## C:\Users\admin\Desktop\JEC LOGO.png

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CS3301– DATA STRUCTURES**

Question Bank

II YEAR A & B / BATCH : 2022 -26

**Vision of Institution**

To build Jeppiaar Engineering College as an Institution of Academic Excellence in Technical education and Management education and to become a World Class University.

**Mission of Institution**

|  |  |
| --- | --- |
| **M1** | To excel in teaching and **learning, research and innovation** by promoting the principles of scientific analysis and creative thinking |
| **M2** | To participate in the production, **development and dissemination of knowledge** and interact with **national and international communities** |
| **M3** | To equip students with **values, ethics and life skills** needed to enrich their lives and enable them to meaningfully contribute to the **progress of society** |
| **M4** | To prepare students **for higher studies and lifelong learning**, enrich them with the **practical and entrepreneurial skills** necessary to excel as future professionals and contribute to **Nation’s economy** |

## Program Outcomes (POs)

|  |  |
| --- | --- |
| **PO1** | **Engineering Knowledge**: Apply the Knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. |
| **PO2** | **Problem analysis**: Identify, formulate, review research literature, and analyze  complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. |
| **PO3** | **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations |
| **PO4** | **Conduct investigations of complex problems**: Use research-based Knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. |
| **PO5** | **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations. |
| **PO6** | **The engineer and society**: Apply reasoning informed by the contextual Knowledge to assess societal, health, safety, legal and cultural issues and the  consequent responsibilities relevant to the professional engineering practice. |
| **PO7** | **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the Knowledge of, and need for sustainable development. |
| **PO8** | **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| **PO9** | **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings. |

|  |  |
| --- | --- |
| **PO10** | **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective  presentations, and give and receive clear instructions. |
| **PO11** | **Project management and finance**: Demonstrate Knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments. |
| **PO12** | **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change. |

**Vision of Department**

To emerge as a globally prominent department, developing ethical computer professionals, innovators and entrepreneurs with academic excellence through quality education and research.

**Mission of Department**

|  |  |
| --- | --- |
| **M1** | To create **computer professionals** with an ability to identify and **formulate the engineering problems** and also to provide **innovative solutions** through **effective teaching learning process.** |
| **M2** | To **strengthen the core-competence** in computer science and engineering and to create an ability to **interact** effectively with industries. |
| **M3** | To produce engineers with good professional sKills, **ethical values** and life skills for the  **betterment of the society.** |
| **M4** | To encourage students towards **continuous and higher level learning** on technological advancements and provide a platform for **employment and self-employment.** |

## Program Educational Objectives (PEOs)

|  |  |
| --- | --- |
| **PEO1** | **To address the real time complex engineering problems using innovative approach with strong core computing skills.** |
| **PEO2** | **To apply core-analytical Knowledge and appropriate techniques and provide solutions to real time challenges of national and global society** |
| **PEO3** | **Apply ethical Knowledge for professional excellence and leadership for the betterment of the society.** |
| **PEO4** | **Develop life-long learning skills needed for better employment and entrepreneurship** |

**SYLLABUS**

**UNIT I LISTS 9**

Abstract Data Types (ADTs) – List ADT – Array-based implementation – Linked list implementation – Singly linked lists – Circularly linked lists – Doubly-linked lists – Applications of lists – Polynomial ADT – Radix Sort – Multilists.

**UNIT II STACKS AND QUEUES 9**

Stack ADT – Operations – Applications – Balancing Symbols – Evaluating arithmetic expressions- Infix to Postfix conversion – Function Calls – Queue ADT – Operations – Circular Queue – DeQueue – Applications of Queues.

**UNIT III TREES 9**

Tree ADT – Tree Traversals - Binary Tree ADT – Expression trees – Binary Search Tree ADT – AVL Trees – Priority Queue (Heaps) – Binary Heap.

**UNIT IV MULTIWAY SEARCH TREES AND GRAPHS 9**

B-Tree – B+ Tree – Graph Definition – Representation of Graphs – Types of Graph - Breadth-first traversal – Depth-first traversal –– Bi-connectivity – Euler circuits – Topological Sort – Dijkstra's algorithm – Minimum Spanning Tree – Prim's algorithm – Kruskal's algorithm

**UNIT V SEARCHING, SORTING AND HASHING TECHNIQUES 9**

Searching – Linear Search – Binary Search. Sorting – Bubble sort – Selection sort – Insertion sort

– Shell sort –. Merge Sort – Hashing – Hash Functions – Separate Chaining – Open Addressing – Rehashing – Extendible Hashing.

**TEXT BOOKS:**

1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,2005.
2. Kamthane, Introduction to Data Structures in C, 1st Edition, Pearson Education, 2007

**REFERENCES:**

1. Langsam, Augenstein and Tanenbaum, Data Structures Using C and C++, 2nd Edition, Pearson Education, 2015.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L.Rivest, Clifford Stein, Introduction to Algorithms", Fourth Edition, Mcgraw Hill/ MIT Press, 2022.
3. Alfred V. Aho, Jeffrey D. Ullman,John E. Hopcroft ,Data Structures and Algorithms, 1st edition, Pearson, 2002.
4. Kruse, Data Structures and Program Design in C, 2nd Edition, Pearson Education, 2006.

## Course Outcomes (COs)

|  |  |
| --- | --- |
| C311.1 | 1. Implement abstract data type for List linear data structure and apply them to problem solutions |
| C311.2 | 2. Implement abstract data type for Stack and Queue data structure and apply them to problem solutions |
| C311.3 | 3. Implement abstract data type for Tree non list linear data structure and apply them to problem solutions |
| C311.4 | 4. Implement abstract data type for Graph non list linear data structure and apply them to problem solutions |
| C311.5 | Analyze the various sorting and searching algorithms and hashing techniques |

**BLOOM TAXANOMY LEVELS**

**BTL 6: Creating**

**BTL 5: Evaluating**

**BTL 4: Analyzing**

**BTL 3: Applying**

**BTL 2: Understanding**

**BTL 1: Remembering**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **UNIT NO** | **TEXT/ REFERENCE BOOK** | **PAGE NO** |
| **UNIT -I** | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. 2. Reema Thareja, “Data Structures Using C”, Second Edition , Oxford University Press, 2011 |  |
| **UNIT -II** | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. 2. Reema Thareja, “Data Structures Using C”, Second Edition , Oxford   University Press, 2011 |  |
| **UNIT -III** | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. 2. Reema Thareja, “Data Structures Using C”, Second Edition , Oxford University Press, 2011 |  |
| **UNIT -IV** | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. 2. Reema Thareja, “Data Structures Using C”, Second Edition , Oxford University Press, 2011 |  |
| **UNIT -V** | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. 2. Reema Thareja, “Data Structures Using C”, Second Edition , Oxford   University Press, 2011 |  |

