Register Number

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |

|  |
| --- |
| **Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110**  (An Autonomous Institution, Affiliated to Anna University, Chennai) |
| Department of Computer Science and Engineering  **Continuous Assessment Test – II**  **Question Paper** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Degree & Branch** | BE (CSE) | | | | **Semester** | III |
| **Subject Code & Name** | UCS1302 Data Structures | | | | **Regulation:** | **2018** |
| **Academic Year** | 2021-2021  ODD | **Batch** | 2020-2024 | **Date** | **01.12.2021** | **FN** |
| **Time: 90 Minutes** | **Answer All Questions** | | | | **Maximum: 50 Marks** | |

Part – A (6×2 = 12 Marks)

|  |  |  |
| --- | --- | --- |
| <KL2> | 1. Show the expression tree for the arithmetic expression a\*b/(c+d/e)   Ans: | <CO3> |
| <KL1> | 1. What is the maximum number of nodes in a binary tree of height n.   Ans: 2n+1-1 | <CO3> |
| <KL2> | 1. Outline the recursive implementation of a function that returns the smallest element in the binary search tree   Ans:  int FindMin(struct bstNode \*t)  {  if (t != NULL)  { if (t->left != NULL)  return FindMin(t->left);  return t->data;  }  } | <CO3> |
| <KL1> | 1. Define the properties of AVL tree. Give an example of an AVL tree.   Ans:  AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes | <CO3> |
| <KL2> | 1. Outline the algorithm for printing the degree of each vertex in a given undirected graph G represented by adjacency matrix.   Ans:  Algorithm PrintDegree( int adj[ ] [ ], int n)  Input: Adjacency matrix of a graph adj with n vertices  Output: Degree of each vertex is printed  Var: int degree, i, j  1. FOR (i = 1 to n)  degree ← 0  FOR (j = 1 to n)  degree ← degree + adj[i][j]  ENDFOR  print “degree of vertex i is “+degree  ENDFOR  2. return | <CO4> |
| <KL3> | 6. Apply BFS to the following graph and write the nodes visited, starting from node 0.  Understanding the Basics of Graph Data Structure  Ans: 0 1 4 2 3  Other answers:  0 4 1 3 2  0 1 4 3 2 | <CO4> |

Part – B (3×6 = 18 Marks)

|  |  |  |
| --- | --- | --- |
| <KL3> | 7. Develop a C function for in order traversal of a binary tree. Trace the function, for the following binary tree.  Binary search tree - Wikipedia  Ans:  void inorder(struct tree \*t) -> Traverse in the order LDR  {  if(t->left!=NULL)  inorder(t->left);  printf("\n Data : %d",t->data);  if(t->right!=NULL)  inorder(t->right);  } | <CO3> |
| <KL2> | 8. Outline the algorithm for searching an element in a Binary search tree.  Ans:  Algorithm find (struct tree \*t, int key)  Input: pointer to Binary search tree t and key to be searched  Output: returns the pointer to the node containing key; if key is not found return NULL  1. if (t==NULL)  return NULL  endif  2. if(key == t->data)  return t  endif  3. if (key < t->data)  return find(t->left, key)  endif  4. if (key > t->data)  return find(t->right, key)  endif | <CO3> |
| <KL3> | 9. Given the following array of integers, apply selection sort and write the contents of the array after FOUR passes of the outermost loop. Write the contents of the array after ONE pass of shell sort, using interval = 3.  A[ ] = {22, 11, 34, -5, 3, 40, 9, 16, 6}  Ans:  Selection Sort  After Pass1  -5,11,34,22,3,40,9,16,6  After 2nd Pass  -5,3,34,22,11,40,9,16,6  After 3rd Pass  -5,3,6,22,11,40,9,16,34  After 4th pass  -5, 3,6,9,11,40,22,16,34 | <CO5> |

Part – C (2×10 = 20 Marks)

|  |  |  |
| --- | --- | --- |
| <KL3> | 10. Given input items 25,54,46,28,19,45,27,48,75. Apply the following operations showing the tree diagram for each step.   1. Construct BST. Delete 46   Ans:     1. Construct AVL tree. Insert 13   Ans: | <CO3> |
| (OR) | | |
| <KL3> | 11. Develop the ADT for PriorityQ using a MinHeap and the algorithm for DELMIN(Q). Trace your algorithm for the following heap. Clearly show the array as well as the tree representations of the heap during trace.  17  20 26  24 31 27 30  49 52  Ans:  struct PQ  {  int elt[100];  int size;  int ctptr;  };  void init(struct PQ \*p,int s)  {  p->size=s; p->ctptr=0; p->elt[0]=-10;  }  int del(struct PQ \*p)  {  int min,last,i,j,child;  min=p->elt[1];  P->elt[1]=p->elt[p->ctptr];  p->ctptr--;  for(i=1;2\*i<=p->ctptr;i=child)  {  child=2\*i;  if(p->elt[child]<p->elt[i] || p->elt[child+1]<p->elt[i])  { if(p->elt[child+1]<p->elt[child])  child++;  }  if(p->elt[i]>p->elt[child])  { t=p->elt[i]; p->elt[i]=p->elt[child]; p->elt[child]=t; }  }  return min;  } | <CO3> |
| <KL2> | 12. Outline the algorithm for *dfs(G)* that visits all the vertices of a directed graph G. Trace your algorithm for a directed graph represented as an adjacency matrix given below.   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | A | B | C | D | | A | 0 | 1 | 1 | 0 | | B | 0 | 0 | 1 | 1 | | C | 0 | 0 | 0 | 0 | | D | 0 | 0 | 0 | 0 |   Ans:  *Algorithm DFS\_visit(G)*  *Input: Graph G*  *Output: DFS of G*  1. FOR each vertex v ∈ G  IF (v is not visited)  dfs(v);  ENDIF  ENDFOR    Algorithm dfs (Node v)  Input: Start node v of a graph G  Output: DFS of G starting from node v  DS: Stack S  1. IF (v == null)  return;  ENDIF  2. push(S, v);  3. WHILE (S is not empty)  v = pop(S);  IF (v has not yet been visited)  visit&mark(v)  ENDIF  FOR (each w adjacent to v)  if (w has not yet been visited and is not yet stacked)  push(S, v);  ENDIF  ENDFOR  ENDWHILE  4. return  DFS(0)  ABCD  Other answers  ACBD  ABDC | <CO4> |
| (OR) | | |
| <KL2> | 13. Given an array of names of the cities sorted in lexical order. Outline the algorithm or C  function that searches for a city in the array. Compute the time complexity.  Ans:  void BinarySearch(char words[][], int beg, int end, char key[])  {  while(beg<end)  {  mid=(beg+end)/2  if(strcmp(key,words[mid])>0)  {  beg=mid+1  search(words,beg,end,key)  }  if(strcmp(key,words[mid])<0)  {  end=mid-1  search(words,beg,end,key)  }  if(strcmp(key,words[mid])==0)  print “Key found”  }  }  Time Complexity  Size of the array is n  After Iteration 1, length of array is n/21  After Iteration 2, length of array is n/22  After Iteration 3, length of array is n/23  After k iterations, length of array becomes 1 which is equal to n/2k  Therefore, n/2k = 1  2k = n  K = log n  O(log n) | <CO4> |

--------------