Alexandria University
Faculty of Engineering
Computer and Systems Engineering Department
Spring SEMESTER 2022

جامعة الاسكندرية
كلية الهندسة

CSE - X22: Data Structures I
INSTRUCTOR: DR. KHALED NAGI
Due: 16 May 2022

Document Content Description

- This Document Contains ***" Sheet #5 "*** answers submission in CSE - X22 Course.
- Sheet 5 - Graphs and Hashing → (11) Problem
- Solution Model Example:

[Question Number]
    [Question Statement]
["Answer"]
    [Solution]

Prepared By

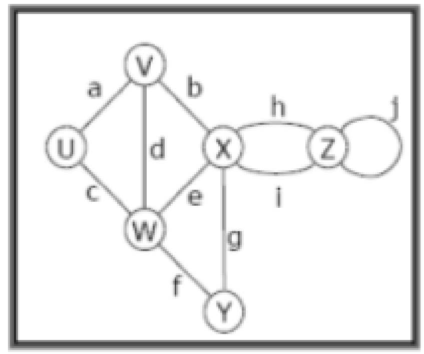
Student Name: *George Samy Wahba Beshay*

Student Academic ID: *20010435*

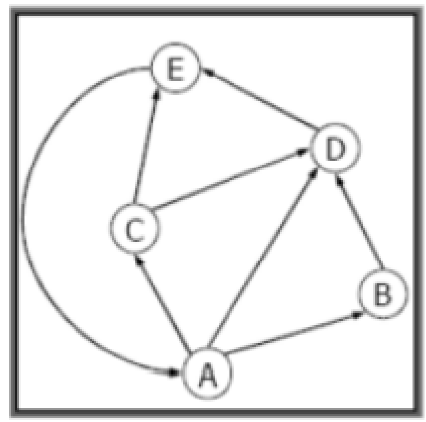
Question 1:

1. For the following graphs, list the following:

1. Vertices
2. Degree of each vertex
3. Parallel edges
4. Self loops
5. Adjacent vertices



1. Vertices
2. In-degree and out-degree of each vertex



Answer:

A. For the first graph:

3. Parallel Edges: (h,i)

4. Self Loops: (j)

5. Adjacent Vertices:

[(U,V) - (U,W) - (V,W) - (V,X) - (W,X) - (W,Y) - (X,Y) - (X,Z)]

