D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Department of Engineering Sciences

Lab Manual

First Year Semester-II

Subject: Data Structures Lab

Even Semester

Index

List of Experiments

Sr. No.	Experiments Name
1	Implement a program to solve Tower of Hanoi problem with n disks using recursion
2	Implement a menu driven program for performing following operations on Stack Data Structures: a. Push b.Pop
3	Implement a program to transform infix expression to postfix expression.
4	Implement a menu driven program for performing following operations on Circular Queue Data Structures: a. Insertion b.Deletion
5	Implement a menu driven program for performing following operations on Singly Linked List Data Structures: a. Insertion and Deletion at Beginning. b. Insertion and Deletion at End. c. Insertion and Deletion in Middle of list based on position entered by user. a. Display status of list
6	Implement a menu driven program for performing following operations on Binary Search Tree (BST) of Integers. a. Create a BST of N Integers. b. Traverse the BST in Inorder, Preorder and Post Order. a. Delete an element from BST.
7	Implement a program for performing following traversal operations on Graph Data Structures: a. Breadth First Search a. Depth First Search
8	Implementation of Binary Search Technique
9	Implement a program to search element from a given set of elements in an array using Hashing Techniques.
10	Case Study – Identify best suited Data structures for solving real world problem.

Course Objective, Course Outcome &Experiment Plan

Course Objective:

1	To implement linear and non-linear data structures.
2	To solve problem involving stacks, queues, linked list, graphs and trees
3	To implement various sorting and searching techniques
4	To use appropriate data structures for real world applications.

Lab Outcomes: At the end of the course students will be able to,

CO1	To differentiate primitive and non-primitive structures.
CO2	To implement operations like insertion, deletion, searching and traversing on various linear data structures.
CO3	To implement operations like insertion, deletion, searching and traversing on various non- linear data structures.
CO4	To apply learned concepts in various domains like DBMS, Compiler Construction
CO5	To demonstrate knowledge of various searching techniques.
CO6	To apply appropriate data structure for solving real world problem.

Module No.	Week No.	Experiments Name	Course Outcome	Weightage
1	W1	Implement a program to solve Tower of Hanoi problem with n disks using recursion	CO1	10
2	W2	Implement a menu driven program for performing following operations on Stack Data Structures: a. Push b.Pop	CO2	03
3	W3	Implement a program to transform infix expression to postfix expression.	CO4	10
4	W4	Implement a menu driven program for performing following operations on Circular Queue Data Structures: a. Insertion b.Deletion	CO2	03
5	W5	Implement a menu driven program for performing following operations on Singly Linked List Data Structures: d. Insertion and Deletion at Beginning. e. Insertion and Deletion at End. f. Insertion and Deletion in Middle of list based on position entered by user. g. Display status of list	CO2	04
6	W6	Implement a menu driven program for performing following operations on Binary Search Tree (BST) of Integers. c. Create a BST of N Integers. d. Traverse the BST in Inorder, Preorder and Post Order. e. Delete an element from BST.	CO3	05
7	W7	Implement a program for performing following traversal operations on Graph Data Structures: b. Breadth First Search c. Depth First Search	CO3	05
8	W9	Implementation of Binary Search Technique	CO5	05
9	W10	Implement a program to search element from a given set of elements in an array using Hashing Techniques.	CO5	05
10	W11	Case Study – Identify best suited Data structures for solving real world problem.	CO6	10

Study and Evaluation Scheme

Course Code	Course Name	Teaching Scheme			Credits Assigned			
FYL202	Data structures Lab	Theory	Practical	Tutorial	Theory	Practical	Tutorial	Total
		--	2	-	-	1	-	1

Course Code	Course Name	Examination Scheme		
FYL202	Data structures Lab	Term Work	Practical & Oral	Total
		25	25	50

Term Work:

- The Term work Marks are based on the weekly experimental performance of the students, Oral performance and regularity in the lab.
- Students are expected to be prepared for the lab ahead of time by referring the manual and perform the experiment under the guidance and discussion. Next week the experiment write-up to be corrected along with oral examination.

Oral and Practical Exam:

End of the semester, there will be Practical examination along with oral evaluation based on the laboratory work and the corresponding theory syllabus.

Data Structures Lab

Experiment No.: 1

Implement a program to solve Tower of Hanoi problem with n disks using Stack

Experiment No. 1

1. **Aim:** Implement a program to solve Tower of Hanoi problem with n disks using Stack

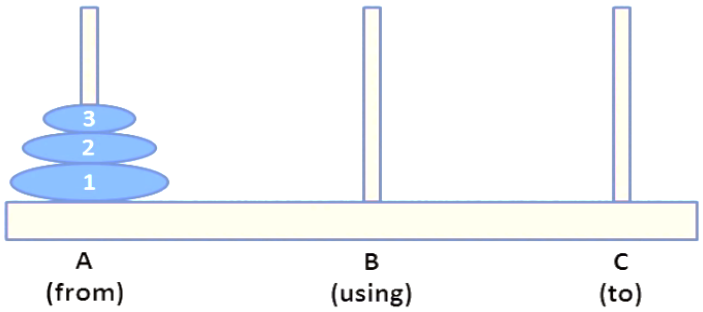
2. **Objectives:**
   - To design and implementation of Tower of Hanoi Problem with n disks.
   - To introduce various techniques for representation of the data in the real world.

3. **Outcomes:**
   - To understand, identify, analyze and design the problem, implement and validate the solution for software.
   - An ability to match the industry requirements in the domains of Programming with the required skills.
   - To engage in lifelong learning

4. **Hardware / Software Required:**Turbo C

5. **Theory:**
   The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. It consists of three poles and a number of disks of different sizes which can slide onto any poles. The puzzle starts with the disk in a neat stack in ascending order of size in one pole, the smallest at the top thus making a conical shape. The objective of the puzzle is to move all the disks from one pole (say 'source pole') to another pole (say 'destination pole') with the help of the third pole (say auxiliary pole) as depicted in figure below.
   The puzzle has the following two rules:

   1. You can't place a larger disk onto a smaller disk
   2. Only one disk can be moved at a time



6. **Algorithm**

Step 1: Calculate the total number of moves required i.e. "pow(2,n) - 1" here n is number of disks.

