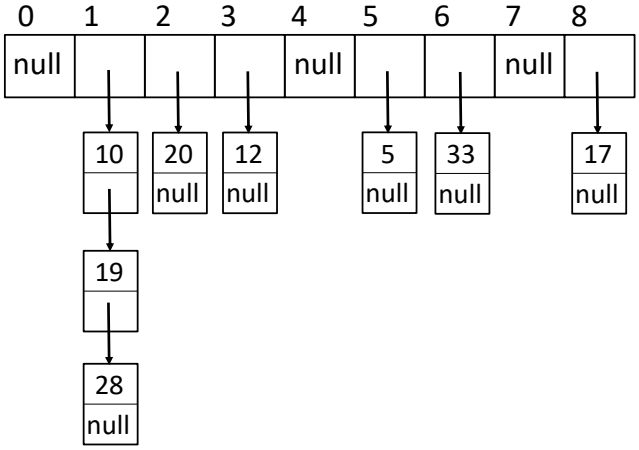
Module IN2002—Data Structures and Algorithms
Answers to Exercise Sheet 6

1. Demonstrate the insertion of the keys 5, 28, 19, 20, 33, 12, 17, 10 into a hash table of 9 slots. The hash function is h(k) = k mod 9, and collisions are resolved by chaining. How much difference would it make if the keys were presented in a different order?

- The hashing function is h(k) = k mod 9
- 5 is placed in slot 5 mod 9 = 5
- 28 is placed in slot 28 mod 9 = 1
- 19 is to be placed in slot 19 mod 9 = 1, so it is added to the singly linked list
- ...and so on

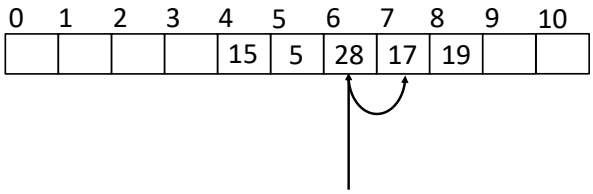


- Adding the elements in a different order would change the ordering of the elements in each singly linked list, but this would not affect the average performance of search or insertion.

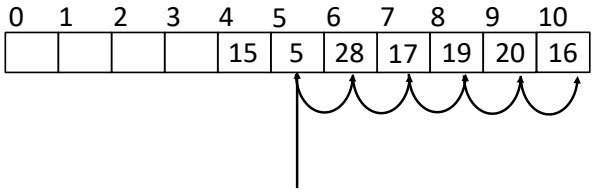
2. Demonstrate the insertion of the keys 5, 28, 19, 15, 17, 20, 16 and 30 into a hash table of 11 slots, with hash function h(k) = k mod 11, and collisions resolved by open addressing, using a) linear probing, and b) quadratic probing. What are the average and worst cases for search?

a) Linear probing

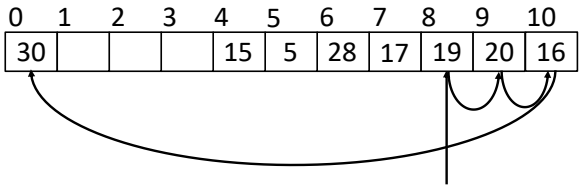
- 5 mod 11 = 5 is empty, so 5 if placed in the slot
- 28 mod 11 = 6 is empty, so 28 is placed in the slot,
- ... and the same happens with 19 and 15.
- However, 17 mod 11 = 6, which is occupied. Consecutive slots are tried until an empty one is found:



- 20 mod 11 = 9, which is empty.
- 16 mod 11 = 5, but it is occupied, so the consecutive slots are tried



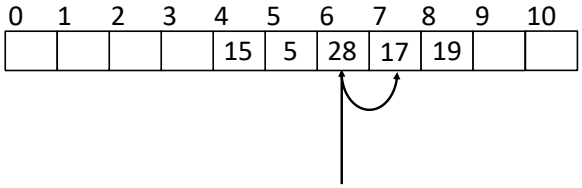
- 30 mod 11 is 8, but it is occupied, so the consecutive slots are tried until an empty one is found