# UNIT I

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Topics** | | | | **Text / Reference book** | **Page No.** |
| Abstract Data Types (ADTs) | | | | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. | 57 |
| List ADT – array-based implementation | | | | 58-59 |
| linked list implementation | | | | 59 |
| singly linked lists | | | | 60 |
| circularly linked lists | | | | 68 |
| doubly-linked lists | | | | 67 |
| applications | of | lists | –Polynomial | 73 |
| Manipulation |  |  |  |  |
| All operations (Insertion, Deletion, | | | | 73 |
| Merge, Traversal). | | | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| S.  No  . | Question | Course Outcom e | Blooms Taxanom y Level |
| 1 | **What is a data structure?**   * A data structure is a method for organizing and storing data which would allow efficient   data retrieval and usage.   * A data structure is a way of organizing data that considers not only the items stored, but   also their relationships to each other. | C311.1 | BTL1 |
| 2 | **Why do we need data structures?**   * Data structures allow us to achieve an important goal: component reuse. * Once data structure has been implemented, it can be used again and again in   various applications. | C311.1 | BTL 1 |
| 3 | **List some common data structures.**   * Stacks * Queues * Lists * Trees * Graphs * Tables | C311.1 | BTL 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | |  |  |
| 4 | **How data structures are classified?**  Data structures are classified into two categories based on how the data items are  operated:   1. Primitive data structure 2. Non-Primitive data structure 3. Linear data structure 4. Non-linear data structure | | | | C311.1 | BTL 1 |
| 5 | **Differentiate linear and non-linear data structure.** | | | | C311.1 | BTL 2 |
|  | **Linear data structure** | **Non-linear data structure** |  |
| Data are arranged in linear or sequential manner | Data are not arranged in linear manner |
| Every items is related to its previous  and next item | Every item is attached with many other  items |
| Data items can be traversed in a  single run. | Data items cannot be traversed in a  single run. |
| Implementation is easy | Implementation is difficult. |
| Example: array, stack, queue, linked  list | Example: tree, graph |
| 6 | **Define ADT (Abstract Data Type)**  An abstract data type (ADT) is a set of operations and mathematical abstractions , which  can be viewed as how the set of operations is implemented. Objects like lists, sets and graphs, along with their operation, can be viewed as abstract data types, just as integers, real numbers and Booleans. | | | | C311.1 | BTL 1 |
| 7 | **Mention the features of ADT.**  a. Modularity   1. Divide program into small functions 2. Easy to debug and maintain 3. Easy to modify   b. Reuse  i. Define some operations only once and reuse them in future  c. Easy to change the implementation | | | | C311.1 | BTL 2 |
| 8 | **Define List ADT**  A list is a sequence of zero or more elements of a given type. The list is represented as  sequence of elements separated by comma. A1, A2, A3…..AN  Where N>0 and A is of type element | | | | C311.1 | BTL 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | **What are the ways of implementing linked list?**  The list can be implemented in the following ways:   1. Array implementation 2. Linked-list implementation 3. Cursor implementation | C311.1 | BTL 1 |
| 10 | **What are the types of linked lists?**  There are three types   1. Singly linked list 2. Doubly linked list 3. Circularly linked list | C311.1 | BTL 1 |
| 11 | **How the singly linked lists can be represented?**    Each node has two elements   1. Data 2. Next | C311.1 | BTL 1 |
| 12 | **How the doubly linked list can be represented?**    Doubly linked list is a collection of nodes where nodes are connected by forwarded and  backward link.  Each node has three fields:   1. Address of previous node 2. Data 3. Address of next node. | C311.1 | BTL 1 |
| 13 | **What are benefits of ADT?**   1. Code is easier to understand 2. Implementation of ADT can be changed without requiring changes to the program   that uses the ADT | C311.1 | BTL 1 |
| 14 | **When singly linked list can be represented as circular linked list?**  In a singly linked list, all the nodes are connected with forward links to the next nodes in  the list. The last node has a next field, NULL. In order to implement the circularly linked | C311.1 | BTL 1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | lists from singly linked lists, the last node’s next field is connected to the first node. |  |  |
| 15 | **When doubly linked list can be represented as circular linked list?**  In a doubly linked list, all nodes are connected with forward and backward links to the  next and previous nodes respectively. In order to implement circular linked lists from  doubly linked lists, the first node’s previous field is connected to the last node and the  last node’s next field is connected to the first node. | C311.1 | BTL 1 |
| 16 | **Where cursor implementation can be used?**  The cursor implementation of lists is used by many languages such as BASIC and  FORTRAN that do not support pointers. The two important features of the cursor  implementation of linked are as follows:   * The data are stored in a collection of structures. Each structure contains data and a   index to the next structure.   * A new structure can be obtained from the system’s global memory by a call to   cursorSpace array. | C311.1 | BTL 1 |
| 17 | **List down the applications of List.**   1. Representation of polynomial ADT 2. Used in radix and bubble sorting 3. In a FAT file system, the metadata of a large file is organized as a linked list of FAT entries. 4. Simple memory allocators use a free list of unused memory regions, basically a   linked list with the list pointer inside the free memory itself. | C311.1 | BTL 1 |
| 18 | **What are the advantages of linked list?(apr/may 2019)**   1. Save memory space and easy to maintain 2. It is possible to retrieve the element at a particular index 3. It is possible to traverse the list in the order of increasing index. 4. It is possible to change the element at a particular index to a different value,without affecting any other elements. | C311.1 | BTL 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| 19 | **Mention the demerits of linked list**   1. It is not possible to go backwards through the list 2. Unable to jump to the beginning of list from the end. | C311.1 | BTL 2 |
| 20 | **The polynomial equation can be represented with linked list as follows:**  struct polynomial  {  int coefficient; int exponent;  struct polynomial \*next;  }; | C311.1 | BTL 2 |
| 21 | **What are the operations performed in list?**  The following operations can be performed on a list  i. Insertion   1. Insert at beginning 2. Insert at end 3. Insert after specific node 4. Insert before specific node   ii. Deletion   1. Delete at beginning 2. Delete at end 3. Delete after specific node 4. Delete before specific node 5. Merging 6. Traversal | C311.1 | BTL 1 |
| 22 | **What are the merits and demerits of array implementation of lists?**  Merits   * Fast, random access of elements * Memory efficient – very less amount of memory is required   Demerits   * Insertion and deletion operations are very slow since the elements should be   moved.   * Redundant memory space – difficult to estimate the size of array. | C311.1 | BTL 1 |
| 23 | **What is a circular linked list?**  A circular linked list is a special type of linked list that supports traversing from the end  of the list to the beginning by making the last node point back to the head of the list. | C311.1 | BTL 1 |