Graph A

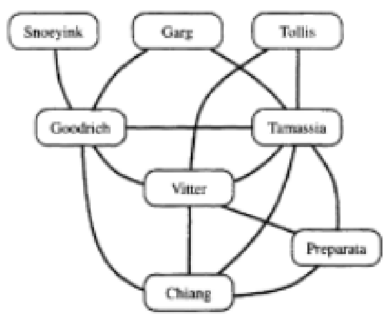
Vertex	Degree
U	2
V	3
W	4
X	5
Y	2
Z	3

B. For the second graph:

Vertex	IN Degree	OUT Degree
A	1	3
B	1	1
C	1	2
D	3	1
E	2	1

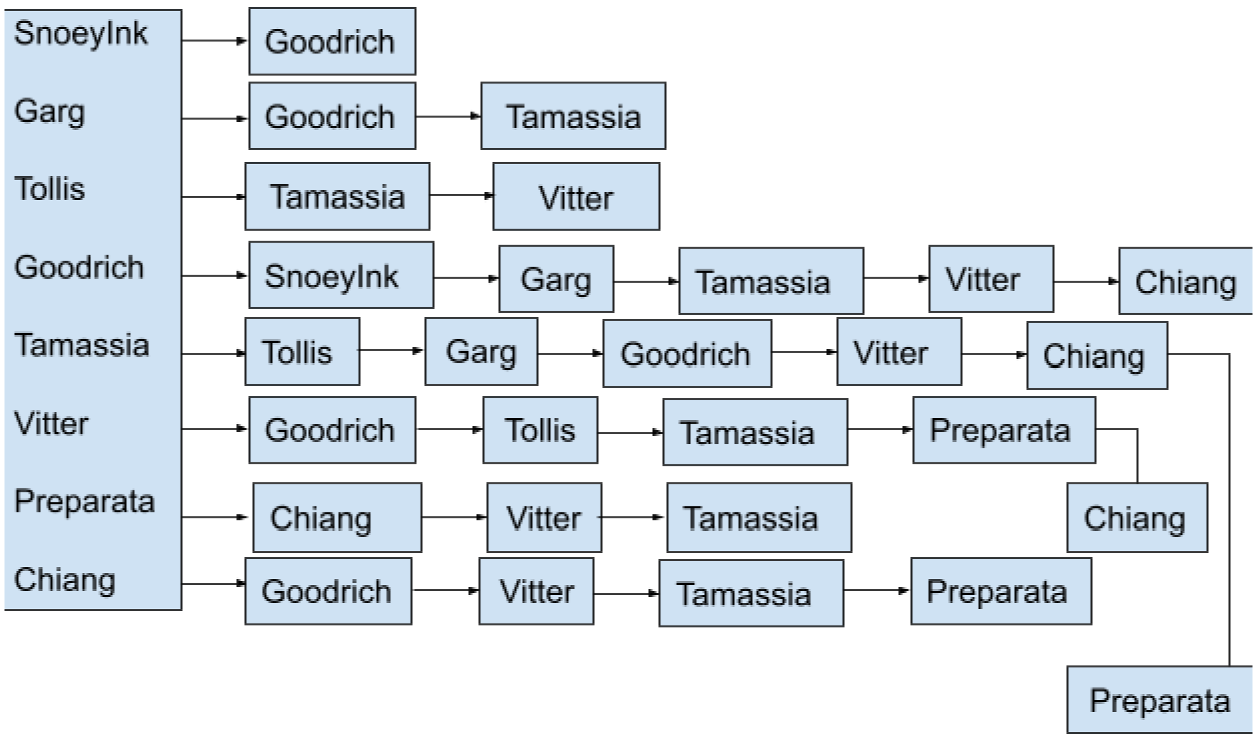
Question 2:

2. Draw the adjacency list and adjacency matrix representation of the following undirected graph



Answer:

1. Adjacency List (Traditional)
   Consists from 1D Array, and each element in this array resembles a vertex, and from each vertex a linked list is attached to.



1. Adjacency Matrix (Traditional)

	Snoeyink	Garg	Tollis	Goodrich	Tamassia	Vitter	Chiang	Preparata
Snoeyink	F	F	F	T	F	F	F	F
Garg	F	F	F	T	T	F	F	F
Tollis	F	F	F	F	T	T	F	F
Goodrich	T	T	F	F	T	T	T	F
Tamassia	F	T	T	T	T	T	T	T
Vitter	F	F	F	T	T	F	T	T
Chiang	F	F	F	T	T	T	F	T
Preparata	F	F	F	F	T	T	T	F

Question 3:

3. Let G be a graph whose vertices are the integers 1 through 8 and let the adjacent vertices of each vertex be given by the table below:

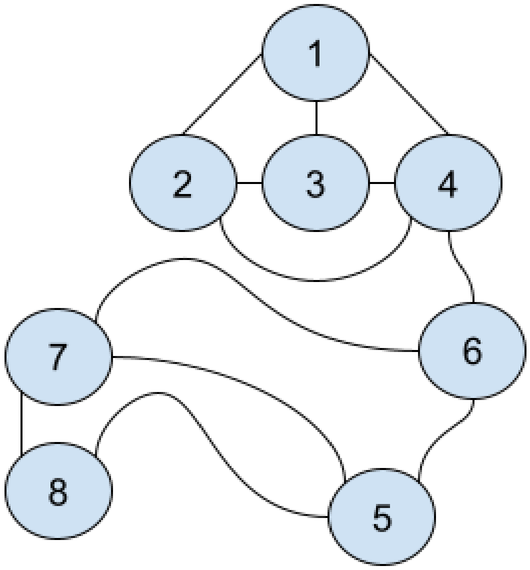
Vertex	Adjacent vertices
1	(2, 3, 4)
2	(1, 3, 4)
3	(1, 2, 4)
4	(1, 2, 3, 6)
5	(6, 7, 8)
6	(4, 5, 7)
7	(5, 6, 8)
8	(5, 7)

Assume that, in a traversal of G, the adjacent vertices of a given vertex are returned in the same order as they are listed in the table.

1. Draw G.
2. Give the sequence of vertices visited using a DFS traversal starting at vertex 1.
3. Give the sequence of vertices visited using a BFS traversal starting at vertex 1.

Answer:

1.



2. DFS: [1 - 2 - 3 - 4 - 6 - 7 - 5 - 8]

3. BFS: [1 - 2 - 3 - 4 - 6 - 5 - 7 - 8]

Question 4:

4. Bob loves foreign languages and wants to plan his course schedule for the following years. He is interested in the following 9 language courses: LA15, LA16, LA22, LA31, LA32, LA126, LA127, LA141 and LA 169.