Step 2: If number of disks (i.e. n) is even then interchange destination pole and auxiliary pole.
Step 3: Iterate from i=1 to total number of moves:
    if i%3 == 1:

Movement of top disk between A pole and C pole
if i%3 == 2:
Movement top disk between A pole and B pole
if i%3 == 0:
Movement top disk between B pole and C pole

7. **Conclusion:** Tower of Hanoi program is a mathematical puzzle consisting of three poles and a number of disks of different sizes which can slide onto any poles.

8. **Viva Questions:**
   - What is a linear data structure?
   - Give two examples of linear data structures.
   - Is it possible to have two designs for the same data structure that provide the same functionality but are implemented differently?

9. **References:**

   - Data Structures using C, ReemaThareja, Oxford
   - Data Structures using C and C++, Rajesh K Shukla, Wiley - India
   - Data Structure Using C, Balagurusamy.
   - Data Structures Using C, Aaron M Tenenbaum, YedidyahLangsam, Moshe JAugenstein, Pearson
   - The Art of Computer Programming: Volume 1: Fundamental Algorithms, Donald E. Knuth.
   - Introduction to Algorithms, Thomas, H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press.

Data Structures Lab

Experiment No.: 2

Implement a menu driven program for performing following operations on Stack Data Structures:

a. Push

b. Pop

Experiment No. 2

1. **Aim:** Implementation of STACK menu driven program

2. **Objectives:**
   - To design and implementation of stack data structures.
   - To introduce various techniques for representation of the data in the real world.

3. **Outcomes:** Students will be able to handle operations like insertion, deletion, searching and traversing on various data structures.

4. **Hardware / Software Required :** Turbo C

5. **Theory:**
   A*stack*isanorderedlistinwhichallinsertionsanddeletionsaremadeatoneend, called the *top*. Given a stack S=(a[1],a[2],.....a[n]) then we say that a1 is the bottommost element and element a[i]) is on top of element a[i-1], 1<i<=n. The restrictions on a stack imply that if the elements A,B,C,D,E are added to the stack , n that order ,then the first element to be removed/deleted must be E. Equivalently we say that the last element to be inserted into the stack will be the first to be removed. For this reason stacks are sometimes referred to as Last In First Out (LIFO)lists.

6. **Algorithm**
   a) Algorithm for Inserting an item into the stack(PUSH)
   PUSH(maxsize,item)
   Let stack[maxsize] is an array for implementing thestack.
   1. Check for stackoverflow
      If top= maxsize – 1, then print : "Stack Overflow" andreturn.
   2. Set top=top+1
   3. Setstack[top]=item
   4. return.

   b) Algorithm for Deleting an item from the stack(POP)
   POP(maxsize)
   1. Check if top < 0 then print : "Stack Underflow" andreturn.
      else set item =stack[top]
   2. Set top = top –1
   3. Return the deleted item from thestack

7. **Conclusion:** We can able to perform insertion of element from top, deletion of element from top and display elements using stack data structures.

8. **Viva Questions:**
   - What is a linear data structure?
   - Give two examples of linear data structures.
   - Is it possible to have two designs for the same data structure that provide the same functionality but are implemented differently?

9. **References:**

   - Kenneth H. Rosen, "Discrete Mathematics and its Applications", Tata McGrawHill.
   - Garry Haggard, John Schlipf, Sue Whitesides. "Discrete Mathematics For Computer Science", Thomson.

- Joe Mott, Abraham Kandel and Theodore Baker, " Discrete Mathematics for Computer Scientist and Mathematicians", Second Edition PHI
- Richard Johnsonbaugh, " Discrete Mathematics " Pearson Education
- https://www.youtube.com/watch?v=jos1Flt21is
- https://www.youtube.com/watch?v=MeRb_1bddWg
- https://ece.uwaterloo.ca/~dwharder/aads/Lecture_materials/3.02.Stacks.pdf

Data Structures Lab

Experiment No.: 3

Implement a program to transform infix expression to postfix expression.

1. **Aim:** Implementation of INFIX to POSTFIX transformation
2. **Objectives:**
   - To develop application using stack data structures.
   - To introduce various techniques for representation of the data in the real world.
   - To improve the logical ability

3. **Outcomes:**
   - Students will be able to apply the learned concepts in various domains like DBMS and Compiler Construction
4. **Hardware / Software Required:** Turbo C

5. **Theory:**

An ***expression*** is defined as a number of operands or data items combined using several operators. There are basically three types of notations for an expression,

1. Infix notation
2. Prefix notation
3. Postfix notation

The **infix** notation is what we come across in our general mathematics, where the operator is written in-between the operands. Example: A+B (note that the operator „+"is written in-between the operands A and B).

The **prefix** notation is one in which the operator is written before the operands, it is also called **polish** notation. The same expression when written in prefix notation looks like: +AB(as operator "+"is written before operands A and B).

In the **postfix** notation the operators are written after the operands ,so it is called postfix notation. It is also known as **reverse polish or suffix** notation. The above expression if written in postfix expression looks like: AB+ (as operator is written after the operands A and B)

The rules to be remembered during infix to postfix conversion are:

1. Parenthesize the expression starting from left to right.
2. During parenthesizing the expression, the operands associated with operator having higher precedence are first parenthesized.
3. The sub-expression (part of expression) which has been converted into postfix is to be treated as single operand.
4. Once the expression is converted to postfix form, remove the parenthesis.

Example : A + (B*C) Parenthesized expression (infix form) A
+ (BC*) Convert the multiplication
A (BC*) + Convert the addition
ABC*+ Postfix form

6. **Algorithm**

a) Algorithm for Infix to Postfix Conversion

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Scan Q from left to right and repeat steps 2 to 4 for each element of Q until the STACK is empty.
2. If an operand is encountered, add it to P
3. If right parenthesis is encountered, then:
    a. Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered.
    b. Remove the left parenthesis (do not add the left parenthesis toP).