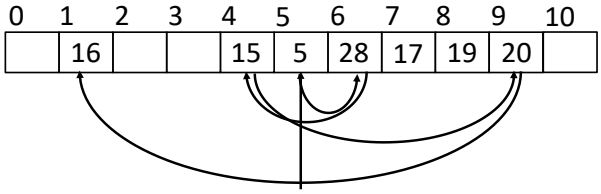
- The average case for a successful search here is equivalent to the average of the times it takes to insert the keys in the hash table (17/8 = 2.125). The worst successful search case would be equivalent to the worst insertion case (16, which required checking 6 slots).
- The worst unsuccessful search would involve searching the first element in the largest cluster, i.e., slot 4. For example, to search for 26 (26 mod 11 = 4), 9 slots would need to be checked before realising that it is not there. The average case for an unsuccessful search here is 4.5 in the cluster ([2+…+9]/8) and 1 on each of the three empty elements, giving ~4.27.. on average.

b) Quadratic probing

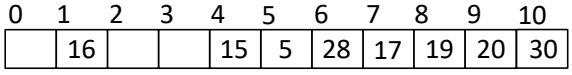
- Again, 5, 28, 19 and 15 find empty slots straight away.
- 17 mod 11 = 6, which is occupied, so then (6 + 1) mod 11 = 7 is tried and found to be empty.



- 20 mod 11 = 9, which is empty. However, 16 mod 11 is 5, which is occupied, so then the quadratic probing tries (5 + 1^2) mod 11 = 6, (5 – 1^2) mod 11 = 4, (5 + 2^2) mod 11 = 9, and (5 – 2^2) mod 11 = 1, which is empty.



- Inserting 30 requires looking at 30 mod 11 = 8, (8 + 1^2) mod 11 = 9, (8 – 1^2) mod 11 = 7, (8 + 2^2) mod 11 = 1, (8 – 2^2) mod 11 = 4, (8 + 3^2) mod 11 = 6, (8 – 3^2) mod 11 = -1. Since this is negative we add the table length to the result to bring us back into the bounds of the table (-1+11= 10). Slot 10 is empty, so 30 is stored there.



- Once again, the average case for a successful search here is equivalent to the average of the times it took to insert the keys in the hash table. The worst successful search case would be equivalent to the worst insertion case (30, which required checking 7 slots). The worst and average unsuccessful search is not as clearly cut as with linear probing. We can calculate it for each case, but there is no general dependency on the cluster length. For example, to search for 27 (27 mod 11 = 5), 7 slots would need to be checked before realising that it is not in the hash table.

3. Write `delete` for a hash table with chaining.

- For deleting a key, we first need to find the right slot, then search through the chain and then delete the node from the chain.

```
void delete(int key) {
     int index = hash(key); // find the slot address
     Node p = table[index]; // head of the chain
     if (p != null) { // if chain is not empty
          if (p.info == key) // if head contains key
               table[index] = p.next; // remove head
          else { // otherwise
               while (p.next != null && //until chain
                         p.next.info != key) { // end or key
                    p = p.next; // found, go forward
               if (p.next != null) // !end => key found
                    p.next = p.next.next; // delete
          }
     }
}
```

4. Suppose we have a hash table of size m, with hash function h(k) = k mod m, and with collisions resolved by chaining. Suppose further that all the keys we will insert divide evenly by 4 (for example, they are pointers on a certain architecture). What happens if m is 20? 19? 18? In general, which values of m give better performance?

- If m is 20, only slots 0, 4, 8, 12 and 16 will be used, and will have longer chains. If m is 18, half of the slots are used (the even-numbered ones). If m is 19, all the slots are used.
- For this distribution of keys, we should choose an odd value for m, so that the keys will be distributed across all the slots.