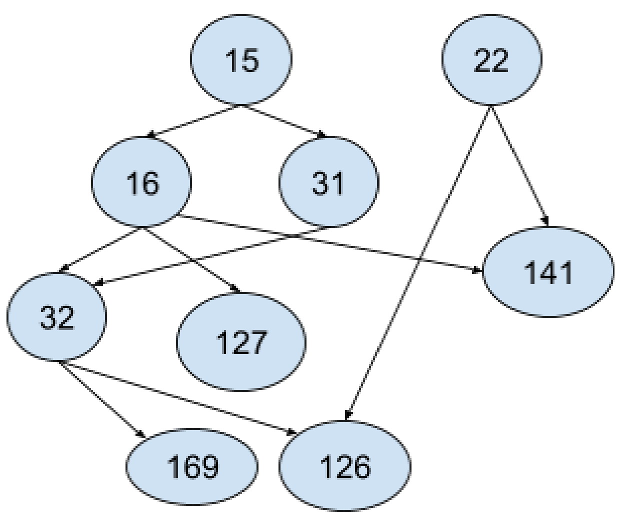
The course prerequisites are:

- LA15 : (none)
- LA16 : LA15
- LA22 : (none)
- LA31 : LA15
- LA32 : LA16, LA31
- LA126 : LA22, LA32
- LA127 : LA16
- LA141 : LA22, LA16
- LA 169 : LA32

Find the sequence of courses that allows Bob to satisfy all the prerequisites.
Hint: Check topological sort

Answer:

After Converted the given data into a **directed graph** as follows:



The sequence can be obtained by using the BFS:

[LA15 - LA22 - LA16 - LA31 - LA141 - LA32 - LA127 - LA169 - LA126]

Question 5:

5. Prove that:

• If G is an undirected graph having n vertices and m edges:

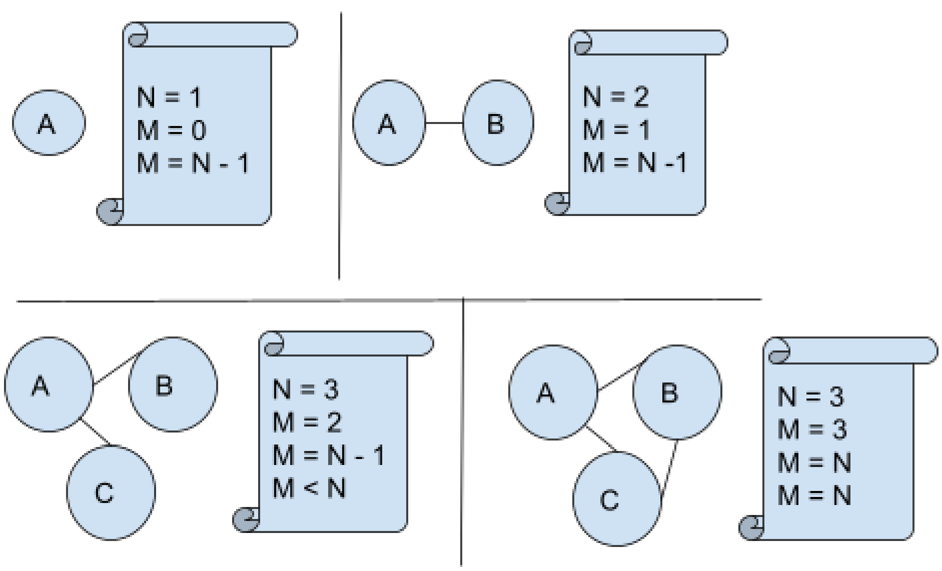
1. If G is connected then $m \geq n - 1$
2. If G is a tree then $m = n - 1$
3. If G is a forest then $m \leq n - 1$
4. If G is a complete graph then $m = n * (n - 1) / 2$

• If G is a directed graph having n vertices, then the maximum number of edges is $n(n - 1)$

Hint: A tree is a connected graph with no cycles. A forest is a graph with each connected component a tree.