4. If an operator is encountered, then check the precedence of incoming operator with the element present at top of stack.
    a. If precedence of incoming operator is higher, then push it onto STACK.
    b. if precedence of incoming operator is less, then
        b.1 Repeatedly pop from STACK and add to P each operator (on the top of STACK)
        b.2 Push the incoming operator onto STACK.
5. Pop the remaining elements from STACK and add it to P.
6. Exit.

7. Conclusion: Thus we can convert the given infix expression into postfix and evaluate postfix expression using STACK.

8. Viva Questions:

- What do you mean by polish notation?
- Difference between postfix and prefix notation?
- How can one convert a postfix expression to its prefix equivalent and vice-versa?

9. References:

- Kenneth H. Rosen, "Discrete Mathematics and its Applications", Tata McGrawHill.
- Garry Haggard, John Schlipf, Sue Whitesides. "Discrete Mathematics For Computer Science", Thomson.
- Joe Mott, Abraham Kandel and Theodore Baker, " Discrete Mathematics for Computer Scientist and Mathematicians", Second Edition PHI
- Richard Johnsonbaugh, " Discrete Mathematics " Pearson Education
- https://www.youtube.com/watch?v=jos1Flt21is
- https://www.youtube.com/watch?v=MeRb_1bddWg
- https://ece.uwaterloo.ca/~dwharder/aads/Lecture_materials/3.02.Stacks.

Data Structures Lab

Experiment No.: 4

Implement a menu driven program for performing following operations on Circular Queue Data Structures:

a. Insertion

b. Deletion

Experiment No. 4

1. **Aim:** Implementation of CIRCULAR QUEUE menu driven program

2. **Objectives:**
   - To design and implementation of Circular Queue data structures.
   - To introduce various techniques for representation of the data in the real world.

3. **Outcomes:**
   - Students will be able to handle operations like insertion, deletion, searching and traversing on various data structures.

4. **Hardware / Software Required :** Turbo C

5. **Theory:**

A *circular queue* is one in which the insertion of an element is done at the very first location of the queue if last location of the queue is full. In other words if we have queue Q of say n elements, then after inserting an element last (i.e., in then-1th) location of the array then ext element will be inserted at the very first location(i.e., location with subscript 0) of the array. It is possible to insert new elements , if and only if those locations(slots)are empty. We can say that a circular queue is one in which the first element comes just after the last element. It can be viewed as a mesh or loop of wire, in which the two ends of the wire are connected together. A circular queue over comes the problem of unutilized space in linear queues implemented as arrays. A circular queue also have a front and rear to keep the track of the elements to be deleted and inserted and therefore to maintain the unique characteristics of the queue.

6. **Algorithm**

Algorithm for Inserting an item in a circular queue.

CQINSERT (maxsize,item)

Let cqueue[maxsize] is an array for implementing aqueue.

i. Initialize front = -1 and rear = -1

ii. Check, if rear = maxsize -1 then set rear =0
else set rear = rear +1

iii. Check, if rear = front then print : "Circular queueoverflow"
Also check, if rear = 0 then set rear = maxsize –1 else set rear =
rear -1 andreturn.

iv. Set cqueue[rear] =item

v. Check, if front = -1 then set front =0

vi. Return

Algorithm for Deleting an item from a circular queue.

CQDELETE(maxsize)

i. Check, if front = -1 then print : "Circular queue underflow" andreturn

ii. Set item =cqueue[front]

iii. Check, if front = rear then set front = -1 and rear =-1
else check, if front = maxsize – 1 then set front =0
else set front = front +1

iv. Return the deleted element from circular queue.

7.**Output Observation/Analysis:** Students should perform experiments with all different cases, do the analysis and should write in their own language.

8. Additional Learning: Any additional information provided by the faculty should be written here.

9. Conclusion: Thus we can analyze circular queue by visualizing the one- dimensional array as circular. The circular queue overcomes the problems associated when deletion is made from the front. And another one is that even if the queue in reality is empty, the normal queues give full condition.

10.Viva Questions:

- What is the significance circular queue?
- How insertion and deletion will be performed in circular queue?
- What is the empty and overflow condition in circular queue?

11. References:

- Kenneth H. Rosen, "Discrete Mathematics and its Applications", Tata McGrawHill.
- Garry Haggard, John Schlipf, Sue Whitesides. "Discrete Mathematics For Computer Science", Thomson.
- Joe Mott, Abraham Kandel and Theodore Baker, " Discrete Mathematics for Computer Scientist and Mathematicians", Second Edition PHI
- Richard Johnsonbaugh, " Discrete Mathematics " Pearson Education
- https://ece.uwaterloo.ca/~dwharder/aads/Lecture_materials/3.03.Queues.pdf
- https://ece.uwaterloo.ca/~dwharder/aads/Lecture_materials/7.01.Priority_queues.pdf
- https://www.youtube.com/watch?v=PGWZUgzDMYI

Data Structures Lab

Experiment No.: 5

Implement a menu driven program for performing following operations on Singly Linked List Data Structures:

a. **Insertion and Deletion at Beginning.**
b. **Insertion and Deletion at End.**
c. **Insertion and Deletion in Middle of list based on position entered by user.**
d. **Display status of list**

Experiment No. 5

1. **Aim:** Implement a menu driven program for the following operations on Singly Linked List Data Structure.
    - Insert a node at the beginning
    - Insert a node at the end
    - Insert a node after the specified node
    - Delete a first node
    - Delete a last node
    - Delete a node after a specified node

2. **Objectives:**
    - To design and implementation of Linked list data structures.
    - To introduce various techniques for representation of the data in the real world.

