Roll Number:_____ Number of Pages: 02

Thapar Institute of Engineering and Technology, Patiala

Department of Computer Science and Engineering

END SEMESTER EXAMINATION

B. E. (COE Third Year): Semester-I (2020/21)	Course Code: UCS301 Course Name: Data Structures
February 05, 2021	Friday, 11:00 Hrs – 13:00 Hrs
Time: 2 Hours, M. Marks: 50	Name of Faculty: MAK, SUG, SP, AA, MK

Note: Attempt any 5 questions. Attempt all questions (subparts) in sequence at one place. Assume missing data, if any, suitably.

Q.1. (a) Determine the shortest paths to all the vertices which can be reached from source vertex '**A**' (**Fig. 1**) using Dijkstra's shortest path algorithm. Illustrate each intermediate step. (6)

(b) Write sequence in which nodes of the graph (**Fig. 1**) have been traversed using BFS, starting from vertex '**A**'. To make a unique solution, assume that whenever you faced with a decision of which node to pick from a set of nodes, pick the one whose label occurs earliest in the alphabet. (4)

Q.2. (a) Draw a Binary Search Tree (**BST**) by sequentially inserting the following elements: (4)

71, 82, 76, 61, 66, 72, 74, 111, 60, and **63**

(b) Delete **76** from the **BST** obtained in **Q.2.(a)** and draw the resultant **BST**. (1)

(c) Considering the tree obtained after **Q.2.(b)** to be an **AVL** Tree, insert **50**, **65**, **55**, and **70** sequentially into it. (3)

(d) Delete **111** from the **AVL** obtained in **Q.2.(c)** and draw the resultant **AVL** tree. (2)

Note: After each insertion or deletion that includes rotation, redraw the respective tree.

Q.3. (a) Consider the worst case of a merge sort algorithm takes **24 seconds** for an input of **size 256**. Calculate the approximate maximum input size of a problem that can be solved in **2 minutes**? (3)

(b) Convert the infix expression $\boxed{A + B - C * D/E + F - G/H}$ into an equivalent postfix expression using stacks. Show contents of the stack at each intermediate step. (7)

Q.4. (a) Consider a hash table of size $m = 9$ and a corresponding hash function $h(k) = k \bmod m$. Compute the locations to which the keys **14**, **19**, **13**, **4**, **5**, **23**, **6**, and **15** are mapped using **linear probing** collision resolution technique. Draw the resultant hash table and determine the total number of collisions occurs. (5)

(b) Write an efficient algorithm(s) or function(s) to print a pattern as shown in **Fig. 2** for a particular value of n using recursion. Here n is the maximum number of '*' that will appear in the horizontal diagonal. For example in **Fig. 2**, input is $n = 4$. (5)

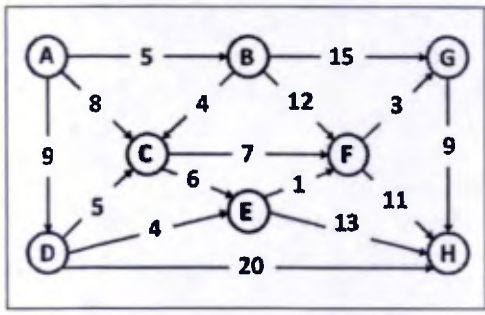

Fig. 1

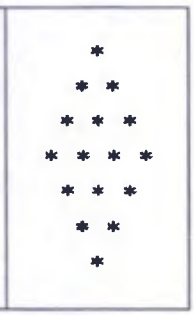

Fig. 2

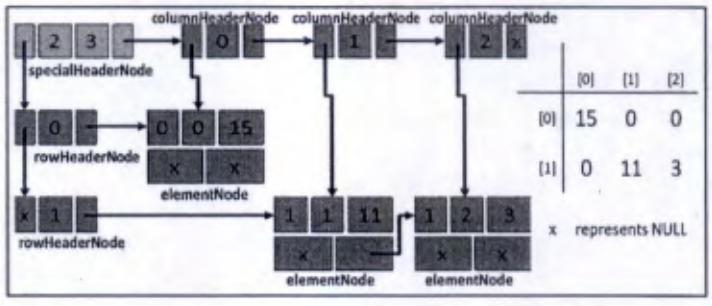

Fig. 3

Q.5. (a) Assuming the graph in **Fig. 1** to be un-directed broadcast graph where vertex '**A**' is the broadcast station. The task is to efficiently transfer a piece of information to all the listeners, i.e. vertices other than the broadcast station. The broadcast station sends a single copy of the broadcast message into the network. Each listener forwards the message to any single neighbor, excluding the neighbor that just sent it the message. No listener is allowed to see more than one copy of any message, and all the listeners should see a copy of the message. (5)

Apply an efficient solution for the said problem on the broadcast graph shown in **Fig. 1**, illustrating all the intermediate steps.