|  |  |  |
| --- | --- | --- |
| Coefficient | Exponent | Next node link |

|  |  |  |  |
| --- | --- | --- | --- |
| 24 | **What are the advantages in the array implementation of list?**   1. Print list operation can be carried out at the linear time 2. Find Kth operation takes a constant time | C311.1 | BTL 1 |
| 25 | **What is the need for the header?**  Header of the linked list is the first element in the list and it stores the number of elements in the list. It points to the first data element of the list. | C311.1 | BTL 1 |
| 26 | **List three examples that uses linked list?**  a. Polynomial ADT b.Radix sort c.Multi lists | C311.1 | BTL 1 |
| 27 | **List out the different ways to implement the list?**   1. Array Based Implementation 2. Linked list Implementation    1. Singly linked list    2. Doubly linked list    3. Cursor based linked list | C311.1 | BTL 1 |
| 28 | **Write the routine for insertion operation of singly linked list.**  Void Insert (ElementType X, List L, Position P)  {Position TmpCell; TmpCell=malloc(sizeof(struct Node)); if(TmpCell==NULL)  FatalError(“Out of space!!!”);  TmpCell->Element =X; TmpCell->Next=P->Next; P->Next=TmpCell;  } | C311.1 | BTL 5 |
| 29 | **Advantages of Array over Linked List.**   1. Array has a specific address for each element stored in it and thus we can access any memory directly. 2. As we know the position of the middle element and other elements are easily accessible too, we can easily perform BINARY SEARCH in array. | C311.1 | BTL 5 |
| 30 | **Disadvantages of Array over Linked List.**   1. Total number of elements need to be mentioned or the memory allocation needs to be done at the time of array creation 2. The size of array, once mentioned, cannot be increased in the program. If number of elements entered exceeds the | C311.1 | BTL 5 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | size of the array ARRAY OVERFLOW EXCEPTION occurs. |  |  |
| 31 | **Advantages of Linked List over Array.**   1. Size of the list doesn't need to be mentioned at the beginning of the program. 2. As the linked list doesn't have a size limit, we can go on adding new nodes (elements) and increasing the size of the list to any extent. | C311.1 | BTL 5 |
| 32 | **Disadvantages of Linked List over Array.**   1. Nodes do not have their own address. Only the address of the first node is stored and in order to reach any node, we need to traverse the whole list from beginning to the desired node. 2. As all Nodes don't have their particular address, BINARY SEARCH cannot be performed | C311.1 | BTL 5 |
| 33 | **Difference between linear linked list and circular linked list(apr/may 2019)**  **Linked list** are used to create trees and graphs. **Circular linked list** : In **circular linked list** the last node address part holds the address **of the** first node hence forming a **circular** chain like structure. **Linked**  **list** is a **linear** data structure which consists of group of nodes in a sequence | C311.1 |  |
| 34 | **Enlist the various operations that can be performed on data structure.**  Various operations that can be performed on the data structure are • Create • Insertion of element • Deletion of element • Searching for the desired element • Sorting the elements in the data structure • Reversing  the list of elements. | C311.1 |  |
| 35 | **What are all not concerned in an ADT?**  The abstract data type is a triple of D i.e. set of axioms, F-set of functions and A-Axioms in which only what is to be done is mentioned  but how is to be done is not mentioned. Thus ADT is not concerned with implementation details | C311.1 |  |
| 37 | **List out the areas in which data structures are applied extensively.**  Following are the areas in which data structures are applied extensively  Operating system- the data structures like priority queues are used for scheduling the jobs in the operating system.   * Compiler design- the tree data structure is used in parsing the source program. Stack data structure is used in handling recursive calls. * Database management system- The file data structure is used in database management systems. Sorting and searching techniques can be applied on these data in the file. * Numerical analysis package- the array is used to perform the numerical analysis on the given set of data. * Graphics- the array and the linked list are useful in graphics applications. * Artificial intelligence- the graph and trees are used for the applications like building expression trees, game playing. | C311.1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 38 | **What are the pitfall encountered in singly linked list?**  Following are the pitfall encountered in singly linked list • The singly linked list has only forward pointer and no backward link is provided. Hence the traversing of the list is possible only in one direction. Backward traversing is not possible. • Insertion and deletion operations are less efficient because for inserting the element at desired position the list needs to be traversed. Similarly, traversing of the list is  required for locating the element which needs to be deleted. | C311.1 |  |
| 40 | **Write down the steps to modify a node in linked lists**.   * Enter the position of the node which is to be modified. * Enter the new value for the node to be modified. * Search the corresponding node in the linked list. * Replace the original value of that node by a new value   . ➢ Display the messages as “ the node is modified”. | C311.1 |  |
| 41 | **State the properties of LIST abstract data type with suitable example.**  Various properties of LIST abstract data type are   1. It is linear data structure in which the elements are arranged adjacent to each other. 2. )It allows to store single variable polynomial 3. (iii)If the LIST is implemented using dynamic memory then it is called linked list. Example of LIST are- stacks, queues,   linked list | C311.1 |  |
| 42 | **Why is the linked list used for polynomial arithmetic?**  We can have separate coefficient and exponent fields for representing each term of polynomial. Hence there is no limit for exponent. We can  have any number as an exponent. | C311.1 |  |
| 43 | **What is the basic purpose of header of the linked list?**  The header node is the very first node of the linked list. Sometimes a dummy value such - 999 is stored in the data field of header node. This  node is useful for getting the starting address of the linked list. | C311.1 |  |
| 44 | **What is static linked list? State any two applications of it.**   * The linked list structure which can be represented using arrays is called static linked list. * It is easy to implement, hence for creation of small databases, it is useful   .➢ The searching of any record is efficient, hence the applications in  which the record need to be searched quickly, the static linked list are used. | C311.1 |  |
| **PART-B** | | | |
| 1 | **Explain the various operations of the list ADT with examples** | C311.1 | BTL 2 |
| 2 | **Write the program for array implementation of lists** | C311.1 | BTL 5 |
| 3 | **Write a C program for linked list implementation of list.** | C311.1 | BTL 5 |
| 4 | **Explain the operations of singly linked lists** | C311.1 | BTL 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | **Explain the operations of doubly linked lists** | C311.1 | BTL 2 |
| 6 | **Explain the operations of circularly linked lists** | C311.1 | BTL 2 |
| 7 | **How polynomial manipulations are performed with lists? Explain the operations** | C311.1 | BTL 1 |
| 8 | **Explain the steps involved in insertion and deletion into a singly and doubly linked list.** | C311.1 | BTL2 |

**UNIT II**

|  |  |  |
| --- | --- | --- |
| **Topics** | **Text / Reference book** | **Page No.** |
| Stack ADT | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. | 78 |
| Operations | 79 |
| Applications – Evaluating arithmetic expressions- Conversion of Infix to postfix expression | 87 |
| Queue ADT | 95 |
| Operations | 95 |
| Circular Queue | 95 |
| Priority Queue - deQueue | 98 |
| applications of queues | 100 |