3. **Outcomes:**
    - Student will be able to handle operations like insertion, deletion, traversing mechanism etc. on linked list data structures.
    - To understand, identify, analyze and design the problem, implement and validate the solution for software.
    - An ability to match the industry requirements in the domains of Programming with the required skills.
    - To engage in lifelong learning

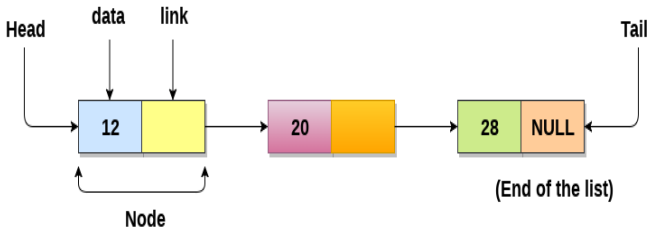
4. **Hardware / Software Required:** Turbo C

5. **Theory:**

A singly linked list is 2 successive node of the linked list linked with each other in sequential linear manner. Movement in forward direction is possible. The linked list consists of a series of structures. They are not required to be stored in adjacent memory locations. Each structure consists of a data field and address field. Address field contains the address of its successors. A linked list can be created using dynamic memory allocation as well. This reduces the chances of wastage of memory.

Basic Linked list operations:

1. Creating s linked list
2. Check whether list is empty
3. Inserting an item
4. Deleting an item
5. Searching an item
6. Retrieving a node



7. Displaying the list
8. Find length of list

6. Algorithm

a) Algorithm to insert a node at the beginning

Step 1: IF PTR = NULL
Write OVERFLOW
Go to Step 7
[END OF IF]
Step 2: SET NEW_NODE = PTR
Step 3: SET PTR = PTR → NEXT
Step 4: SET NEW_NODE → DATA = VAL
Step 5: SET NEW_NODE → NEXT = HEAD
Step 6: SET HEAD = NEW_NODE
Step 7: EXIT

b) Algorithm to insert a node at the end

Step 1: IF PTR = NULL Write OVERFLOW
Go to Step 1
[END OF IF]
Step 2: SET NEW_NODE = PTR
Step 3: SET PTR = PTR - > NEXT
Step 4: SET NEW_NODE - > DATA = VAL
Step 5: SET NEW_NODE - > NEXT = NULL
Step 6: SET PTR = HEAD
Step 7: Repeat Step 8 while PTR - > NEXT != NULL
Step 8: SET PTR = PTR - > NEXT
[END OF LOOP]
Step 9: SET PTR - > NEXT = NEW_NODE
Step 10: EXIT

c) Algorithm to insert a node after the specified node

STEP 1: IF PTR = NULL
WRITE OVERFLOW
GOTO STEP 12
END OF IF
STEP 2: SET NEW_NODE = PTR
STEP 3: NEW_NODE → DATA = VAL
STEP 4: SET TEMP = HEAD
STEP 5: SET I = 0
STEP 6: REPEAT STEP 5 AND 6 UNTIL I
STEP 7: TEMP = TEMP → NEXT
STEP 8: IF TEMP = NULL

WRITE "DESIRED NODE NOT PRESENT"
GOTO STEP 12
END OF IF
END OF LOOP
STEP 9: PTR → NEXT = TEMP → NEXT
STEP 10: TEMP → NEXT = PTR
STEP 11: SET PTR = NEW_NODE
STEP 12: EXIT

d) Algorithm to delete a first node

Step 1: IF HEAD = NULL
Write UNDERFLOW
Go to Step 5
[END OF IF]
Step 2: SET PTR = HEAD
Step 3: SET HEAD = HEAD -> NEXT
Step 4: FREE PTR
Step 5: EXIT

e) Algorithm to delete a last node

Step 1: IF HEAD = NULL
Write UNDERFLOW
Go to Step 8
[END OF IF]
Step 2: SET PTR = HEAD
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT!= NULL
Step 4: SET PREPTR = PTR
Step 5: SET PTR = PTR -> NEXT
[END OF LOOP]
Step 6: SET PREPTR -> NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT

f) Algorithm to delete a node after a specified node

STEP 1: IF HEAD = NULL
WRITE UNDERFLOW
GOTO STEP 10
END OF IF
STEP 2: SET TEMP = HEAD
STEP 3: SET I = 0
STEP 4: REPEAT STEP 5 TO 8 UNTIL I
STEP 5: TEMP1 = TEMP
STEP 6: TEMP = TEMP → NEXT
STEP 7: IF TEMP = NULL
WRITE "DESIRED NODE NOT PRESENT"

GOTO STEP 12
END OF IF
STEP 8: I = I+1
END OF LOOP
STEP 9: TEMP1 → NEXT = TEMP → NEXT
STEP 10: FREE TEMP
STEP 11: EXIT

7. Output Observation/Analysis: Students should perform experiments with all different cases, do the analysis and should write in their own language.

Test Case No.	Test Case Description	Test Data	Expected Output	Actual Output

8. Additional Learning: Any additional information provided by the faculty should be written here.

9. Conclusion: Thus, we can easily perform insertion and deletion of nodes on singly linked list structure. Also accessibility of a node in the forward direction is easier.

.

10. Viva Questions:

- What are the differences between a linked list and an array?
- What is a queue and how can you implement queue using a linked list?
- What is a stack and how can you implement stack using a linked list?

11. References:

- Data Structures using C, ReemaThareja, Oxford
- Data Structures using C and C++, Rajesh K Shukla, Wiley - India
- Data Structure Using C, Balagurusamy.
- Data Structures Using C, Aaron M Tenenbaum, YedidyahLangsam, Moshe J Augenstein, Pearson

Data Structures Lab

Experiment No. : 6

Implement a menu driven program for the following operations on Binary Search Tree (BST) of Integers.

a. **Create a BST of N Integers.**
b. **Traverse the BST in Inorder, Preorder and Post Order.**
c. **Delete an element from BST.**

Experiment No. 6

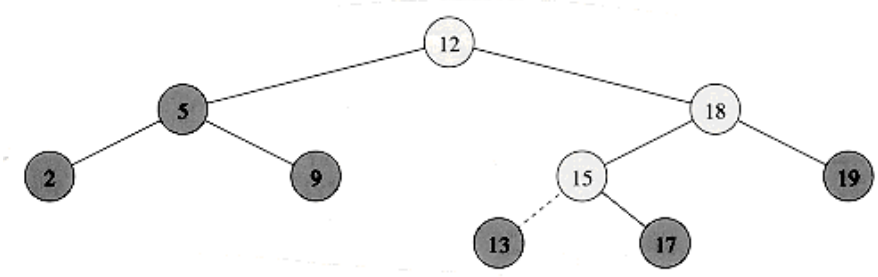
1. **Aim:** Implement a menu driven program for the following operations on Binary Search Tree (BST) of Integers.
   Create a BST of N Integers.
   Traverse the BST in Inorder, Preorder and Post Order.
   Delete an element from BST.
2. **Objectives:**
   To design and implementation of binary search tree data structures.
   To introduce various techniques for representation of the data in the real world.

