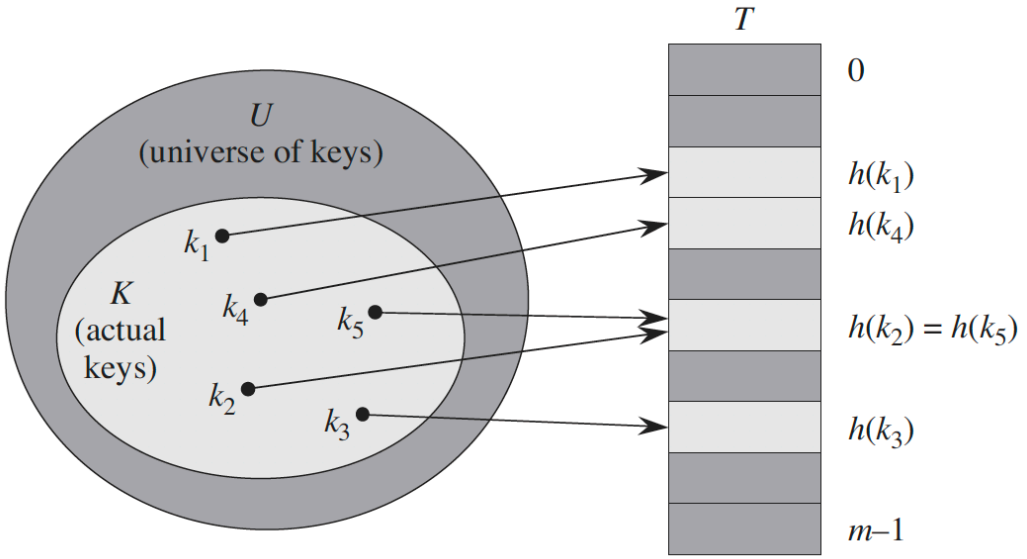
Suggestion: hash tables implemented as arrays are good if you know roughly the size of your data set before loading and it stays that size. Or, if you are willing to accept a single costly operation, you can re-hash everything into a different table size.

Alternative: Allow the hash table to maintain “chains” at each location. This is called “hashing with chaining”; as opposed to what we have already seen, which is called “open addressing”.

Reminder: Collisions

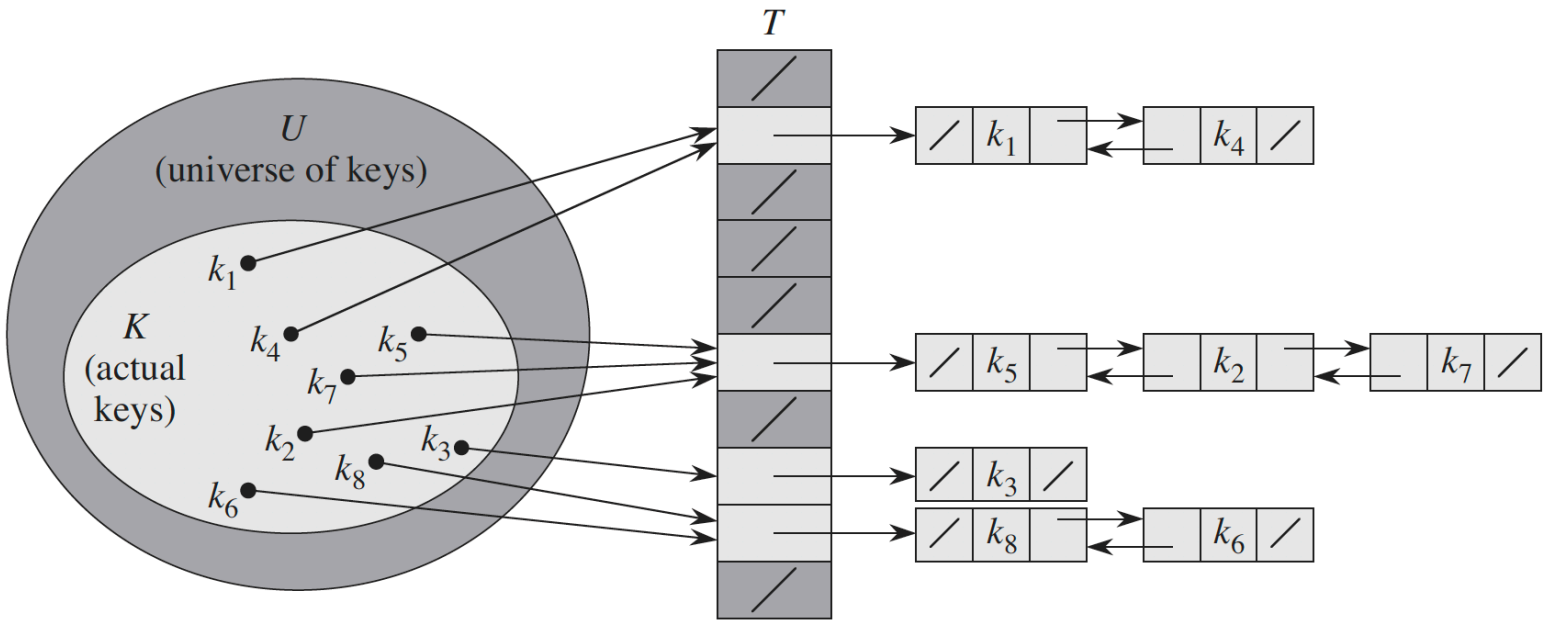
Two keys may hash to the same slot. We call this situation a **collision**.