(b) Heapsort algorithm is executed on an array $A[\,] = [17, 20, 13, 12, 19, 14, 11, 16, 15, 18]$ to arrange its elements in ascending order. What would be the contents of array $A[\,]$ after three passes of the heap sort algorithm (show all the intermediate steps)? (5)

Q.6. A Sparse matrix is represented in the form of a linked list, having four different types of nodes (**Fig. 3**), namely **elementNode**, **columnHeaderNode**, **rowHeaderNode**, and **specialHeaderNode**. Write an efficient algorithm/pseudo-code for these two functions following the given function signatures and template (**Fig. 4**). Assume the functions specified in the template are already implemented. (10)

(a) Compute sum of two sparse matrices.

specialHeaderNode* sparseMatAdd (**specialHeaderNode** *mat1, **specialHeaderNode** *mat2)

(b) Insert new **elementNode** at its correct place in the matrix.

specialHeaderNode* insert(**specialHeaderNode** *mat1, **elementNode** *e)

```
1.  struct elementNode
2.  { int rowNum, colNum, val;
3.    elementNode *nextEleNodeInRow;
4.    elementNode *nextEleNodeInCol;
5.  };
6.  struct columnHeaderNode
7.  { int columnNum;
8.    elementNode *firstEleInCol;
9.    columnHeaderNode *nextColHeader;
10. };
```

```
11. struct rowHeaderNode
12. { int rowNum;
13.   elementNode *firstEleInRow;
14.   rowHeaderNode *nextRowHeader;
15. };
16. struct specialHeaderNode
17. { int totalRow, totalCol;
18.   columnHeaderNode *ColHeader;
19.   rowHeaderNode *RowHeader;
20. };
```

```
21. //Function that creates, initializes, and returns a new elementNode. It initializes both the
22. //member pointers in elementNode structure to NULL.
23. elementNode* newElementNode(int row, int column, int value);
24.
25. //Function that creates and initializes row as well as column header nodes for a particular
26. //matrix. It initializes all the member pointers in columnHeaderNode and rowHeaderNode
27. //structures to NULL.
28. specialHeaderNode* generateRowColHeaders(specialHeaderNode *mat1, int totalRow, int totalCol);
```

Fig. 4. Template for Q.6.

Q.7. Execute the algorithm given in **Fig. 5** on an array **A[1..15] = [0, 70, 74, 52, 86, 84, 62, 90, 56, 91, 75, 94, 89, 58, 78, 88]**. Write the final contents of an array **A[ ]** and stack **S1**. (10)

```
//Q1 and Q2 are empty Queues, S1 and S2 are empty Stacks, an Integer i is initialized to 1, j is an
//Integer, and flag is a Boolean.
1.  Build_Min_Heap(A);
2.  if (A.heap_size)
3.  {  Q1.enqueue(A[i]);
4.     while (!Q1.empty() || !Q2.empty())
5.     {  flag = !true;
6.        while (!Q1.empty())
7.        {  i++;
8.           if (!flag)
9.           {  S1.push(Q1.front());
10.             if (!S2.empty())
11.               S1.push(S2.top());
12.             flag = !flag ? !flag : flag; }
13.          if (Left(i) <= A.heap_size)
14.            Q2.enqueue(A[Left(i)]);
15.          if (Right(i) <= A.heap_size)
16.          {  Q2.enqueue(A[Right(i)]);
17.             S2.push(A[Right(i)]);  }
18.          Q1.dequeue(); }
19.       flag = true;
20.       while (!Q2.empty())
21.       {  i++;
22.          if (flag)
23.          {  S1.push(Q2.front());
24.             S1.push(S2.top());
25.             flag = !flag ? !flag : !flag; }
26.          if (Left(i) <= A.heap_size)
27.            Q1.enqueue(A[Left(i)]);
28.          if (Right(i) <= A.heap_size)
29.          {  Q1.enqueue(A[Right(i)]);
30.             S2.push(A[Right(i)]);
31.          }
32.          Q2.dequeue(); } } }
```

Fig. 5. Algorithm for Q.7.

----------ALL THE BEST----------