3. **Outcomes:**
   Student will be able to handle operations like searching, insertion, deletion, traversing mechanism etc. on binary search tree data structures.
   To understand, identify, analyze and design the problem, implement and validate the solution for software.
   An ability to match the industry requirements in the domains of Programming with the required skills.
   To engage in lifelong learning

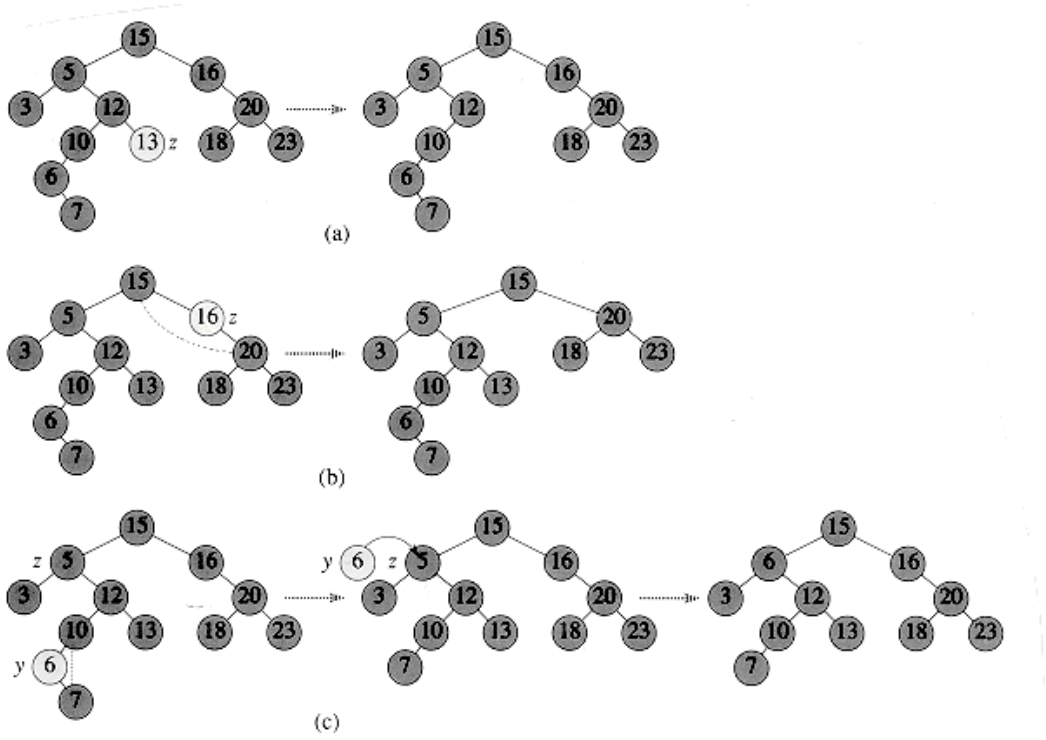
4. **Hardware / Software Required :** Turbo C

5. **Theory:**
   A binary search tree or BST is a binary tree that is either empty or in which the data element of each node has a key,and:
   1. All keys in the left subtree (if there is one) are less than the key in the root node.
   2. All keys in the right subtree (if there is one) are greater than (or equal to) the key in the root node.
   3. The left and right subtrees of the root are binary search trees.

   Insertion: Inserting an item with key 13 into a binary search tree. Lightly shaded nodes indicate the path from the root down to the position where the item is inserted. The dashed line indicates the link in the tree that is added to insert the item.



   Deletion: Deleting a node z from a binary search tree. In each case, the node actually removed is lightly shaded. (a) If z has no children, we just remove it. (b) If z has only onechild,wespliceoutz.(c)Ifzhastwochildren,wespliceoutitssuccessory, which has at most one child, and then replace the contents of z with the contents of y.



1. Algorithm

Insertion:

To insert a new value *v* into a binary search tree *T*. The procedure is passed a node *z* for which *key*[*z*] = *v*, *left*[*z*] = NIL, and *right*[*z*] = NIL. It modifies *T* and some of the fields of *z* in such a way that *z* is inserted into an appropriate position in the tree.

```
1.  y ← NIL
2.  x ← root[T]
3.  while x ≠ NIL
4.     do y ← x
5.  if key[z] < key[x]
6.        then x ← left[x]
7.        else x ← right[x]
8.     p[z] ← y
9.     if y = NIL
10.    then root[T] ← z
11.  else if key[z] < key[y]
12.       then left[y] ← z
13.  else right[y] ← z
```

TREE-INSERT begins at the root of the tree and traces a path downward. The pointer *x* traces the path, and the pointer *y* is maintained as the parent of *x*. After initialization, the **while** loop in lines 3-7 causes these two pointers to move down the tree, going left or right depending on the comparison of *key*[*z*] with *key*[*x*],until *x* is set to NIL.This NIL occupies the position where we wish to place the input item *z*. Lines 8-13 set the pointers that cause *z* to be inserted.

Deletion: The procedure for deleting a given node *z* from a binary search tree takes as an argument a pointer to *z*. If *z* has no children, we modify its parent *p*[*z*] to replace *z* with NIL as its child. If the node has only a single child, we "splice out" *z* by making a new link between its child and its parent. Finally, if the node has two children, we splice out *z*'s successor *y*, which has no left child and replace the contents of *z* with the contents of *y*.