|  |  |  |  |
| --- | --- | --- | --- |
| S.  No. | Question | Course Outcome | Blooms  Taxanomy Level |
| 1 | **Define Stack.**  A stack is an ordered list in which all insertions and deletions are made at one end, called  the top. It is an abstract data type and based on the principle of LIFO (Last In First Out). | C311.2 | BTL 1 |
| 2 | **What are the operations of the stack?**   1. CreateStack/ InitStack(Stack) – creates an empty stack 2. Push(Item) – pushes an item on the top of the stack 3. Pop(Item) – removes the top most element from the stack | C311.2 | BTL 1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | d. Top(Stack) – returns the first element from the stack  e. IsEmpty(Stack) – returns true if the stack is empty |  |  |
| 3 | **Write the routine to push a element into a stack.**  Push(Element X, Stack S)  {  if(IsFull(S)  {  Error(“Full Stack”);  }  else S→Array[++S→TopOfStack]=X;  } | C311.2 | BTL 5 |
| 4 | **How the operations performed on linked list implementation of stack?**   1. Push and pop operations at the head of the list. 2. New nodes should be inserted at the front of the list, so that they become the top of the stack. 3. Nodes are removed from the front(top) of the stack. | C311.2 | BTL 1 |
| 5 | **What are the applications of stack?**  The following are the applications of stacks   * Evaluating arithmetic expressions * Balancing the parenthesis * Towers of Hanoi * Function calls Tree traversal | C311.2 | BTL 1 |
| 6 | **What are the methods to implement stack in C?**  The methods to implement stacks are:   * Array based * Linked list based | C311.2 | BTL 1 |
| 7 | **How the stack is implemented by linked list?**  It involves dynamically allocating memory space at run time while performing stack  operations.  Since it consumes only that much amount of space is required for holding its data  elements , it prevents wastage of memory space. struct stack  {  int element;  struct stack \*next;  }\*top; | C311.2 | BTL 1 |
| 8 | **Write the routine to pop a element from a stack.**  int pop()  {  if(top==NULL) | C311.2 | BTL 5 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | {  printf(“\n Stack is empty.\n”); getch();  exit(1);  }  else  {  int temp;  temp=top→element; /\* retreiving the top element\*/ top=top→next; /\* Updating the stack pointer \*/ return temp; /\* returning the popped value \*/  }  } |  |  |
| 9 | **Define queue.**  It is a linear data structure that maintains a list of elements such that insertion happens at  rear end and deletion happens at front end. FIFO – First In First Out principle | C311.2 | BTL 1 |
| 10 | **What are the operations of a queue?**  The operations of a queue are   * isEmpty() * isFull() * insert() * delete() * display() | C311.2 | BTL 1 |
| 11 | **Write the routine to insert a element onto a queue.**  void insert(int element)  {  if(front==-1 )  {  front = rear = front +1; queue[front] = element; return;  }  if(rear==99)  {  printf(“Queue is full”); getch();  return;  }  rear = rear +1; queue[rear]=element;  } | C311.2 | BTL 5 |
| 12 | **What are the types of queue?**  The following are the types of queue: | C311.2 | BTL 1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | * Double ended queue * Circular queue * Priority queue |  |  |
| 13 | **Define double ended queue**   * It is a special type of queue that allows insertion and deletion of elements at both   Ends.   * It is also termed as DEQUE. | C311.2 | BTL 1 |
| 14 | **What are the methods to implement queue in C?**  The methods to implement queues are:   * Array based * Linked list based | C311.2 | BTL 1 |
| 15 | **How the queue is implemented by linked list?**   * It is based on the dynamic memory management techniques which allow allocation and   De-allocation of memory space at runtime.  **Insert operation**  It involves the following subtasks:   * 1. Reserving memory space of the size of a queue element in memory   2. Storing the added value at the new location   3. Linking the new element with existing queue   4. Updating the *rear* pointer   **Delete operation**  It involves the following subtasks:   1. Checking whether queue is empty 2. Retrieving the front most element of the queue 3. Updating the front pointer 4. Returning the retrieved value | C311.2 | BTL 1 |
| 16 | **Write the routine to delete a element from a queue**  int del()  {  int i;  if(front == NULL) /\*checking whether the queue is empty\*/  {  return(-9999);  }  else  {  i = front→element; | C311.2 | BTL 5 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | front = front→next; return i;  }  } |  |  |
| 17 | **What are the applications of queue?**  The following are the areas in which queues are applicable   1. Simulation 2. Batch processing in an operating systems 3. Multiprogramming platform systems 4. Queuing theory 5. Printer server routines 6. Scheduling algorithms like disk scheduling , CPU scheduling 7. I/O buffer requests | C311.2 | BTL 1 |
| 18 | **Define circular queue**  A Circular queue is a queue whose start and end locations are logically connected with  each other. That means the start location comes after the end location. | C311.2 | BTL 1 |
| 19 | **What are push and pop operations?**   * Push – adding an element to the top of stack * Pop – removing or deleting an element from the top of stack | C311.2 | BTL 1 |
| 20 | **What are enqueue and dequeue operations?**   * **Enqueue** - adding an element to the queue at the rear end   If the queue is not full, this function adds an element to the back of the queue, else it prints “**OverFlow**”.  void enqueue(int queue[], int element, int& rear, int arraySize)  {  if(rear == arraySize) // Queue is full printf(“OverFlow\n”);  else{  queue[rear] = element; // Add the element to the back rear++;  }  }   * **Dequeue** – removing or deleting an element from the queue at the front end | C311.2 | BTL 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | If the queue is not empty, this function removes the element from the front of the queue, else it prints “**UnderFlow**”.  void dequeue(int queue[], int& front, int rear) { if(front == rear) // Queue is empty  printf(“UnderFlow\n”); else {  queue[front] = 0; // Delete the front element front++;  }  } | | | |  |  |
| 21 | **Distinguish between stack and queue.** | | | | C311.2 | BTL4 |
|  | **STACK** | **QUEUE** |  |
| Insertion and deletion are made at one end. | Insertion at one end rear and deletion at other end front. |
| The element inserted last would be removed first. So LIFO structure. | The element inserted first would be removed first. So FIFO structure. |
| Full stack condition: If(top==Maxsize)  Physically and Logically full stack | Full stack condition:  If(rear = = Maxsize)  Logically full. Physically may or may not be full. |
| 22 | **Convert the infix (a+b)\*(c+d)/f into postfix & prefix expression**  Postfix : a b + c d + \* f /  Prefix : / \* + a b + c d f | | | | C311.2 | BTL5 |
| 23 | **Write postfix from of the expression –A+B-C+D?**  A-B+C-D+ | | | | C311.2 | BTL5 |
| 24 | **How do you test for an empty queue?**  To test for an empty queue, we have to check whether READ=HEAD where REAR is a pointer pointing to the last node in a queue and HEAD is a pointer that pointer to the dummy header. In the case of array implementation of queue, | | | | C311.2 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | the condition to be checked for an empty queue is READ<FRONT. |  |  |
| 25 | **What are the postfix and prefix forms of the expression?**  A+B\*(C-D)/(P-R)  Postfix form: ABCD-\*PR-/+ Prefix form: +A/\*B-CD-PR | C311.2 | BTL1 |
| 26 | **Explain the usage of stack in recursive algorithm implementation?**  In recursive algorithms, stack data structures is used to store the return address when a recursive call is encountered and also to store the values of all the parameters essential to the current state of the procedure. | C311.2 | BTL5 |
| 27 | **Define priority queue with diagram and give the operations.**  Priority queue is a data structure that allows at least the following two operations.   1. Insert-inserts an element at the end of the list called the rear. 2. DeleteMin-Finds, returns and removes the minimum element in the priority Queue.     Operations: Insert, DeleteMin | C311.2 | BTL1 |
| 28 | **Give the applications of priority queues.**  There are three applications of priority queues   1. External sorting. 2. Greedy algorithm implementation. 3. Discrete even simulation. 4. Operating systems. | C311.2 | BTL3 |
| 29 | **How do you test for an empty stack?**  To check if the stack is empty, we only need to check whether top and bottom are the same number.  bool stack\_empty(stack S) //@requires is\_stack(S);  { return S->top == S->bottom; } | C311.2 | BTL1 |
| 30 | **What are the features of stacks?**   * Dynamic data structures * Do not have a fixed size * Do not consume a fixed amount of memory * Size of stack changes with   each push() and pop() operation. | C311.2 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Each push() and pop() operation increases and decreases the size of the stack by 1, respectively. |  |  |
| 31 | **Write a routine for IsEmpty condition of queue.**  If a queue is empty, this function returns 'true', else it returns 'false'.  bool isEmpty(int front, int rear) { return (front == rear);  } | C311.2 | BTL5 |
| **32** | **How do you test for an empty stack?**  **The condition for testing an empty stack is top =-1, where top is the pointer pointing to the topmost element of the stack, in the array implementation of stack. In linked list implementation of stack the condition for an empty stack is the header node link**  **field is NULL.** | C311.2 |  |
| 33 | **Define a suffix expression**.  The notation used to write the operator at the end of the operands is called suffix notation. Suffix notation format : operand operand operator Example: ab+, where a & b are operands and ‘+’ is addition  operator | C311.2 |  |
| 34 | **What do you meant by fully parenthesized expression? Give example.** A pair of parentheses has the same parenthetical level as  that of the operator to which it corresponds. Such an expression is called fully parenthesized expression. Ex: (a+((b\*c) + (d \* e)) | C311.2 |  |
| 35 | **Write the postfix form for the expression -A+B-C+D?**  A-B+C-D+ | C311.2 |  |
| 36 | **What are the postfix and prefix forms of the expression?**  A+B\*(C-D)/(P-R)  Postfix form: ABCD-\*PR-/+ Prefix form: +A/\*B-CD-PR | C311.2 |  |
| 37 | **Explain the usage of stack in recursive algorithm implementation**?  In recursive algorithms, stack data structures is used to store the return address when a recursive call is encountered and also to store the values of all the parameters essential to the current state of the  function. | C311.2 |  |
| 38 | **Write down the function to insert an element into a queue, in which the queue is implemented as an array**  Q – Queue X – element to added to the queue Q IsFull(Q) – Checks and true if Queue Q is full Q->Size - Number of elements in the queue Q Q->Rear – Points to last element of the queue Q Q->Array – array used to store queue elements void enqueue (int X, Queue Q) { if(IsFull(Q)) Error (“Full queue”); else { Q->Size++; Q->Rear = Q-  >Rear+1; Q->Array[ Q->Rear ]=X; } } | C311.2 |  |
| **PART-B** | | | |
| 1 | **Explain Stack ADT and its operations** | C311.2 | BTL5 |
| 2 | **Explain array based implementation of stacks** | C311.2 | BTL5 |
| 3 | **Explain linked list implementation of stacks** | C311.2 | BTL5 |

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | **Explain the applications of Stacks** | C311.2 | BTL5 |
| 5 | **Explain how to evaluate arithmetic expressions using stacks** | C311.2 | BTL5 |
| 6 | **Explain queue ADT** | C311.2 | BTL2 |
| 7 | **Explain array based implementation of queues** | C311.2 | BTL2 |
| 8 | **Explain linked list implementation of queues** | C311.2 | BTL2 |
| 9 | **Explain the applications of queues** | C311.2 | BTL5 |
| 10 | **Explain circular queue and its implementation** | C311.2 | BTL2 |
| 11 | **Explain double ended queue and its operations** | C311.2 | BTL2 |
| 12 | **Explain priority queue and its operations** | C311.2 | BTL5 |

