CSC 225 Fall 2017: Assignment 6

Due at the beginning of class on Thursday Nov. 30

Question	1	2	3	4	5
Marks					

1. Consider the minDegreeVertex method below that returns the vertex number of a minimum degree vertex of the graph.

```
public class Graph
{   int n; int [] [] A; int count;
   public int minDegreeVertex(int level, int start_vertex, int end_vertex)
   {
      int u, v, du, dv, i, mid;

      int size= end_vertex- start_vertex+1;

      if (start_vertex == end_vertex) return(start_vertex);
      mid= start_vertex + (end_vertex- start_vertex)/2;
      u= minDegreeVertex(level+1, start_vertex, mid);
      v= minDegreeVertex(level+1, mid+1, end_vertex);
      du=0;
      for (i=0; i < n; i++)
      {
         du+= A[u][i];   count++;
      }
      dv=0;
      for (i=0; i < n; i++)
      {
         dv+= A[v][i];   count++;
      }
      if (du <= dv) return(u);
      else return(v);
   }
```

For this question, *G* is a simple (no loops or multiple edges) undirected graph with *n* vertices and adjacency matrix A. The minDegreeVertex method is called like this:

minv= G.minDegreeVertex(0, 0, G.n-1);

(a) [5] The goal of part (a) to prove that the minDegreeVertex method returns the vertex number *minv* of a minimum degree vertex of the graph. To do this, prove by induction that each call to the method returns the vertex number of a minimum degree vertex v in the range start_vertex $\leq v \leq$ end_vertex. You may assume that $n = 2^k$ for this question, for some integer k. Since $n = 2^k$, the subproblems solved are of sizes 2^r for $0 \leq r \leq k$. The induction should be on *size* (the size of a subproblem) and not n for this proof.

(b) [5] Give a recurrence relation $C(n, s)$ that gives the number of times that G.count is incremented when a subproblem of size s is solved.
The size of a subproblem is given by:
size= end_vertex- start_vertex+1;

(c) [5] Assume $n = 2^k$ for some integer k. Solve your recurrence relation from (b) using repeated substitution to get a closed formula for the value for $C(n, 2^r)$ where $0 \leq r \leq k$.

(d) [5] Either prove by induction that your closed formula from (c) gives the number of times that G.count is incremented or carry out the steps of an induction proof until the proof fails.

(e) [5] The value of G.count is the number of times that an entry of the adjacency matrix is accessed by the method. Does this method do a minimum possible number of accesses to entries of the adjacency matrix? This question is asking about an exact count and not a Big Oh type of analysis. If you say yes, justify your answer. If you say no, give java code for an algorithm that is optimal.

(f) [5] Let $M(n)$ be the minimum number of accesses required to entries of the adjacency matrix in the worst case. Is $M(n) \in \Theta(C(n, n))$? Justify your answer.

2. For question 2, print the last page of this assignment and show your solutions on it. For parts (a) and (b), suppose this data is inserted into a hash table of size 17 in the order given.

574	460	107	374	871	421	513
579	207	29	2	811	81	5

Use the hash function (syntax as for C or Java)
hash(x)= (((x / 100) % 10) + ((x / 10) % 10) + (x % 10)) % 17;

(a) [7] Draw a picture of the final data structure assuming that collisions are handled using buckets (each hash value has a linked list containing the data values hashing to that position). Assume that insertions are done at the front of the list when a data value is added to a bucket.

(b) [8] Suppose instead the data values are contained in the array H[0..16], and that collisions are resolved by finding the first empty slot past the location that a key hashes to (wrap around from H[16] back to H[0]). Draw the contents of H after these insertions. Use NULL to indicate empty positions in the hash table.

3. Suppose that a hashing strategy is designed so that it starts with an initial hash table size of H= 8. You may assume that only insertions are performed (no deletions). Any time the hash table is going to be more than 50 percent full (when an attempt is made to add item $H/2 + 1$ to a table of size H), the hash table size is doubled to $2H$, and then the $H/2$ keys in the previous hash table are rehashed using $H/2$ **extraneous** key insertions into the new table of size $2H$. The key insertions used to initially place each key into the hash table are called **necessary** key insertions (these are not extraneous).

