Hashing and Hash Tables

1. Suppose our hash function simply returns 17 every time it is called. Is such a hash function legal? If so, describe the effect of using it. If not, explain why.

If we're separate chaining and 17 is in the alotted array space then it's fine, though it is not very useful as it amounts to a linked list with the extra space alotted to the array of head pointers of which only one is being used. Different hash function should therefore be used or made. For linear probing this will only work as many times as alotted spaces there are in the array.

2. Describe an algorithm to delete an element in a hash table using linear probing.

Get its index from the hash function and check if the element there is the element to delete. If not keep moving up one and checking until either the element matches or a null is encountered. When deleting place a tombstone instead of null for future searching or deleting of elements that collided, or place a null and rehash the consecutive non-null entries after.

3. Describe the algorithm used to delete an element in a hash table using separate chaining.

Get its array index from the hash function and from there you traverse the linked list and remove as you would in a normal linked list or stop when reaching the end of the linked list.

4. Insert the keys `E A S Y Q U T I O N` in that order into an initially empty table of $M = 5$ lists, using separate chaining. Use the hash function $11k\%M$ to transform the kth letter of the alphabet into a table index. You do not need to resize the hash table as a part of your solution. Show only the final hash table after all insertions have been completed.

h(E)= 11*5%5 = 0, h(A) = 11*1%5 = 1, h(S) = 11*19%5 = 4, h(Y) = 11*25%5 = 0, h(Q) = 11*17%5 = 2, h(U) = 11*21%5 = 1, h(T) = 11*20%5 = 0, h(I) = 11*9%5 = 4, h(O) = 11*15%5 = 0, h(N) = 11*14%5 = 4

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

5. Insert the keys `E A S Y Q U T I O N` in that order into an initially empty table of size $M = 16$ using linear probing. Use the hash function $11k\%M$ to transform the kth letter of the alphabet into a table index. You do not need to resize the hash table as a part of your solution. Show only the final hash table after all insertions have been completed.

6 – 2	5 – 2	5 – 10
2 – 0	2 – 3	4 – 1
3 – 6	8 – 1	1 – 11
11 – 8	7 – 11	0 – 6
8 – 4	7 – 8	4 – 8
10 – 3	3 – 10	

Figure 1: A set of vertices connected within a graph

h(E)= 11*5%16 = 7, h(A) = 11*1%16 = 11, h(S) = 11*19%16 = 1, h(Y) = 11*25%16= 3, h(Q) = 11*17%16 = 11, h(U) = 11*21%16 = 7, h(T) = 11*20%16 = 12, h(I) = 11*9%16 = 3, h(O) = 11*15%16 = 5, h(N) = 11*14%16 = 10

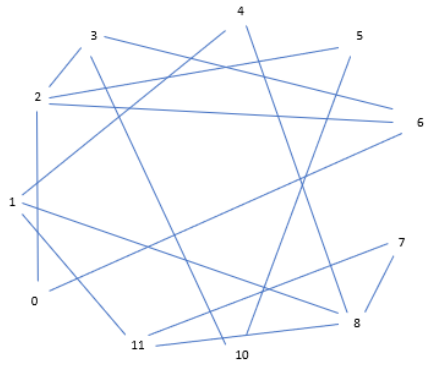
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	S		Y	I	O		E	U		N	A	Q	T		

Undirected Graphs

6. When representing a graph in memory, when is it better to use an adjacency list? When is it better to use an adjacency matrix?

List when the graph is sparse (few edges relative to number of vertices) and matrix when the graph is dense (high ratio, relatively speaking).

7. Given the list of vertices and edges in Figure 1, draw the resulting graph.



Two connected components: {1, 4, 7, 8, 11} and {0, 2, 3, 5, 6, 10}

8. Given the list of vertices and edges in Figure 1, write out the resulting adjacency list once the graph is loaded into memory.

```
0: 6, 2
1: 11, 4, 8
2: 3, 5, 0, 6
3: 2, 10, 6
4: 1, 8
5: 10, 2
6: 0, 3, 2
7: 8, 11
8: 7, 1, 4, 11
9:
10: 5, 3
11: 1, 7, 8
```

9. Does breadth first search tell us anything about the distance from node v to node w when neither is at the root of the search?

The difference between the distances of v and w from the root gives a lower bound on the distance from v to w.

10. A colleague suggests you to use a stack instead of a queue when running breadth first search. Would your colleague's suggestion still compute shortest paths in a non-edge-weighted undirected graph? Why or why not?

Using a queue the breadth first search finds the shortest path as soon as it finds the node/vertex. In the example in the lecture notes, if we use a stack, then instead of removing 2, we remove 5. This isn't the important part since 5, 1, and 2 could have gone into the stack (or queue) in the reverse of the order in the example. But, three would be removed next. Then, since 4 is adjacent to 3, that gives a path of length 3 instead of 2, which is the length of the first path finding 3 using a queue. You wouldn't be able to find a path a path of length 2 until you get to the bottom of the stack, which would be the element 2. And, this would require further change beyond just switching queue to stack.