SAMPLE SOLUTION

IMPORTANT FIRST STEPS:

1. **Close your laptops and put them away (if necessary, you may refer to your course notes).**
2. **Form a group of 2-3 students.**
3. **Clearly put your names and IDs on 1 copy of this worksheet.**
4. **Be sure to turn this exercise in at the end of class.**

Hashing

Today's exercises are all about hash tables and the notion of a mapping between keys and elements. Recall that a hash table is just a particular type of mapping where the key is passed through a *hash algorithm* that results in an index value into an array.

Let's get hashing!

Suppose we have the following hash function: $h_1(x) = \lfloor x/10 \rfloor$

..and the following compression mapping: $h_2(x) = x \% 10$

(Note that we often simplify this to $h(x) = \lfloor x/10 \rfloor \% 10$)

Given a hash table (i.e. array) with 10 locations, hash the following information into the table.

When you get stuck (i.e. have a problem adding an element), flip to the next page.

Student number	Data (name)
12540	Jasmine
51288	Billy
90100	Sarah
41233	Jane
54991	Jack
45329	Mel
14236	Jimmy

0	90100, Sarah
1	
2	45329, Mel
3	41233, Jane
4	12540, Jasmine
5	
6	
7	
8	51288, Billy
9	54991, Jack

SAMPLE SOLUTION

What happened? **No place for Jimmy**

This is what we call a collision. Apply the pigeonhole principle to describe a collision (remember what a mapping is):

A collision occurs when there are more hash mappings than there are original locations in the hash table to map to… i.e. the mappings are the pigeons and the locations in the table that can be mapped to are the pigeonholes.

Before moving on, take a moment to discuss with your partners your feelings on the quality of this hash function. Is a good hash function? Can the collisions be blamed on the data, or the hash function? Is that always true? Write down your thoughts in note form:

Answers may vary.

This is not a vary good hash function as the chance of collision is high, especially as the table size is increased beyond 10.

SAMPLE SOLUTION

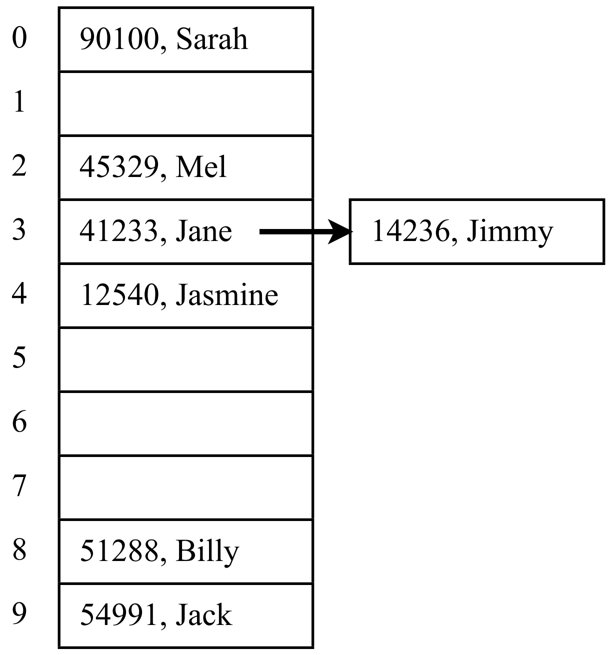
The two general strategies or *collision resolution policies* we're going to consider are **chaining** and **open addressing/probing**.

Chaining:

Chaining is incredibly simple. Keep each location in the array as a linked list. When you have a collision, simply add the new element to the list.

Re-hash the problem above using chaining as your collision resolution strategy (use the space to your right for your chain):

Student number	Data (name)
12540	Jasmine
51288	Billy
90100	Sarah
41233	Jane
54991	Jack
45329	Mel
14236	Jimmy



Answer the following:

What is the *type* of your array? (I.e. if you were to implement this in say C++)

Pointers to student objects

What is the *worst-case search complexity* of your hash table?

It will be the number of elements added.

What can you say about the relationship between the size of the table, N, and the size of S, where S is the set of elements to be hashed?

There can be more elements added than there are spaces in the table. (|S| > N)

SAMPLE SOLUTION

Open Addressing (or Probing):

Open addressing is a little more complex. Instead we define a *sequence* of table entries. Whenever we collide, we follow this sequence until we find a free spot, or we exhaust the sequence. For example, suppose you were trying to find an empty room in which you could nap in a hallway of closed doors. One strategy would be to start at the beginning of the hallway and knock on the first door-- if someone is inside (collision!), move on to the next door. Eventually you will either run out of rooms, or you will find a space for your nap! (No guarantees about pillows, though.)

Linear Probing: In the room-searching algorithm above, the sequence in the case of a collision is an example of a *linear probe.* That is, if you find a collision, keep searching in a linear fashion (i.e. check n, then n+1, etc) until you find a free spot.

Re-hash the problem above using linear probing as your collision resolution strategy:

h(k) = k % 10

Student number	Data (name)
4	Jasmine
44	Billy
444	Sarah
6	Jane
5	Jack
8	Mel
88	Jimmy

0	88, Jimmy
1	
2	
3	
4	4, Jasmine
5	44, Billy
6	444, Sarah
7	6, Jane
8	5, Jack
9	8, Mel

Answer the following:

What is the *worst-case search complexity* of your hash table?

N

What can you say about the relationship between the size of the table, N, and ISI where S is the set of elements to be hashed?