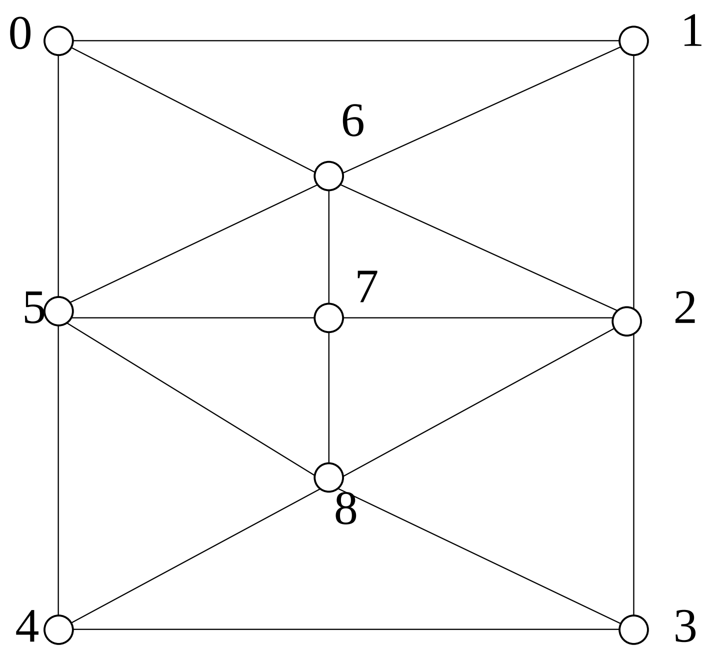
(a) [5] Derive a recurrence relation E(H) for the number of extraneous key insertions that have occurred in total up until the point in time that the hash table size is H. Explain where the terms in the recurrence relation are coming from.

(b) [5] Solve your recurrence relation from (a) using repeated substitution.

(c) [5] The table of size H can be used until up to $n = H/2$ keys have been inserted. Choose a function f(n) that is as simple as possible such that $E(H)$ is in $\Theta(f(n))$ and then prove that $E(H) \in \Theta(f(n))$.

(d) [5] Is the number of extraneous hashes a good choice for a proxy operation for evaluating the amount of work that has been done in maintaining the hash table? Justify your answer. If your answer is yes, then explain what also has to be done besides extraneous hashes in order to maintain the hash table and why the total amount of work is proportional to the number of extraneous hashes. If your answer is no, explain what is not being taken into account.

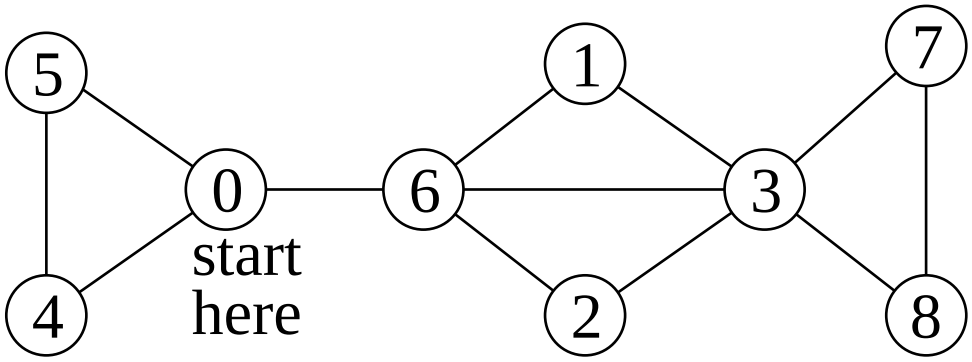
4.(a) [10] Perform BFS (Breadth First Search) on the graph given below. Start at vertex 0. When traversing the neighbours of a vertex, traverse them **in numerical order**. Show all your work including the contents the queue, the parent array, the BFI (breadth first index) and the level.



	0	1	2	3	4	5	6	7	8
Queue:									
Parent:									
BFI:									
Level:									

(b) [5] Mark the edges of the BFS tree on the picture and orient them so that the edge is directed from the child to the parent.

5.(a) [15] Perform DFS (Depth First Search) on the graph given below. Start at vertex 0. When traversing the neighbours of a vertex, traverse them **in numerical order**. Show all your work including the contents of the stack (at each step), the parent array and the DFI array.



	0	1	2	3	4	5	6	7	8
Parent:									
DFI:									

Stack contents at each step (at step i, the ith edge is added to the DFS tree):

Step 0	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8

(b) [5] Mark the edges of the DFS tree on the picture and orient them so that the edge is directed from the child to the parent.

Use this page for question 2:
hash(x)= (((x / 100) % 10) + ((x / 10) % 10) + (x % 10)) % 17.

x	574	460	107	374	871	421	513
hash(x)							

x	579	207	29	2	811	81	5
hash(x)							

(a) The linked lists:

	start
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

(b) The hash table:

	key value
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	