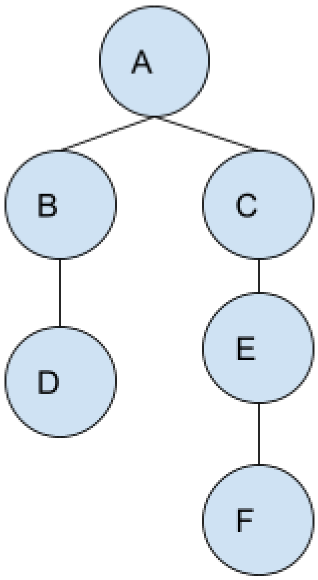
Answer:

1. By examining more than one scenario



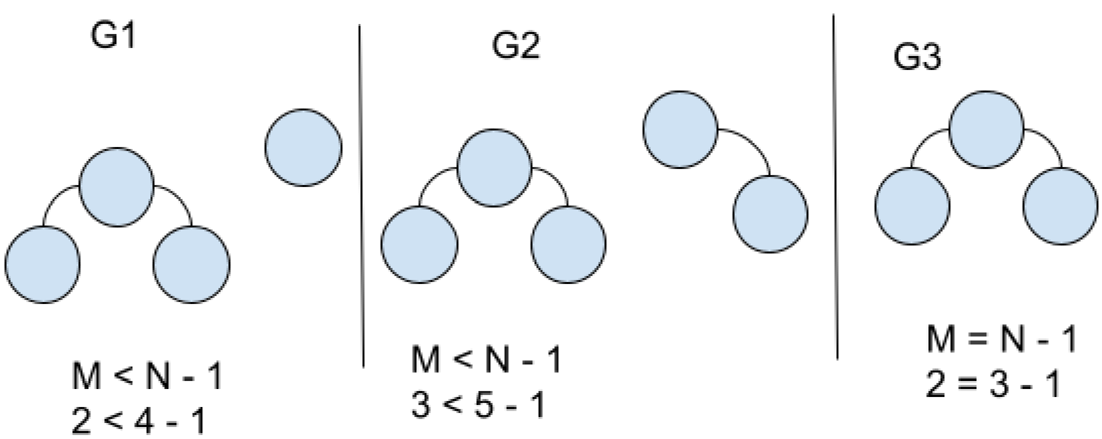
Conclusion: if G is a connected graph then number of edges is less than or equal the number of vertices.

2. By Examining any type of tree, we will notice that the number of edges is equal to the number of vertices - 1, this is because any vertex is ALWAYS related to its superior vertex (parent) by an edge except for the root of the tree. So m = n - 1.



In the shown graph, the tree consists of 6 vertices and 5 edges.

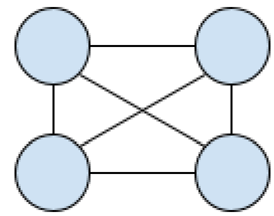
3. If G is a forest, then G is composed of disjoint trees, and based on that in each tree m = n - 1, then if the forest is composed of 2 trees, this formula will be m < n - 1, but if the forest is composed of a solo tree we will find out that m = n - 1



4. Based on the definition of a complete graph, a graph G is complete IFF between each possible pair of vertices that belongs to this graph an edge. So if a graph is composed of 4 vertices, in order to name this graph "Complete" there should be an edge between each possible pair of vertices. So we can notice that the number of edges is equal to:

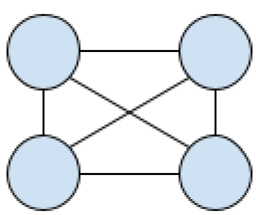
$\sum^{n-1} (i)$, such that n is the number of vertices.

It is clear that this summation formula is equal to (n-1)*(n)/2 .

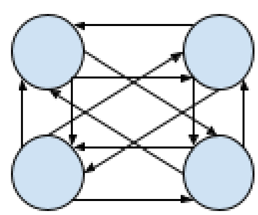


m = 6 = 4 * 3 / 2

If G is a directed graph containing n vertices, so the maximum number of edges can be calculated from getting a complete undirected graph (in which all possible vertices are connected by an edge) and multiplying it by 2 in order to replace each edge by 2 edges (To and from), and the following representation can make this point clear:



Complete undirected graph containing 6 edges



Directed graph containing maximum number (all possible) edges = 12

So the maximum number of edges in a directed graph can be calculated from the following relation:

M = Number of Edges in a complete graph * 2

M = n(n-1)/2*2 = n(n-1).

Question 6:

Write an algorithm to detect if an undirected graph contains cycles.