S <= N

(***Notice how each element in the sequence first applies the hash function and THEN adds i)

SAMPLE SOLUTION

One of the risks with linear probing relates to why we want to have our information as evenly dispersed as possible. If we *don't* have that dispersion then we get collisions, and so any collision-resolution principle (CRP) should help us restore a more even data-distribution. Think for a moment, does linear probing offer that?

No… each probe sequence is at risk for colliding with other elements as they are hashed in.

What happens if we get a lot of collisions in an area?

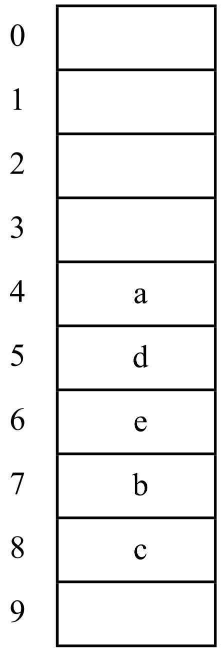
Interfering probe sequences and clustering.

Consider the following:

If I hash "a" to location 4 in an array, and subsequently try to hash "b" to that same location (i.e. 4), a collision will result. The sequence as defined in linear probing tells me to go to the immediately following location (i.e. 4+1 = 5). I collide again (this time with "d"), so I'll try for location 6, collide with "e", and finally end up in spot 7 (notice how *each*) spot in the sequence is tried-- **every time we hash a new element, it is a brand new application of the CRP.**

If I later try to hash "c" to spot 6, I'm going to get another collision, which will result in trying spot 7 (which will collide based on above) and finally 8. Notice how the collision sequence paths overlap one another. If I try to hash something new to 4 again, it will end up in spot 9 by the linear probe sequence (even though we've only had 4 elements hashed to index 4). In other words, the sequences are *clustering* together.

We can avoid this through a more clever choice of probe sequence.



SAMPLE SOLUTION

R'th Probing: With the clustering problem in mind, let's try a couple other options:

Re-hash the problem above using the following probe sequence (remember try one, if it collides, try the next in the sequence, etc; once you place an element, then the next time you have an element that collides, start again at the beginning of the sequence)

$h(k), h(k) + R, h(k) + 2(R), \dots, h(k) + (N-1)(R)$ (all mod N)

Try a few different values of R and compare what happens (I recommend 2, 4, 3).

h(k) = k % 10

Student number	Data (name)
4	Jasmine
14	Billy
114	Sarah
1114	Jane
11114	Jack
8	Mel
118	Jimmy

0	1114, Jane
1	
2	11114, Jack
3	
4	4, Jasmine
5	
6	14, Billy
7	
8	114, Sarah
9	

R= ___2___

0	11114, Jack
1	
2	114, Sarah
3	
4	4, Jasmine
5	
6	1114, Jane
7	
8	14, Billy
9	

R= ___4___

0	114, Sarah
1	118, Jimmy
2	
3	1114, Jane
4	4, Jasmine
5	
6	11114, Jack
7	14, Billy
8	8, Mel
9	

R= ___3___

Answer the following:

What is the *worst-case search complexity* of your hash table? How, if at all, is this affected by R?

The worst-case is always O(N). However, if everything maps to the same location, the actual time it takes to search can be affected by your choice of R. In the case of R=2, or R=4 above the search time is empirically only N/2 (but the collision likelihood is higher).

Did we resolve the clustering problem? (Hint: the probe sequence still has a linear rate of expansion).

No. The clustering effect still occurs-- the probe sequences still collide (though it is not as bad as when there are no spaces between the elements as in linear probing.

Can you offer a suggestion for how to choose a good R value? (For the answer to this question, look to your lecture notes.) **A number relatively prime to N.**

SAMPLE SOLUTION

Quadratic Probing: Re-hash the problem above using the following probe sequence

$h(k), h(k) + 1^2, h(k) + 2^2, \ldots , h(k) + (N-1)^2$ (all mod N)

h(k) = k % 10

Student number	Data (name)
4	Jasmine
14	Billy
114	Sarah
1114	Jane
11114	Jack
8	Mel
118	Jimmy

0	11114, Jack
1	
2	118, Jimmy
3	1114, Jane
4	4, Jasmine
5	14, Billy
6	
7	
8	114, Sarah
9	8, Mel

Answer the following:

What is the *worst-case search complexity* of your hash table?

N

What do you note about the distribution of colliding elements now? What is the risk for (consecutive) clustering?

Less chance of clustering.

We've introduced a new problem, however… what is it? (Hint: write out the probe sequence for 1-(N-1) for h(k) = 4. What do you notice?)

Will only hash to half the locations in the table if an entire probe sequence is used.

Think about Random Probing (why is it a poor choice, in general?).
Have to record the seed value, and thus might as well record the actual location of the element (advantage of hashing is lost).

SAMPLE SOLUTION

What happens when you need to delete an element that falls in the middle of the probe sequence? Recreate your last hash table from the previous page.

Delete “Jane” (1114).

What happens when you search for “Jack”? (Remember that you reapply your probe sequence in the same way to do searching.)

0	11114, Jack
1	
2	118, Jimmy
3	
4	4, Jasmine
5	14, Billy
6	
7	
8	114, Sarah
9	8, Mel

In your own words now, describe the role of a tombstone.

It marks the location of previous elements that might have caused a collision, thus forcing subsequent elements to be placed *later* in the probe sequence. When searching if the tombstone is not there, the search will terminate prematurely.