Algorithm assignment 5
Nisharg Gosai

Problems

1. (15pts) Implement in pseudocode a queue data structure using a singly linked list. The operations ENQUEUE and DEQUEUE should take O(1) time. Do you need to add any attributes to the list?

```
EMPTY- QUEUE:
1       if L.head == NULL
2               return True
3        return False

ENQUEUE(L,x):
1       if EMPTY-QUEUE:
2               L.head = x
3       else:
4               L.tail.next = x
5               L.tail = x
6               x.next = NULL

DEQUEUE(L):
1       if EMPTY-QUEUE:
2               error “underflow”
3       else:
4               x = L.head
5               if L.head== L.tail:
6                       L.tail = NULL
7               L.head =  L.head.next
8        return x
```

We require two attributes in this case:
Data: The value of the node is stored in this attribute.
Next: This attribute points to the list's subsequent node.

2. (15pts) Suggest how to implement a direct-address table in which the keys of stored elements do not need to be distinct and the elements can have satellite data. All three dictionary operations should run in O(1) time. (Don’t forget that DELETE takes as an argument a pointer to an object to be deleted, not the key.)

In this scenario, a doubly linked list is utilized. Given that keys may not be unique and can have associated satellite data, they are each linked to a distinct doubly linked list. The three main operations are as follows:

- INSERT: Adding an element is done by attaching it to the end or beginning of the doubly linked list associated with its key. This process is efficient, with a time complexity of O(1), due to the nature of the doubly linked list.
- DELETE: To delete an element, you use a pointer to that specific object. With this pointer, the element can be quickly removed from its list. This is because the list's structure allows for easy adjustment of the neighboring elements' pointers to exclude the deleted item. This operation also has a time complexity of O(1).
- SEARCH: The search function simply retrieves the first node in the linked list that matches the specified key. This is a swift operation, achievable in O(1) time.

3. (20pts) You use a hash function h to hash n distinct keys into an array T of length m. Assuming independent uniform hashing, what is the expected number of collisions?

Since there are "m" available hash slots and each key is equally likely to be hashed to any slot in the hash table, regardless of where the other elements are being hashed to,

Probability that two distinct keys, "a" and "b" collide -> h(a) = h(b) = 1/m.

n/2 ways of choosing 2 keys out of n and each pair has 1/m probability

The expected number of collisions, E[C] = n/2 * 1/m

$$= \frac{n!}{2!(n-2)!} * \frac{1}{m}$$

$$= \frac{n(n-1)(n-2)!}{2!(n-2)!} * \frac{1}{m}$$

$$= \frac{n(n-1)}{2} * \frac{1}{m}$$

$$= \frac{n^2-n}{2m}$$

4. (18pts) Consider a hash table with 9 slots, and the hash function h(k) = k mod 9. Demonstrate what happens upon inserting the keys 5, 28, 19, 15, 20, 33, 12, 17, 10 with collisions resolved by chaining.

Key = k	**h(k) = k mod 9**	**Hash table Position, Value**	
5	5	0	null
28	1	1	10->19->28
19	1	2	20
15	6	3	12
20	2	4	null
33	6	5	5
12	3	6	33->15
17	8	7	null
10	1	8	17

5. (14pts) Suppose you have the following hash table, implemented using open addressing and linear probing. The hash function we are using is the following: h(k) = k mod 9. In which order could the elements have been added to the hash table? Assume that the hash table has never been resized, and no elements have been deleted yet.

0	1	2	3	4	5	6	7	8
9	18		12	3	14	4	21	

For 9,
9 mod 9 = 0, 9 goes to position 0

For 18,
18 mod 9 = 0, 9 occupies position 0
So 18 goes to position 1

For 12,
12 mod 9 = 3, 12 goes to position 3

For 3,
3 mod 9 = 3, 12 occupies position 3
So 3 goes to position 4

For 14,
14 mod 9 = 5, 14 goes to position 5

For 4,
4 mod 9 = 4, 3 occupies position 4
14 occupies position 5
So 4 goes to position 6

For 21,
21 mod 9 = 3, 12 occupies position 3
3 occupies position 4
14 occupies position 5
4 occupies position 6
So 21 goes to position 7

6. (20pts) Write the pseudocode for a non-recursive implementation of the inorder tree walk.

```
ITERATIVE-ITW(T)
1.      Create empty stack S
2.      x = T.root
3.      y = 0
4.      while ! y:
5.              If x != NULL:
6.                      PUSH(S.x)
7.                      x = x.left
8.              Else:
9.                      If ! S.EMPTY:
10.                             x=POP(S)
11.                             print x
12.                             x = x.right
13.                     Else:
14.                             y = 1
```

7. (10pts) For the set {1, 4, 5, 10, 16, 17, 21} of keys, draw binary search trees of height 2, 3, 4, 5, and 6.