We've looked at open addressing, lets now look at *chaining*.

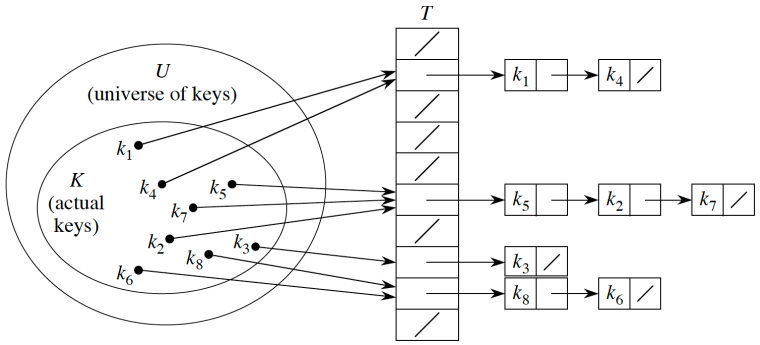


Chaining

In chaining, we place all the elements that hash to the same slot into the same linked list.



Chaining

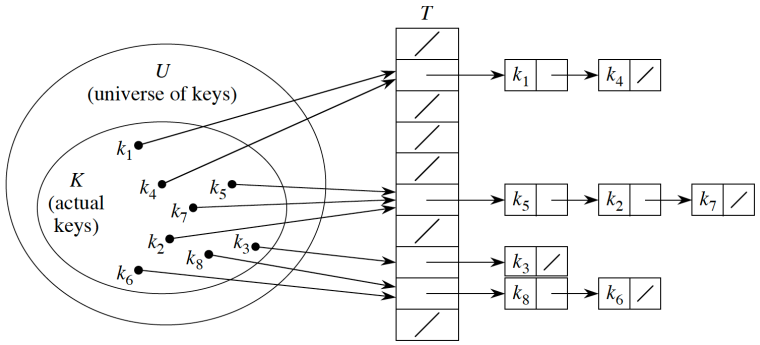


Inserting at the head of a linked list is O(1). Searching a linked list is O(n), which is fine for small *n*, but not as good if *n* is large.

If table has *m* slots and *m* keys, and our hash function spreads the keys fairly evenly over the table, then we can expect our chains to be short.

In fact, it is not unusual for the table size to be chosen to be m = n/3, where *n* is the size of the collection of data items, provided one is sure that your hash function will spread the keys evenly.

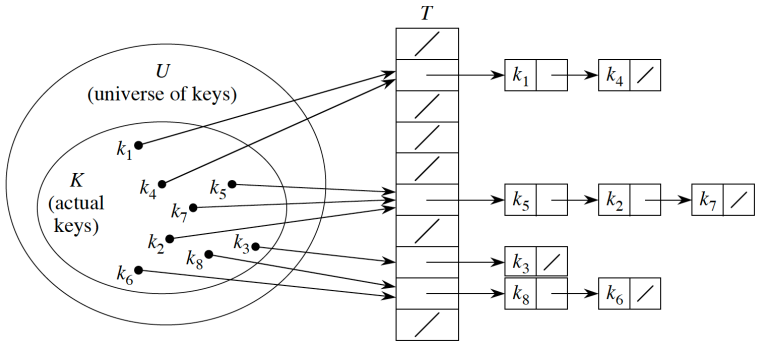
Universal hashing



This scheme is vulnerable to any adversary who is able to select data that creates long lists, making search more time-consuming.