# UNIT III

|  |  |  |
| --- | --- | --- |
| **Topics** | **Text / Reference book** | **Page No.** |
| Tree ADT– tree traversals | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. | 106 |
| Binary Tree ADT | 111 |
| expression trees – applications of trees | 113 |
| binary search tree ADT–Threaded Binary Trees | 116 |
| AVL Trees | 126 |
| B-Tree | 149 |
| B+ Tree | 154 |
| Heap - Applications of heap. | 208 |

|  |  |  |  |
| --- | --- | --- | --- |
| S.  No. | Question | Course Outcome | Blooms Taxanomy  Level |
| 1 | **Define non-linear data structure**  Data structure which is capable of expressing more complex relationship than that of physical adjacency is called non-linear data structure. | C311.3 | BTL1 |
| 2 | **Define tree?**  A tree is a data structure, which represents hierarchical relationship between individual data items. | C311.3 | BTL1 |
| 3 | **Define leaf?**  In a directed tree any node which has out degree o is called a terminal node or a leaf. | C311.3 | BTL1 |
| 4 | **Explain the representations of priority queue.**  Using Heap structure, Using Linked List | C311.3 | BTL2 |
| 5 | **List out the steps involved in deleting a node from a binary search tree.**   1. t has no right hand child node t->r == z 2. t has a right hand child but its right hand child node has no left sub tree   t->r->l == z  3.t has a right hand child node and the right hand child node has a left hand child node t->r-  >l != z | C311.3 | BTL1 |
| 6 | **Convert the infix expression (A-B/C)\*(D/E-F) into a postfix.**  Postfix: ABC/-DE/F-\* | C311.3 | BTL2 |
| 7 | **What are the steps to convert a general tree into binary tree?**   * use the root of the general tree as the root of the binary tree * determine the first child of the root. This is the leftmost node in the general tree at the next   level   * insert this node. The child reference of the parent node refers to this node * continue finding the first child of each parent node and insert it below the parent node with the   child reference of the parent to this node.   * when no more first children exist in the path just used, move back to the parent of the last node   entered and repeat the above process. In other words, determine the first sibling of the last  node entered.   * complete the tree for all nodes. In order to locate where the node fits you must search for the | C311.3 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | first child at that level and then follow the sibling references to a nil where the next sibling can  be inserted. The children of any sibling node can be inserted by locating the parent and then  inserting the first child. Then the above process is repeated. |  |  |
| 8 | **What is meant by directed tree?**  Directed tree is an acyclic diagraph which has one node called its root with in degree o while all other nodes have in degree I. | C311.3 | BTL1 |
| 9 | **What is a ordered tree?**  In a directed tree if the ordering of the nodes at each level is prescribed then such a tree is called ordered tree. | C311.3 | BTL1 |
| 10 | **What are the applications of binary tree?**   1. Binary tree is used in data processing. 2. File index schemes 3. Hierarchical database management system | C311.3 | BTL1 |
| 11 | **What is meant by traversing?**  Traversing a tree means processing it in such a way, that each node is  visited only once. | C311.3 | BTL1 |
| 12 | **What are the different types of traversing?**  The different types of traversing are   1. Pre-order traversal-yields prefix form of expression. 2. In-order traversal-yields infix form of expression. 3. Post-order traversal-yields postfix form of expression. | C311.3 | BTL1 |
| 13 | **What are the two methods of binary tree implementation?**  Two methods to implement a binary tree are   1. Linear representation. 2. Linked representation | C311.3 | BTL1 |
| 14 | **What is a balance factor in AVL trees?**  Balance factor of a node is defined to be the difference between the height of the node's left subtree and the height of the node's right subtree. | C311.3 | BTL1 |
| 15 | **What is meant by pivot node?**  The node to be inserted travel down the appropriate branch track along the way of the deepest level node on the branch that has a balance factor of +1 or -1 is called pivot node. | C311.3 | BTL1 |
| 16 | **What is the length of the path in a tree?** | C311.3 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | The length of the path is the number of edges on the path. In a tree there is exactly one path form the root to each node. |  |  |
| 17 | **Define expression trees?**  The leaves of an expression tree are operands such as constants or variable names and the other nodes contain operators. | C311.3 | BTL1 |
| 18 | **What is a threaded binary tree?**  A threaded [binary tree](http://en.wikipedia.org/wiki/Binary_tree) may be defined as follows: "A binary tree is *threaded* by making all right child pointers that would normally be null point to the inorder successor of the node, and all left child pointers that would normally be null point to the inorder predecessor of the node | C311.3 | BTL1 |
| 19 | **What is meant by binary search tree?**  Binary Search tree is a binary tree in which each internal node *x* stores an element such that the element stored in the left sub tree of *x* are less than or equal to *x* and elements stored in the right sub tree of *x* are greater than or equal to *x*. | C311.3 | BTL2 |
| 20 | **Write the advantages of threaded binary tree.**  The difference between a binary tree and the threaded binary tree is that in the binary trees the nodes are null if there is no child associated with it and so there is no way to traverse back. But in a threaded binary tree we have threads associated with the nodes i.e they either are linked to the predecessor or successor in the in order traversal of the nodes.  This helps us to traverse further or backward in the in order traversal fashion.  There can be two types of threaded binary tree :-   1. Single Threaded: - i.e. nodes are threaded either towards its in order predecessor or successor. 2. Double threaded: - i.e. nodes are threaded towards both the in order predecessor and successor. | C311.3 | BTL5 |
| 21 | **What is the various representation of a binary tree?**  Tree Representation Array representation Linked list representation | C311.3 | BTL1 |
| 22 | **List the application of tree.**  (i) Electrical Circuit  ii) Folder structure   1. Binary tree is used in data processing. 2. File index schemes 3. Hierarchical database management system | C311.3 | BTL1 |
| 23 | **Define binary tree and give the binary tree node structure.** | C311.3 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 24 | **What are the different ways of representing a Binary Tree?**   * Linear Representation using Arrays. * Linked Representation using Pointers. | C311.3 | BTL1 |
| 25 | **Give the pre & postfix form of the expression (a + ((b\*(c- e))/f).** | C311.3 | BTL2 |
| 26 | **Define a heap. How can it be used to represent a priority queue?**  A priority queue is a different kind of queue, in which the next element to be removed is defined by (possibly) some other criterion. The most common way to implement a priority queue is to use a different kind of binary tree, called a heap. A heap avoids the long paths that can arise with binary search trees. | C311.3 | BTL1 |
| 27 | **What is binary heap?**  It is a complete binary tree of height h has between 2h and 2h+1 - 1 node. The value of the root node is higher than their child nodes | C311.3 | BTL1 |
| 28 | **Define Strictly binary tree?**  If every nonleaf node in a binary tree has nonempty left and right subtrees ,the tree is termed  as a strictly binary tree. | C311.3 | BTL1 |
| 29 | **Define complete binary tree?**  A complete binary tree of depth d is the strictly binary tree all of whose are at level d. | C311.3 | BTL1 |
| 30 | **What is an almost complete binary tree?**  A binary tree of depth d is an almost complete binary tree if :  \_ Each leaf in the tree is either at level d or at level d-1  \_ For any node nd in the tree with a right descendant at level d,all the left descendants of  nd that are leaves are at level d. | C311.3 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 31 | **Define AVL Tree.**  A AVL tree is a binary search tree except that for every node in the tree,the height of the  left and right subtrees can differ by atmost 1. | C311.3 | BTL1 |
| 32 | **What is the length of the path in a tree?**  The length of the path is the number of edges on the path. In a tree there is exactly one path form the root to each node. | C311.3 | BTL1 |
| 33 | **Define sibling?**  Nodes with the same parent are called siblings. The nodes with common parents are called siblings. | C311.3 | BTL1 |
| 34 | **What are the two methods of binary tree implementation?**  Two methods to implement a binary tree are, a. Linear representation.  b. Linked representation | C311.3 | BTL1 |
| 35 | **List out few of the Application of tree data-structure?**  Ø The manipulation of Arithmetic expression Ø Used for Searching Operation  Ø Used to implement the file system of several popular operating systems  Ø Symbol Table construction Ø Syntax analysis | C311.3 | BTL1 |
| 36 | **Define tree traversal and mention the type of traversals?**  Visiting of each and every node in the tree exactly is called as tree  traversal. Three types of tree traversal 1. Inorder traversal 2. Preoder traversal 3. Postorder traversal. | C311.3 | BTL1 |
| 37 | **What are the types of threaded binary tree?**   1. Right-in threaded binary tree 2. . Left-in threaded binary tree 3. . Fully-in threaded binary tree | C311.3 | BTL1 |
| 38 | **List out the steps involved in deleting a node from a binary search tree**.   * Deleting a node is a leaf node (ie) No children * Deleting a node with one child. * Deleting a node with two Childs | C311.3 | BTL1 |
| 39 | **What is B Tree**  A B-tree is a tree data structure that keeps data sorted and allows searches, insertions, and deletions in logarithmic amortized time. Unlike self-balancing binary search trees, it is optimized for systems that read and write large blocks of data. It is most commonly used in database and file systems.  Important properties of a B-tree: • B-tree nodes have many more than two children. • A B-tree node may contain more than just a single element. | C311.3 | BTL1 |
| 40 | **What is binomial heaps?**  A binomial heap is a collection of binomial trees that satisfies the  following binomial-heap properties: 1. No two binomial trees in the collection have the same size. 2. Each node in each tree has a key. 3. | C311.3 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Each binomial tree in the collection is heap-ordered in the sense that each non-root has a key strictly less than the key of its parent |  |  |
| 41 | **Define complete binary tree**.  If all its levels, possible except the last, have maximum number of nodes and if all the nodes in the last level appear as far left as possible. | C311.3 | BTL1 |
| **PART-B** | | | |
| 1 | **Define Tree. Explain the tree traversals with algorithms and examples.** | C311.3 | BTL5 |
| 2 | **Construct an expression tree for the expression (a + b**  **\* c) +((d \* e + 1) \* g). Give the outputs when you apply preorder, inorder and postorder traversals.** | C311.3 | BTL5 |
| 3 | **Explain binary search tree ADT in detail.** | C311.3 | BTL5 |
| 4 | **Explain AVL tree ADT in detail.** | C311.3 | BTL5 |
| 5 | **Explain b tree and B+ tree ADT in detail.** | C311.3 | BTL5 |
| 6 | **Explain Heap tree ADT in detail.** | C311.3 | BTL5 |
| 7 | **Explain threaded binary tree ADT in detail.** | C311.3 | BTL2 |