Answer:

// The idea of finding a cycle in an undirected graph is to DFS traversal the graph, and on each node we will check its adjacent nodes, if any adjacent node was visited (VisitFlag is true) AND (Logical And) not the parent of the current node that we are pointing at, then a cycle is detected.

```
Algorithm DetectCycle(Graph G){
        Input: Graph G (Vertex)
        Output: Boolean value either True or False that detects the presence of a cycle in
        a graph.
        int RemainingVertices = G.NumVertices();                    // Global Variable
        Vertex CurrentV = G;
        Vertex CurrentParent = null;
        If RecursionFunction(CurrentV,CurrentParent){
                return false;
        }
        return true;
}
Algorithm RecursionFunction(Vertex V, Vertex P){                    // The current vertex and
                                                                    // the parent.
        Boolean TotalFlag = True;
        NumberOfVertices --;
        for(i in V.adjacentVertices){
                if(i.VisitFlag == 1 && i != P){
                        Break;
                        return false;
```

```
            }
        }
        for(i in V.adjacentVertices){
            if(i.VisitFlag == 0){
                TotalFlag = TotalFlag && RecursionFunction(i,V);
            }
        }
}
```

Question 7:

Draw the 11-entry hash table that results from using the hash function

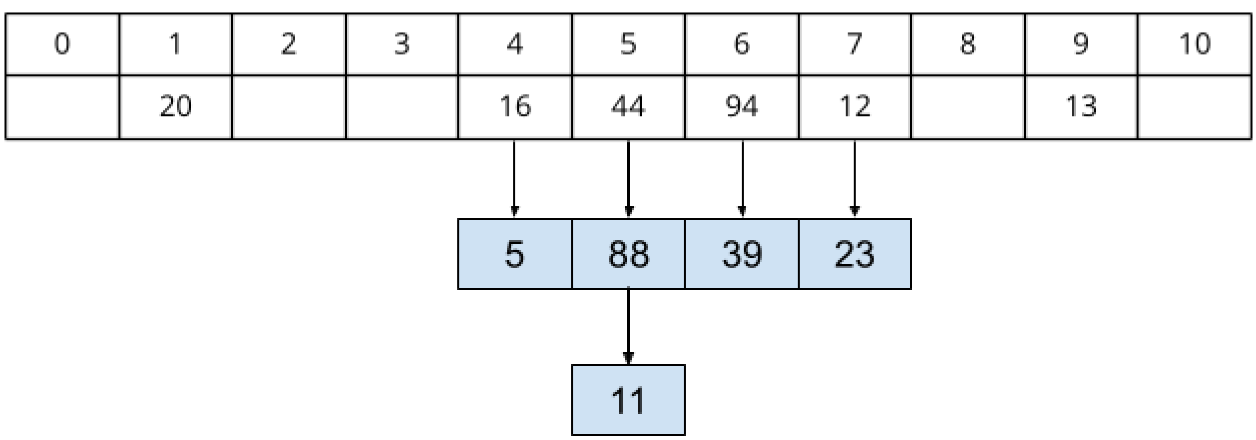
h(i) = (2i + 5) mod 11

to hash the keys:

12, 44, 13, 88, 23, 94, 11, 39, 20, 16 and 5.

assuming collisions are handled by separate chaining

Answer:



Tracking the hash function insertion:

Key 12: h(12) = 7 // Empty → Insert
Key 44: h(44) = 5 // Empty → Insert
Key 13: h(13) = 9 // Empty → Insert
Key 88: h(88) = 5 // Occupied → Insert at linked list starting from index 5
Key 23: h(23) = 7 // Occupied → Insert at linked list starting from index 7
Key 94: h(94) = 6 // Empty → Insert
Key 11: h(11) = 5 // Occupied → Insert at linked list starting from index 5
Key 39: h(39) = 6 // Occupied → Insert at linked list starting from index 6
Key 20: h(20) = 1 // Empty → Insert
Key 16: h(16) = 4 // Empty → Insert
Key 5: h(5) = 4 // Occupied → Insert at linked list starting from index 4

Question 8:

Solve the previous problem again assuming collisions are handled by linear probing.