If you use chaining, an adversary who knows your hash function could devise a set of keys that all hash to the same position, creating a list for the whole collection of data items (search and insert become O(*n*)).

Universal hashing



This scheme is vulnerable to any adversary who is able to select data that creates long lists, making search more time-consuming.

If you use chaining, an adversary who knows your hash function could devise a set of keys that all hash to the same position, creating a list for the whole collection of data items (search and insert become O(*n*)).

Solution: choose the hash function randomly before creating the hash table, that is independent of keys to be stored. This is **universal hashing**.

Assumes hash table will not be persistent. Use different hash function on each execution.

Today’s outline

1. Double hashing
2. Chaining
3. Universal hashing

Universal hashing

A *universal* set of hash functions H is a set of hash functions such that if you pick keys k and j at random, and choose a hash function h randomly from H, then the chance of $h(k) = h(j)$ is no more than $1/m$ (where m is the size of the hash table).

So how does one get a universal set of hash functions?

A universal set of hash functions

Choose a prime number p big enough that every possible key *k* is < *p*, and choose your table size *m* < *p*.

Now make $h_{a,b}(k) = \big((ak + b) mod\ p\big) mod\ m$

The parameters *a* and *b* may take on integer values up to *p* - 1, but you must choose *a* > 0.

a and *b* are chosen randomly at program start up.

Example: Universal hashing

$$h_{a,b}(k) = \big((ak + b) mod\ p\big) mod\ m$$

Let p = 17 (prime), m = 6 (table size), a and b chosen randomly:

$$\begin{aligned} h_{3,4}(8) &= \big((3 \cdot 8 + 4)\%17\big)\%6 \\ &= (28\%17)\%6 \\ &= 11\%6 \\ &= 5 \end{aligned}$$

Choosing a and b at random as the program starts denies an adversary the ability to choose input that generates the worst case.

Class Challenge 1

$$h_{a,b}(k) = \big((ak + b) mod\ p\big) mod\ m$$

Try $h_{4,5}(8)$ and $h_{4,5}(8)$, with p = 18, and m = 7.

Suggested reading

Double hashing is discussed in section 11.4.

Chaining is discussed in section 11.2.

Universal hashing is discussed in section 11.3.3.

Solutions

Quadratic vs double hashing

Insert (12, 13, 43, 52, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

$H(k, i) = (k\%m + i^2) \% m$

Total collisions = 0

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

double

$H(k, i) = (k\%m + i(1+k\%(m-1)) \% m$

Total collisions = 0

Quadratic vs double hashing

Insert (12, 13, 43, 52, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(12, 0) = (k%m + i^2) % m = 2

Total collisions = 0

0	
1	
2	12
3	
4	
5	
6	
7	
8	
9	

0	
1	
2	12
3	
4	
5	
6	
7	
8	
9	

double

H(12, 0) = (k%m + i(1+k%(m-1)) % m = 2

Total collisions = 0

Quadratic vs double hashing

Insert (~~12~~, 13, 43, 52, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(13, 0) = (k%m + i^2) % m = 3

Total collisions = 0

0	
1	
2	12
3	13
4	
5	
6	
7	
8	
9	

0	
1	
2	12
3	13
4	
5	
6	
7	
8	
9	

double

H(13, 0) = (k%m + i(1+k%(m-1)) % m = 3

Total collisions = 0

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, 43, 52, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(43, 0) = (k%m + i^2) % m = 3

Total collisions = 1

0	
1	
2	12
3	13
4	
5	
6	
7	
8	
9	

0	
1	
2	12
3	13
4	
5	
6	
7	
8	
9	

double

H(43, 0) = (k%m + i(1+k%(m-1)) % m = 3

Total collisions = 1

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, 52, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(43, 1) = (k%m + i^2) % m = 4

Total collisions = 1

0	
1	
2	12
3	13
4	43
5	
6	
7	
8	
9	

0	
1	43
2	12
3	13
4	
5	
6	
7	
8	
9	

double

H(43, 1) = (k%m + i(1+k%(m-1)) % m = 1

Total collisions = 1

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, 52, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(52, 0) = (k%m + i^2) % m = 2