**UNIT IV**

|  |  |  |
| --- | --- | --- |
| **Topics** | **Text / Reference book** | **Page No.** |
| Definition – Representation of Graph | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. | 229 |
| Types of graph | 300 |
| Breadth-first traversal | 335 |
| Depth-first traversal | 342 |
| Topological Sort | 302 |
| Bi-connectivity | 338 |
| Cut vertex | 342 |
| Euler circuits | 342 |
| Applications of graphs | 348 |

|  |  |  |  |
| --- | --- | --- | --- |
| S. N  o. | Question | Course Outco me | Bloo ms Taxan  omy Level |
| 1 | **Define Graph?**  A graph G consist of a nonempty set V which is a set of nodes of the graph, a set E which is the set of edges of the graph, and a mapping from the set for edge E to a set of pairs of elements of V. It can also be represented as G= (V, E). | C311.4 | BTL1 |
| 2 | **Explain the topological sort.**  It is an Ordering of vertices in a directed acyclic graph such that if there is a path from vi to vj, then vj appears after vi in the ordering. | C311.4 | BTL1 |
| 3 | **Define NP**  NP is the class of decision problems for which a given proposed solution for a given input can be checked quickly to see if it is really a solution. | C311.4 | BTL1 |
| 4 | **Define biconnected graph.**  A connected undirected graph is biconnected if there are no vertices whose removal disconnects the rest of the graph. | C311.4 | BTL1 |
| 5 | **Define shortest path problem?**  For a given graph G=(V, E), with weights assigned to the edges of G, we have to find the shortest path (path length is defined as sum of the weights of the edges) from any given source vertex to all the remaining vertices of G. | C311.4 | BTL1 |
| 6 | **Mention any two decision problems which are NP-Complete.**  NP is the class of decision problems for which a given proposed solution for a given input can be checked quickly to see if it is really a solution | C311.4 | BTL2 |
| 7 | **Define adjacent nodes?**  Any two nodes which are connected by an edge in a graph are called adjacent nodes. For E is associated with a pair of nodesexample, if and edge x (u,v) where u, v V, then we say that the edge x connects the nodes u and v.  | C311.4 | BTL1 |
| 8 | **What is a directed graph?**  A graph in which every edge is directed is called a directed graph. | C311.4 | BTL1 |
| 9 | **What is a undirected graph?**  A graph in which every edge is undirected is called a directed graph. | C311.4 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | **What is a loop?**  An edge of a graph which connects to itself is called a loop or  sling. | C311.4 | BTL1 |
| 11 | **What is a simple graph?**  A simple graph is a graph, which has not more than one edge between a pair of nodes than such a graph is called a simple graph. | C311.4 | BTL1 |
| 12 | **What is a weighted graph?**  A graph in which weights are assigned to every edge is called a weighted graph. | C311.4 | BTL1 |
| 13 | **Define out degree of a graph?**  In a directed graph, for any node v, the number of edges which have v as their initial node is called the out degree of the node v. | C311.4 | BTL1 |
| 14 | **Define indegree of a graph?**  In a directed graph, for any node v, the number of edges which have v as their terminal node is called the indegree of the node v. | C311.4 | BTL1 |
| 15 | **Define path in a graph?**  The path in a graph is the route taken to reach terminal node from a starting node. | C311.4 | BTL1 |
| 16 | **What is a simple path?**  A path in a diagram in which the edges are distinct is called a simple path. It is also called as edge simple. | C311.4 | BTL1 |
| 17 | **What is a cycle or a circuit?**  A path which originates and ends in the same node is called a cycle or circuit. | C311.4 | BTL1 |
| 18 | **What is an acyclic graph?**  A simple diagram which does not have any cycles is called an acyclic graph. | C311.4 | BTL1 |
| 19 | **What is meant by strongly connected in a graph?**  An undirected graph is connected, if there is a path from every vertex to every other vertex. A directed graph with this property is called strongly  connected. | C311.4 | BTL1 |
| 20 | **When is a graph said to be weakly connected?**  When a directed graph is not strongly connected but the underlying graph is connected, then the graph is said to be weakly connected. | C311.4 | BTL1 |
| 21 | **Name the different ways of representing a graph?**   1. Adjacency matrix 2. Adjacency list | C311.4 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 22 | **What is an undirected acyclic graph?**  When every edge in an acyclic graph is undirected, it is called an undirected acyclic graph. It is also called as undirected forest. | C311.4 | BTL1 |
| 23 | **What are the two traversal strategies used in traversing a graph?**   1. Breadth first search 2. Depth first search | C311.4 | BTL1 |
| 24 | **What is a minimum spanning tree?**  A minimum spanning tree of an undirected graph G is a tree formed from graph edges that connects all the vertices of G at the lowest total cost. | C311.4 | BTL1 |
| 25 | **Define topological sort?**  A topological sort is an ordering of vertices in a directed acyclic graph, such that if there is a path from vi to vj appears after vi in the ordering. | C311.4 | BTL1 |
| 26 | **What is the use of Kruskal’s algorithm and who discovered it?** Kruskal’s algorithm is one of the greedy techniques to solve the minimum spanning tree problem. It was discovered by Joseph Kruskal when he was a second-year graduate student. | C311.4 | BTL1 |
| 27 | **What is the use of Dijksra’s algorithm?**  Dijkstra’s algorithm is used to solve the single-source shortest- paths problem: for a given vertex called the source in a weighted connected graph, find the shortest path to all its other vertices. The single-source shortest-paths problem asks for a family of paths, each leading from the source to a different vertex in the graph, though some paths may have edges in common. | C311.4 | BTL1 |
| 28 | **Prove that the maximum number of edges that a graph with n Vertices is n\*(n-1)/2.**  Choose a vertex and draw edges from this vertex to the remaining n-1 vertices. Then, from these n-1 vertices, choose a vertex and draw edges to the rest of the n-2 Vertices. Continue this process till it ends with a single Vertex. Hence, the total number of edges added in graph is  (n-1)+(n-2)+(n-3)+…+1 =n\*(n-1)/2. | C311.4 | BTL5 |
| 29 | **Define minimum cost spanning tree?**  A spanning tree of a connected graph G, is a tree consisting of edges and all the vertices of G. In minimum spanning tree T, for a given graph G, the total weights of the edges of the spanning tree must be minimum compared to all other spanning trees generated from G. -Prim’s and Kruskal is the algorithm for finding Minimum Cost Spanning Tree. | C311.4 | BTL1 |
| 30 | **Define Adjacency in graph.**  Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on. | C311.4 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 31 | **Define Basic Operations of Graph.**  Following are basic primary operations of a Graph   * **Add Vertex** − Adds a vertex to the graph. * **Add Edge** − Adds an edge between the two vertices of the graph. * **Display Vertex** − Displays a vertex of the graph. | C311.4 | BTL1 |
| 32 | **What is Levels in graph?**  Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on. | C311.4 | BTL1 |
| 33 | **What is visiting and traversing in graph.**   * Visiting refers to checking the value of a node when control is on the node. * Traversing means passing through nodes in a specific order. | C311.4 | BTL1 |
| 34 | **Write the definition of weighted graph?**  A graph in which weights are assigned to every edge is called a weighted graph. | C311.4 | BTL1 |
| 35 | **Define adjacency matrix?**  Adjacency matrix is an n x n matrix A whose elements aij are given by aij = 1 if (vi, vj) Exists =0 otherwise | C311.4 | BTL1 |
| 36 | **Define adjacent nodes?**  Any two nodes, which are connected by an edge in a graph, are called adjacent nodes. For example, if an edge x E is associated with a pair of nodes  (u,v) where u, v V, then we say that the edge x connects the nodes u and v. | C311.4 | BTL1 |
| 37 | **What is topological sort**?  It is an ordering of the vertices in a directed acyclic graph, such that: If there is a path from u to v, then v appears after u in the ordering. | C311.4 | BTL1 |
| 38 | **Write BFS algorithm**   1. Initialize the first node’s dist number and place in queue 2. Repeat until all nodes have been examined 3. Remove current node to be examined from queue 4. Find all unlabeled nodes adjacent to current node 5. If this is an unvisited node label it and add it to the queue 6. Finished. | C311.4 | BTL1 |
| 39 | **What are the two traversal strategies used in traversing a graph?**   1. Breadth first search 2. Depth first search | C311.4 | BTL1 |
| 40 | **What is articulation point**  Articulation Points (or Cut Vertices) in a Graph A vertex in an undirected connected graph is an articulation point (or cut vertex) if removing it (and edges through it) disconnects the graph. Articulation points represent vulnerabilities in a connected network – single points whose failure would split the network into 2 or more disconnected components. They are useful for designing reliable  network | C311.4 | BTL1 |
| PART-B | | | |
| 1 | **Explain the various representation of graph with example in detail?** | C311.4 | BTL2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | **Define topological sort? Explain with an example?** | C311.4 | BTL5 |
| 3 | **Explain Dijkstra's algorithm with an example?** | C311.4 | BTL5 |
| 4 | **Explain Prim's algorithm with an example?** | C311.4 | BTL5 |
| 5 | **Explain Krushal's algorithm with an example?** | C311.4 | BTL2 |
| 6 | **Write and explain the prim’s algorithm and depth first search algorithm.** | C311.4 | BTL5 |
| 7 | **For the graph given below, construct Prims algorithm**  **2**  **4** 1 **2 1 7**  **8 4** 5  3 **5 1** 4 **6**  **1 2**  6 7 | C311.4 | BTL5 |
| 8 | **Explain the breadth first search algorithm** | C311.4 | BTL5 |
| 9 | **Write the algorithm to compute lengths of shortest path** | C311.4 | BTL5 |
| 10 | **Explain the depth first search algorithm.** | C311.4 | BTL2 |