TREE-DELETE(*T*,*z*)

1. **if** *left*[*z*] = NIL or *right*[*z*] =NIL
2. **then** *y* ← *z*
3. **else** *y* ← TREE-SUCCESSOR(*z*)
4. **if** *left*[*y*] ≠NIL
5. **then** *x* ← *left*[*y*]
6. **else** *x* ← *right*[*y*]
7. **if** *x* ≠NIL
8. **then** *p*[*x*] ← *p*[*y*]
9. **if** *p*[*y*] = NIL
10. **then** *root*[*T*] ← *x*
11. **else if** *y* = *left*[*p*[*y*]]
12. **then** *left*[*p*[*y*]] ← *x*
13. **else** *right*[*p*[*y*]] ← *x*
14. **if** *y* ≠*z*
15. **then** *key*[*z*] ← *key*[*y*]
16. If *y* has other fields, copy them,too.
17. **return** *y*

TREESUCCESSOR(*x*)

1. **if** *right*[*x*] ≠NIL
2. **then return** TREE-MINIMUM(*right*[*x*])
3. *y* ← *p*[*x*]
4. **while** *y* ≠NIL and *x* =*right*[*y*]
5. **do** *x* ← y
6. *y* ← *p*[*y*]
7. **return** *y*

TREE-MINIMUM(*x*)

1. **while** *left*[*x*] ≠NIL
2. **do** *x* ← *left*[*x*]
3. **return** *x*

2. **Conclusion:** The keys in a binary search tree are always stored in such a way as to satisfy the ***binary-search-tree property***: The binary-search-tree property allows us to print out all the keys in a binary search tree in sorted order by a simple recursive algorithm, called an ***inordertreewalk***. The algorithm derives its name from the fact that the key of the root of a subtree is printed between the values in its left subtree and those in its right subtree.

3. **Viva Questions:**
   - How can you find successors and predecessors in a binary search tree in order
   - Draw the binary search tree whose elements are inserted in the following order:
     2. 2 96 94 107 26 12
4. **References:**

   - Kenneth H. Rosen, “Discrete Mathematics and its Applications”, Tata McGrawHill.
   - Garry Haggard, John Schlipf, Sue Whitesides. “Discrete Mathematics For Computer Science”, Thomson.

- Joe Mott, Abraham Kandel and Theodore Baker, " Discrete Mathematics for Computer Scientist and Mathematicians", Second Edition PHI
- Richard Johnsonbaugh, " Discrete Mathematics " Pearson Education
- http://www.geeksforgeeks.org/618/
- http://geeksquiz.com/binary-search-tree-set-1-search-and-insertion/
- https://www.quora.com/How-can-you-find-successors-and-predecessors-in-a-binary-search-tree-in-order
- http://nptel.ac.in/courses/106106127/50
- http://nptel.ac.in/courses/106103069/45

Data Structures Lab

Experiment No.: 7

Implement a program for performing following traversal operations on Graph Data Structures:

a. **Breadth First Search**
b. **Depth First Search**

Experiment No. 7

1. **Aim:** Implement a program for performing following traversal operations on Graph Data Structures:
    c. Breadth First Search
    d. Depth First Search

2. Objectives:

- Students will be able to select efficient searching algorithms to solve common programming problems
- An ability to match the industry requirements in the domains of Programming with the required skills.
- To engage in lifelong learning

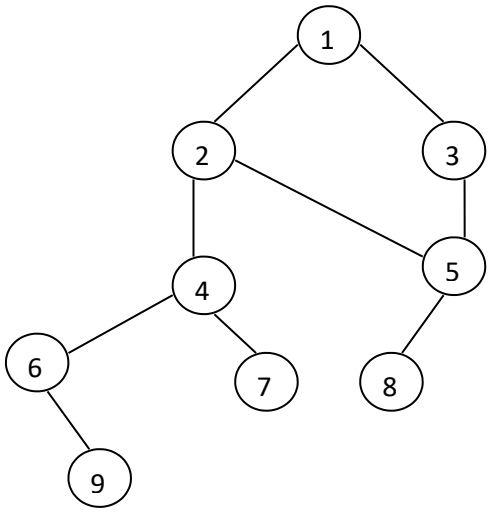
3. Outcomes:

- Student will be able to handle operations like creation, traversing mechanism etc. on graph data structures.
- To understand, identify, analyze and design the problem, implement and validate the solution for software.
- An ability to match the industry requirements in the domains of Programming with the required skills.

4. Hardware / Software Required: Turbo C

5. Theory: BREADTH FIRST SEARCH

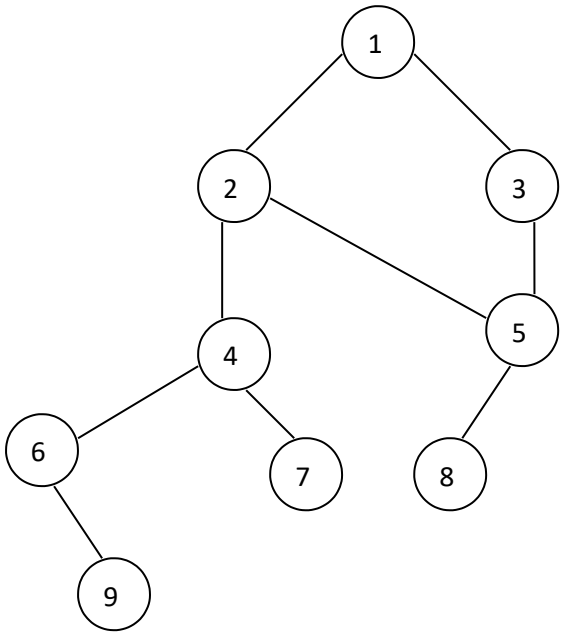
This is the method, instead of going deep into the graph, we examine the nodes across the breadth of a tree, before going to the next level. Unlike DFS, where we visit the node then push it on to the top of the stack. The next node should be unvisited one, if no such node exists, we backtrack the most LRU node. In BFS we use queue. In this method, we pick a node to visit first and place its unprocessed adjacent nodes in queue. Repeatedly, take a node from the front of a queue. If this node is unvisited, then we visit the node and then place all its neighbours in thequeue.