Total collisions = 2

0	
1	
2	12
3	13
4	43
5	
6	
7	
8	
9	

0	
1	43
2	12
3	13
4	
5	
6	
7	
8	
9	

double

H(52, 0) = (k%m + i(1+k%(m-1)) % m = 2

Total collisions = 2

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, 52, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(52, 1) = (k%m + i^2) % m = 3

Total collisions = 3

0	
1	
2	12
3	13
4	43
5	
6	
7	
8	
9	

0	52
1	43
2	12
3	13
4	
5	
6	
7	
8	
9	

double

H(52, 1) = (k%m + i(1+k%(m-1)) % m = 0

Total collisions = 2

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, ~~52~~, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(52, 2) = (k%m + i^2) % m = 6

Total collisions = 3

0	
1	
2	12
3	13
4	43
5	
6	52
7	
8	
9	

0	52
1	43
2	12
3	13
4	
5	
6	
7	
8	
9	

double

H(52, 1) = (k%m + i(1+k%(m-1)) % m = 0

Total collisions = 2

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, ~~52~~, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(72, 0) = (k%m + i^2) % m = 2

Total collisions = 4

0	
1	
2	12
3	13
4	43
5	
6	52
7	
8	
9	

0	52
1	43
2	12
3	13
4	
5	
6	
7	
8	
9	

double

H(72, 0) = (k%m + i(1+k%(m-1)) % m = 2

Total collisions = 3

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, ~~52~~, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(72, 1) = (k%m + i^2) % m = 3

Total collisions = 5

0	
1	
2	12
3	13
4	43
5	
6	52
7	
8	
9	

0	52
1	43
2	12
3	13
4	
5	
6	
7	
8	
9	

double

H(72, 1) = (k%m + i(1+k%(m-1)) % m = 3

Total collisions = 4

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, ~~52~~, 72, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(72, 2) = (k%m + i^2) % m = 6

Total collisions = 6

0	
1	
2	12
3	13
4	43
5	
6	52
7	
8	
9	

0	52
1	43
2	12
3	13
4	72
5	
6	
7	
8	
9	

double

H(72, 2) = (k%m + i(1+k%(m-1)) % m = 4

Total collisions = 4

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, ~~52~~, ~~72~~, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

$H(72, 3) = (k\%m + i^2) \% m = 5$

Total collisions = 6

0	
1	
2	12
3	13
4	43
5	72
6	52
7	
8	
9	

0	52
1	43
2	12
3	13
4	72
5	
6	
7	
8	
9	

double

$H(72, 2) = (k\%m + i(1+k\%(m-1)) \% m = 4$

Total collisions = 4

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, ~~52~~, ~~72~~, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

$H(63, 0) = (k\%m + i^2) \% m = 3$

Total collisions = 6

0	
1	
2	12
3	13
4	43
5	72
6	52
7	
8	
9	

0	52
1	43
2	12
3	13
4	72
5	
6	
7	
8	
9	

double

$H(63, 0) = (k\%m + i(1+k\%(m-1)) \% m = 3$

Total collisions = 4

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, ~~52~~, ~~72~~, 63) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(63, 1) = (k%m + i^2) % m = 4

Total collisions = 6

0	
1	
2	12
3	13
4	43
5	72
6	52
7	
8	
9	

0	52
1	43
2	12
3	13
4	72
5	
6	
7	
8	
9	

double

H(63, 1) = (k%m + i(1+k%(m-1)) % m = 4

Total collisions = 4

Quadratic vs double hashing

Insert (~~12~~, ~~13~~, ~~43~~, ~~52~~, ~~72~~, ~~63~~) with h(k) = k%10, using first quadratic probing then double hashing.

quadratic

H(63, 2) = (k%m + i^2) % m = 7

Total collisions = 6

0	
1	
2	12
3	13
4	43
5	72
6	52
7	63
8	
9	

0	52
1	43
2	12
3	13
4	72
5	63
6	
7	
8	
9	

double

H(63, 2) = (k%m + i(1+k%(m-1)) % m = 5

Total collisions = 4

Class Challenge 1

$$h_{a,b}(k) = \big((ak + b) mod\ p\big) mod\ m$$

Try $h_{4,5}(8)$ and $h_{3,18}(8)$, with p = 18, and m = 7.

$h_{4,5}(8)$ = ((4*8 + 5) % 18) % 7

= (37 % 18) % 7

= 1 % 7

= 1

$h_{3,18}(8)$ = ((3*8 + 18) % 18) % 7

= (42 % 18) % 7

= 6%7

= 6

Image attributions