# UNIT V

|  |  |  |
| --- | --- | --- |
| **Topics** | **Text / Reference book** | **Page No.** |
| Searching- Linear Search | 1. Mark Allen Weiss, “Data Structures and Algorithm Analysis in C”, 2nd Edition, Pearson Education,1997. | 235 |
| Binary Search | 236 |
| Sorting – Bubble sort | 237 |
| Selection sort – Insertion sort | 235 |
| Shell sort – Radix sort | 238 |
| Hashing- Hash Functions | 165 |
| Separate Chaining | 168 |

|  |  |  |
| --- | --- | --- |
| Open Addressing |  | 173 |
| Rehashing – Extendible Hashing | 181 |

|  |  |  |  |
| --- | --- | --- | --- |
| S.  No. | Question | Course Outcome | Blooms Taxanom  y Level |
| 1 | **Define sorting**  Sorting arranges the numerical and alphabetical data present in a list in a specific order or sequence. There are a number of sorting techniques available. The algorithms can be chosen based on the following factors   * Size of the data structure * Algorithm efficiency   Programmer’s knowledge of the technique | C311.5 | BTL1 |
| 2 | **Mention the types of sorting**   * Internal sorting * External sorting | C311.5 | BTL2 |
| 3 | **What do you mean by internal and external sorting?**  An internal sort is any data sorting process that takes place entirely within the main memory of a computer. This is possible whenever the data to be sorted is small enough to all be held in the main memory.  External sorting is a term for a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead they must reside in the slower external memory (usually a hard drive). | C311.5 | BTL1 |
| 4 | **How the insertion sort is done with the array?**  It sorts a list of elements by inserting each successive element in the previously sorted  Sub list.  Consider an array to be sorted A[1],A[2],….A[n]   1. Pass 1: A[2] is compared with A[1] and placed them in sorted order. 2. Pass 2: A[3] is compared with both A[1] and A[2] and inserted at an appropriate   place. This makes A[1], A[2],A[3] as a sorted sub array.   1. Pass n-1: A[n] is compared with each element in the sub array A [1], A [2] …A [n-1] and inserted at an appropriate position. | C311.5 | BTL1 |
| 5 | **Define hashing.**  Hash function takes an identifier and computes the address of that identifier in the hash table using some function | C311.5 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 6 | **What is the need for hashing?**  Hashing is used to perform insertions, deletions and find in constant average time. | C311.5 | BTL1 |
| 7 | **Define hash function?**  Hash function takes an identifier and computes the address of that identifier in the hash table using some function. | C311.5 | BTL1 |
| 8 | **List out the different types of hashing functions?**  The different types of hashing functions are,   1. The division method 2. The mind square method 3. The folding method 4. Multiplicative hashing 5. Digit analysis | C311.5 | BTL1 |
| 9 | **What are the problems in hashing?**   1. Collision 2. Overflow | C311.5 | BTL1 |
| 10 | **What are the problems in hashing?**  When two keys compute in to the same location or address in the hash table  through any of the hashing function then it is termed collision. | C311.5 | BTL1 |
| 11 | **what is insertion sort? How many passes are required for the elements to be sorted ?**  one of the simplest sorting algorithms is the insertion sort. Insertion sort consist of N-1 passes . For pass P=1 through N-1 , insertion sort ensures that the elements in positions 0 through P-1 are in sorted order .It makes use of the fact that elements in position 0 through P- 1 are already known to be in sorted order . | C311.5 | BTL1 |
| 12 | **Write the function in C for insertion sort ?**  void insertionsort(elementtype A[ ] , int N)  {  int j, p; elementtype tmp;  for(p=1 ; p <N ;p++ )  {  tmp = a[ p] ;  for ( j=p ; j>0 && a [ j -1 ] >tmp ;j--) a [ j ]=a [j-1 ] ;  a [ j ] = tmp ;  }} | C311.5 | BTL5 |