BFS: 1 2 3 4 5 6 7 8 9

DEPTH FIRST SEARCH

This is the method of visiting all nodes, by traversing all nodes as deeply as possible. When there are no adjacent, non-visited nodes, then we proceed backwards (backtrack), and we repeat the process, by maintaining the stack to keep track of visited nodes, so that this can help in backtracking.



DFS: 1 2 4 6 9 7 5 8 3

6. Algorithm

Algorithm of BFS

1. Select any node in the graph. Mark this node as visited.
2. Find the adjacent node, add them to the queue.
3. Visit this node, which is at the front of the queue. Delete this node form queue and place its adjacent in node in queue. (We can use adjacency matrix for it). Make this new node as visited.
4. Repeat step 2 and 3 till queue becomes empty.
5. Stop.

Algorithm of DFS

1. Select any node in the graph. Mark this node as visited, push this node on the stack.
2. Find the adjacent node to the node on top of stack, and which, and which is not yet visited. (We can use adjacency matrix for it). Make this new node as visited and push onto the stack.
3. Repeat step 2, until no new adjacent node to the top of the stack, node can be found. When no new adjacent node can be found, pop the top of the stack.
4. Repeat step 2 and 3 till stack becomes empty.
5. Repeat above steps, if there are any more nodes which are still unvisited.

7. Conclusion: In this way, we have studied breadth wise graph traversal& depth first search technique which uses queue for implementation.

8. Viva Questions:

- Show that the number of vertices of odd degree in a finite graph is even.
- Prove that the edges explored by a breadth first or depth first traversal of a connected graph from a tree
- Explain how existence of a cycle in an undirected graph may be detected by traversing the graph in a depth first manner.

9. References:

- Kenneth H. Rosen, “Discrete Mathematics and its Applications”, Tata McGrawHill.
- Garry Haggard, John Schlipf, Sue Whitesides. “Discrete Mathematics for Computer Science”, Thomson.
- Joe Mott, Abraham Kandel and Theodore Baker, “Discrete Mathematics for Computer Scientist and Mathematicians”, Second Edition PHI
- Richard Johnsonbaugh, “Discrete Mathematics” Pearson Education
- "https://en.wikibooks.org/wiki/Data Structures /Graphs
- https://www.youtube.com/watch?v=r1-8p11fSPw
- http://nptel.ac.in/courses/106105078/pdf/Lesson%2004.pdf

Data Structures Lab

Experiment No.: 8

Implementation of Binary Search Technique

Experiment No. 8

1.Aim: Implement a program to perform Binary Search Technique on a given set of elements in an array.

2. **Objectives:**

- To improve the logical ability
- To learn efficient searching mechanisms for accessing data.

1. Outcomes:

- Students will be able to select appropriate searching technique for given problem.

4. Hardware / Software Required: Turbo C

5. Theory:

- Binary search is a searching algorithm that works efficiently with a sorted list. The mechanism of binary search can be better understood by an analogy of a telephone directory. When we are searching for a particular name in a directory, we first open the directory from the middle and then decide whether to look for the name in the first part of the directory or in the second part of the directory. Again, we open some page in the middle and the whole process is repeated until we finally find the right name.
- Now, let us consider how this mechanism is applied to search for a value in a sorted array. Consider an array A[] that is declared and initialized as int A[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; and the value to be searched is VAL = 9.
- The algorithm will proceed in the following manner. BEG = 0, END = 10, MID = (0 + 10)/2 = 5 Now, VAL = 9 and A[MID] = A[5] = 5 A[5] is less than VAL, therefore, we now search for the value in the second half of the array. So, we change the values of BEG and MID.
- Now, BEG = MID + 1 = 6, END = 10, MID = (6 + 10)/2 =16/2 = 8 VAL = 9 and A[MID] = A[8] = 8 A[8] is less than VAL, therefore, we now search for the value in the second half of the segment. So, again we change the values of BEG and MID. Now, BEG = MID + 1 = 9, END = 10, MID = (9 + 10)/2 = 9 Now, VAL = 9 and A[MID] = 9.
- In this algorithm, we see that BEG and END are the beginning and ending positions of the segment that we are looking to search for the element. MID is calculated as (BEG + END)/2. Initially, BEG = lower_bound and END = upper_bound. The algorithm will terminate when A[MID] = VAL.
- When the algorithm ends, we will set POS = MID. POS is the position at which the value is present in the array. However, if VAL is not equal to A[MID], then the values of BEG, END, and MID will be changed depending on whether VAL is smaller or greater than A[MID].

  a)If VAL < A[MID], then VAL will be present in the left segment of the array. So, the value of END will be changed as END = MID – 1.

  b) If VAL > A[MID], then VAL will be present in the right segment of the array. So, the value of BEG will be changed as BEG = MID + 1.

6. Algorithm

BINARY_SEARCH(A, lower_bound, upper_bound, VAL)
Step 1: [INITIALIZE] SET BEG=lower_bound
END=upper_bound, POS=-1
Step 2: Repeat Steps3and4while BEG <= END
Step 3: SET MID=(BEG+END)/2
Step 4: IF A[MID]=VAL
SET POS=MID
PRINT POS
Go to Step 6
ELSE IF A[MID]>VAL
SET END=MID-1
ELSE
SET BEG=MID+1
[END OF IF]
[END OF LOOP]
Step 5: IF POS=1
PRINT "VALUE IS NOT PRESENT IN THE ARRAY"
[END OF IF]
Step 6: EXIT

7. Output Observation/Analysis: Students should perform experiments with all different cases, do the analysis and should write in their own language.

Test Case No.	Test Case Description	Test Data	Expected Output	Actual Output

8.Additional Learning: Any additional information provided by the faculty should be written here.

9.Conclusion:

Thus, Binary search is a searching algorithm is implemented that works efficiently with a sorted list

10. Viva Questions:

- What is searching and different types of searching techniques?
- Which technique is more efficient?