Answer:

0	1	2	3	4	5	6	7	8	9	10
11	39	20	5	16	44	88	12	23	13	94

Tracking the hash function insertion:

Key 12: h(12) = 7 // Empty → Insert
Key 44: h(44) = 5 // Empty → Insert
Key 13: h(13) = 9 // Empty → Insert
Key 88: h(88) = 5 // Occupied → Insert at (5+1)%11 = 6

Key 23: h(23) = 7 *// Occupied → Insert at (7+1)%11 = 8*

Key 94: h(94) = 6 *// Occupied → Insert at (6+1)%11 → Occupied*

// Insert at (7+1)%11 → Occupied

// Insert at (8+1)%11 → Occupied

// Insert at (9+1)%11 = 10

Key 11: h(11) = 5 *// Occupied → keep incrementing until an empty slot*

// is found, inserted at 0

Key 39: h(39) = 6 *// Occupied → keep incrementing until an empty slot*

// is found, inserted at 1

Key 20: h(20) = 1 *// Occupied → Insert at(1+1)%11 = 2*

Key 16: h(16) = 4 *// Empty → Insert*

Key 5: h(5) = 4 *// Occupied → keep incrementing until an empty slot*

// is found, inserted at 3

Question 9:

Solve the previous problem again assuming collisions are handled by quadratic probing

Answer:

0	1	2	3	4	5	6	7	8	9	10
	11	20		39	44	88	12	23	13	94

Tracking the hash function insertion:

Key 12: h(12) = 7 *// Empty → Insert*

Key 44: h(44) = 5 *// Empty → Insert*

Key 13: h(13) = 9 *// Empty → Insert*

Key 88: h(88) = 5 *// Occupied → Insert at (5+1)%11 = 6*

Key 23: h(23) = 7 // Occupied → Insert at (7+1)%11 = 8

Key 94: h(94) = 6 // Occupied → Insert at (6+1)%11 → Occupied

// Insert at (6+4)%11 → 10

*Key 11: h(11) = 5 // Occupied → keep incrementing (i*i) until an empty slot*

// is found, inserted at 1

*Key 39: h(39) = 6 // Occupied → keep incrementing (i*i) until an empty slot*

// is found, inserted at 4

Key 20: h(20) = 1 // Occupied → Insert at(1+1)%11 = 2

Key 16: h(16) = 4 // Hash function doesn't generate one of the remaining

// 2 indices (0,3) so it won't be inserted

Key 5: h(5) = 4 // Hash function doesn't generate one of the remaining

// 2 indices (0,3) so it won't be inserted

Question 10:

Solve the previous problem again assuming collisions are handled by double hashing using the secondary hash function:

h'(k) = 7(k mod 7)

Answer:

0	1	2	3	4	5	6	7	8	9	10
88	11	39	5	16	44	94	12	20	13	23

Tracking the hash function insertion:

Key 12: h(12) = 7 // Empty → Insert

Key 44: h(44) = 5 // Empty → Insert

Key 13: h(13) = 9 // Empty → Insert

Key 88: h(88) = 0 *// Occupied → Insert at (5+1*28)%11 = 0*

Key 23: h(23) = 7 *// Occupied → Insert at (7+1*14)%11 = 10*

Key 94: h(94) = 6 *// Empty → Insert*

Key 11: h(11) = 5 *// Occupied → Insert at (5+i*28)%11, keep incrementing i*
// until an empty slot is found, inserted at 1

Key 39: h(39) = 6 *// Occupied → Insert at (6+i*28)%11, keep incrementing i*
// until an empty slot is found, inserted at 2

Key 20: h(20) = 1 *// Occupied → Insert at (1+i*42)%11, keep incrementing i*
// until an empty slot is found, inserted at 8

Key 16: h(16) = 4 *// Empty → Insert*

Key 5: h(5) = 4 *// Occupied → Insert at (4+i*35)%11, keep incrementing i*
// until an empty slot is found, inserted at 3

Question 11:

Describe how to perform a removal from a hash table that uses linear probing to resolve collisions where we do not use a special marker to represent deleted elements. That is, we must rearrange the contents so that it appears that the removed entry was never inserted in the first place.

Answer:

```
Algorithm Remove(key){
    Int index = h(key);
    if(hashTable(index) == key){
        hashTable(index) = null;
        return;
    }
```

```
        index = (index + 1 ) % hashTable.Size();
        while(hashTable(index) != null && hashTable(index) != key ){
                index = (index + 1 ) % hashTable.Size();
        }
        if(hashTable(index) == null){
                print("Key was not found.");
        } else {
                hashTable(index) = null;
                index = (index + 1 ) % hashTable.Size();
                while(h(hashTable(index)) < index && hashTable(index) != null){
                        hashTable(index-1) = hashTable(index);
                        hashTable(index) = null;
                        index = (index + 1 ) % hashTable.Size();
                }
        }
        return;
}
```

End of Sheet 5.