|  |  |  |  |
| --- | --- | --- | --- |
| 13 | **Who invented shellsort ? define it ?** Shellsort was invented by Donald Shell . It works by comparing element that are distant . The distance between the comparisons decreases as the algorithm runs until the last phase in which adjacent elements are compared . Hence it is referred as diminishing increment  sort. | C311.5 | BTL1 |
| 14 | **write the function in c for shellsort?**  Void Shellsort(Elementtype A[ ],int N)  {  int i , j , increment ; elementtype tmp ;  for(elementtype=N / 2;increment > 0;increment / = 2) For( i= increment ; i <N ; i ++)  {  tmp=A[ ];  for( j=I; j>=increment; j - =increment) if(tmp< A[ ]=A[j – increment];  A[ j ]=A[ j – increment]; Else  Break;  A[ j ]=tmp;  }} | C311.5 | BTL5 |
| 15 | **Differentiate between merge sort and quick sort? Mergesort quick sort**   1. Divide and conquer strategy Divide and conquer strategy 2. Partition by position Partition by value | C311.5 | BTL4 |
| 16 | **Mention some methods for choosing the pivot element in quick sort?**   1. Choosing first element 2. Generate random number 3. Median of three | C311.5 | BTL2 |
| 17 | **What are the three cases that arise during the left to right scan in quick sort?**   1. I and j cross each other 2. I and j do not cross each other 3. I and j points the same position | C311.5 | BTL1 |
| 18 | **What is the need of external sorting?**  External sorting is required where the input is too large to fit into memory. So external sorting Is necessary where the program is too large | C311.5 | BTL1 |
| 19 | **What is sorting?**  Sorting is the process of arranging the given items in a logical order. Sorting is an example where the analysis can be precisely performed. | C311.5 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
| 20 | **What is mergesort?**  The mergesort algorithm is a classic divide conquer strategy. The problem is divided into two arrays and merged into single array | C311.5 | BTL1 |
| 21 | **Compare the various hashing techniques.**  Technique Load Factor  Separate chaining - close to 1  Open Addressing - should not exceed 0.5  Rehashing - reasonable load factor | C311.5 | BTL2 |
| 22 | **Define collision in hashing.**  When two different keys or identifiers compute into the same location or address in the hash table through any of the hashing functions, then it is termed Collision. | C311.5 | BTL1 |
| 23 | **Define Double Hashing.**  Double Hashing is a collision-resolution technique used in open addressing category. In double hashing, we apply a second hash function to x and probe at a distance of hash2 (x),  2hash2 (x) , and so on. | C311.5 | BTL1 |
| 24 | **What are applications of hashing?**  The applications of hashing are,   * Compliers use hash table to keep track of declared variables on source code. * Hash table is useful for any graph theory problem, where the nodes have real names instead of numbers. * Hash tables are used in programs that play games. * Online spelling checkers use hashing. | C311.5 | BTL1 |
| 25 | **What does internal sorting mean?**  Internal sorting is a process of sorting the data in the main memory | C311.5 | BTL1 |
| 26 | **What are the various factors to be considered in deciding a sorting algorithm?**  Factors to be considered in deciding a sorting algorithm are,   1. Programming time 2. Executing time for program 3. Memory or auxiliary space needed for the programs environment. | C311.5 | BTL1 |
| 27 | **How does the bubble sort get its name?**  The bubble sort derives its name from the fact that the smallest data item bubbles up to the top of the sorted array. | C311.5 | BTL1 |
| 28 | **What is the main idea behind the selection sort?** | C311.5 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | The main idea behind the selection sort is to find the smallest entry among in a(j),a(j+1),… a(n) and then interchange it with a(j).  This process is then repeated for each value of j. |  |  |
| 29 | **Is the heap sort always better than the quick sort?**  No, the heap sort does not perform better than the quick sort.  Only when array is nearly sorted to begin with the heap sort algorithm gains an advantage. In such a case, the quick deteriorates to its worst performance of O (n2). | C311.5 | BTL4 |
| 30 | **Name some of the external sorting methods.**  Some of the external sorting methods are,   1. Polyphase sorting 2. Oscillation sorting 3. Merge sorting | C311.5 | BTL2 |
| 31 | **Define radix sort**  Radix Sort is a clever and intuitive little sorting algorithm. **Radix sort** is a on comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and | C311.5 | BTL1 |
| 32 | **Define searching**  Searching refers to determining whether an element is present in a given list of elements  or not. If the element is present, the search is considered as successful, otherwise it is considered as an unsuccessful search. The choice of a searching technique is based on the following factors   1. Order of elements in the list i.e., random or sorted 2. Size of the list | C311.5 | BTL1 |
| 33 | **Mention the types of searching**  The types are   * Linear search * Binary search | C311.5 | BTL2 |
| 34 | **What is meant by linear search?**  **Linear search** or **sequential search** is a method for finding a particular value in a list  that consists of checking every one of its elements, one at a time and in sequence, until  the desired one is found. | C311.5 | BTL1 |
| 35 | **What is binary search?**  For binary search, the array should be arranged in ascending or descending order.  In each step, the algorithm compares the search key value with the middle element of the | C311.5 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | array. If the key match, then a matching element has been found and its index, or  Position, is returned.  Otherwise, if the search key is less than the middle element, then the algorithm repeats its  action on the sub-array to the left of the middle element or, if the search key is greater, on  the sub-array to the right. |  |  |
| 36 | **What are the collision resolution methods?**  The following are the collision resolution methods   * Separate chaining * Open addressing * Multiple hashing | C311.5 | BTL1 |
| 37 | **Define separate chaining**  It is an open hashing technique. A pointer field is added to each record location, when an  overflow occurs; this pointer is set to point to overflow blocks making a linked list. In this method, the table can never overflow, since the linked lists are only extended upon the arrival of new keys. | C311.5 | BTL1 |
| 38 | **What is open addressing?**  Open addressing is also called closed hashing, which is an alternative to resolve the  Collisions with linked lists. In this hashing system, if a collision occurs, alternative cells  are tired until an empty cell is found.  There are three strategies in open addressing:   * Linear probing * Quadratic probing * Double hashing | C311.5 | BTL1 |
| 39 | **What is Rehashing?**  If the table is close to full, the search time grows and may become equal to the table size.  When the load factor exceeds a certain value (e.g. greater than 0.5) we do  Rehashing: Build a second table twice as large as the original and rehash there all the keys of the original table.  Rehashing is expensive operation, with running time O(N) However, once done, the new hash table will have good performance. | C311.5 | BTL1 |
| 40 | **What is Extendible Hashing?**  Used when the amount of data is too large to fit in main memory and external storage is used.  **N** records in total to store, **M** records in one disk block | C311.5 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **The problem**: in ordinary hashing several disk blocks may be examined to find an element - a time consuming process.  **Extendible hashing**: no more than two blocks are examined. |  |  |
| 41 | **List the different sorting algorithms.**   * Bubble sort * Selection sort * Insertion sort * Shell sort * Quick sort * Radix sort * Heap sort * Merge sort | C311.5 | BTL1 |
| 42 | **Why bubble sort is called so?**  The bubble sort gets its name because as array elements are sorted they gradually “bubble” to their proper positions, like bubbles rising in a glass of  soda. | C311.5 | BTL1 |
| 43 | **State the logic of bubble sort algorithm**.  The bubble sort repeatedly compares adjacent elements of an array. The first and second elements are compared and swapped if out of order. Then the second and third elements are compared and swapped if out of order. This sorting process continues until the last two elements of the array are  compared and swapped if out of order. | C311.5 | BTL1 |
| 44 | **What number is always sorted to the top of the list by each pass of the Bubble sort algorithm?**  Each pass through the list places the next largest value in its proper place. In essence, each item “bubbles” up to the location where it belongs. | C311.5 | BTL1 |
| 45 | **When does the Bubble Sort Algorithm stop?**  The bubble sort stops when it examines the entire array and finds that no "swaps" are needed. The bubble sort keeps track of the occurring swaps by  the use of a flag. | C311.5 | BTL1 |
| 46 | **What is the output of selection sort after the 2nd iteration given the following sequence?** 16 3 46 9 28 14  Ans: 3 9 46 16 28 14 | C311.5 | BTL1 |
| 47 | **How does insertion sort algorithm work?**  In every iteration an element is compared with all the elements before it. While comparing if it is found that the element can be inserted at a suitable position, then space is created for it by shifting the other elements one position up and inserts the desired element at the suitable position. This procedure is repeated for all the elements in the list until we get the sorted elements. | C311.5 | BTL1 |
| 48 | **What operation does the insertion sort use to move numbers from the unsorted section to the sorted section of the list**?  The Insertion Sort uses the swap operation since it is ordering numbers within a single list. | C311.5 | BTL1 |
| 49 | **How many key comparisons and assignments an insertion sort makes in its worst case?**  The worst case performance in insertion sort occurs when the elements of the input array are in descending order. In that case, the first pass requires one comparison, the second pass requires two comparisons, third pass three  comparisons,….kth pass requires (k-1), and finally the last pass requires (n- | C311.5 | BTL1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1) comparisons. Therefore, total numbers of comparisons are: f(n) = 1+2+3+………+(n-k)+…..+(n-2)+(n-1) = n(n-1)/2 = O(n2) |  |  |
| 50 | **Which sorting algorithm is best if the list is already sorted? Why?**  Insertion sort as there is no movement of data if the list is already sorted and complexity is of the order O(N). | C311.5 | BTL1 |
| PART -B | | | |
| 1 | **Explain the sorting algorithms** | C311.5 | BTL2 |
| 2 | **Explain the searching algorithms** | C311.5 | BTL5 |
| 3 | **Explain hashing** | C311.5 | BTL5 |
| 4 | **Explain open addressing** | C311.5 | BTL5 |
| 5 | **Write a C program to sort the elements using bubble sort, insertion sort and radix sort.** | C311.5 | BTL5 |
| 6 | **Write a C program to perform searching operations using linear and binary search.** | C311.5 | BTL5 |
| 7 | **Explain in detail about separate chaining.** | C311.5 | BTL2 |
| 8 | **Explain Rehashing in detail.** | C311.5 | BTL5 |
| 9 | **Explain Extendible hashing in detail.** | C311.5 | BTL5 |