11. References:

- Data Structures using C, ReemaThareja, Oxford
- Data Structures using C and C++, Rajesh K Shukla, Wiley - India
- Data Structure Using C, Balagurusamy.
- Data Structures Using C, Aaron M Tenenbaum, YedidyahLangsam, Moshe J Augenstein, Pearson
- The Art of Computer Programming: Volume 1: Fundamental Algorithms, Donald E. Knuth.
- Introduction to Algorithms, Thomas, H. Cormen, Charles E. Leiserson, Ronald L. Rivest,

Clifford Stein, The MIT Press.

Data Structures Lab

Experiment No.: 9

Implement a program to search element from a given set of elements in an array using Hashing Techniques

Experiment No. 9

1. Aim: Implement a program to search element from a given set of elements in an array using Hashing Techniques

2. Objectives:

- Students will be able to understand hashing with linear probing and process adapted for searching.
- An ability to match the industry requirements in the domains of Programming with the required skills.

3. Outcomes:

- To investigate the hashing mechanism and applicability of the mechanism in various applications.
- To understand, identify, analyze and design the problem, implement and validate the solution for software.
- An ability to match the industry requirements in the domains of Programming with the required skills.
- To engage in lifelong learning

4. Hardware / Software Required: Turbo C

5. Theory:

A hash table is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots. A hash table is a data structure which is used to store key-value pairs. Hash function is used by hash table to compute an index into an array in which an element will be inserted or searched.

Linear probing is a collision resolving technique in Open Addressed Hash tables. In this method, each cell of a hash table stores a single key–value pair. If a collision is occurred by mapping a new key to a cell of the hash table that is already occupied by another key. This method searches the table for the following closest free location and inserts the new key there.

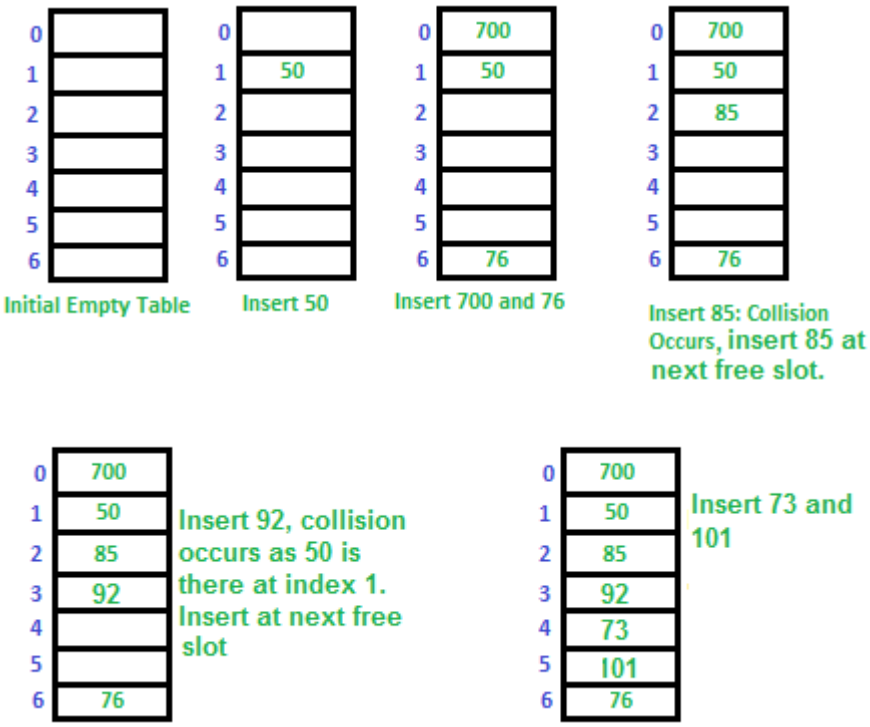
Example: In linear probing, we linearly probe for next slot. For example, the typical gap between two probes is 1 as seen in the example below.

Let hash(x) be the slot index computed using a hash function and S be the table size

If slot hash(x) % S is full, then we try (hash(x) + 1) % S

If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S
If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S

Let us consider a simple hash function as “key mod 7” and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.



Similar way the Key is searched in the array by calculating hkey and index.

6. Algorithm:

a) Algorithm for inserting element in array using linear probing

Hashtable is an array of size = TABLE_SIZE

Step 1: Read the value to be inserted, key

Step 2: let i = 0

Step 3: hkey = key% TABLE_SIZE

Step 4: compute the index at which the key has to be inserted in hash table

index = (hkey + i) % TABLE_SIZE

Step 5: if there is no element at that index then insert the value at index and STOP

Step 6: If there is already an element at that index

step 4.1: i = i+1

step 7: if i < TABLE_SIZE then go to step 4

b) Algorithm for Searching element in array using linear probing

Hashtable is an array of size = TABLE_SIZE

Step 1: Read the value to be searched, key

Step 2: let i = 0

Step 3: hkey = key% TABLE_SIZE

Step 4: compute the index at which the key can be found

index = (hkey+ i) % TABLE_SIZE

Step 5: if the element at that index is same as the search value, then print element found and STOP

Step 6: else

step 4.1: i = i+1

step 7: if i < TABLE_SIZE then go to step 4

7. **Conclusion:** Student can get knowledge of hashing and insertion and searching using linear probing mechanism.

8. **Viva Questions:**

1. What is Hashing? What are the advantages of Hashing?

2. How linear probing works? How it is different than other search mechanisms?

3. What are the limitation of hashing using linear probing?

9. References:

- https://quescol.com/data-structure/linear-probing
- https://www.log2base2.com/algorithms/searching/linear-probing-hash-table.html
- http://web.stanford.edu/class/archive/cs/cs166/cs166.1166/lectures/12/Small12.pdf

Data Structures Lab

Experiment No.: 10

Case Study

Experiment No. 10

Case Study

Topics to be included:

1. **Introduction**

   Theory background of system functionalities

2. **Problem Description**

   Identify best suited data structures to solve the problem with justification. Apply various operations on the selected data structures.

3. **Algorithm**
4. **Summary**