ARUNAI ENGINEERING COLLEGE

(Affiliated to Anna University)

Velu Nagar, Thiruvannamalai-606 603
www.arunai.org

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

BACHELOR OF ENGINEERING
2021 - 2022

THIRD SEMESTER

CS3361 – DATA STRUCTURES LAB

CS3361 DATA STRUCTURES LABORATORY L T P C 0 0 3 1.5

OBJECTIVES

- To implement linear and non-linear data structures
- To understand the different operations of search trees
- To implement graph traversal algorithms
- To get familiarized to sorting and searching algorithms

1. Array implementation of Stack, Queue and Circular Queue ADTs
2. Implementation of Singly Linked List
3. Linked list implementation of Stack and Linear Queue ADTs
4. Implementation of Polynomial Manipulation using Linked list
5. Implementation of Evaluating Postfix Expressions, Infix to Postfix conversion
6. Implementation of Binary Search Trees
7. Implementation of AVL Trees
8. Implementation of Heaps using Priority Queues
9. Implementation of Dijkstra’s Algorithm
10. Implementation of Prim’s Algorithm
11. Implementation of Linear Search and Binary Search
12. Implementation of Insertion Sort and Selection Sort
13. Implementation of Merge Sort
14. Implementation of Open Addressing (Linear Probing and Quadratic Probing)

TOTAL: 45 PERIODS

PROGRAMME OUTCOMES (POs)

After going through the four years of study, computer science & engineering graduates will exhibit :

	Graduate Attribute	Programme Outcome
1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2	Problem analysis	Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and designsystem components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, andsynthesis of the information to provide valid conclusions
5	Modern tool usage	Create, select, and apply appropriate techniques, resources, andmodern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6	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

7	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.
8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10	Communication	Communicate effectively on complex engineering activities with the engineering community and with the 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
11	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
12	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

PROGRAM SPECIFIC OUTCOMES (PSOs)

1. To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.
2. To apply software engineering principles and practices for developing quality software for scientific and business applications.
3. To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems.

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS:

SOFTWARE:

- C / C++

HARDWARE:

Standalone desktops - 30 Nos. (or) Server supporting 30 terminals or more.

OUTCOMES:

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

- To demonstrate array implementation of linear data structure algorithms.
- To implement the applications using Stack.
- To implement the applications using Linked list
- To implement Binary search tree and AVL tree algorithms.
- To implement the Heap algorithm.
- To implement Dijkstra’s algorithm.
- To implement Prim’s algorithm
- To implement Sorting, Searching and Hashing algorithms.

Course Outcomes	Description	Level in Bloom’s Taxonomy
CO1	To demonstrate array implementation of linear data structure algorithms.	K2
CO2	To implement the applications using Stack.	K3
CO3	To implement the applications using Linked list	K3
CO4	To implement Binary search tree and AVL tree algorithms.	K3
CO5	To implement the Heap algorithm	K3
CO6	To implement Dijkstra’s algorithm	K3
CO7	To implement Prim’s algorithm	A3
CO8	To implement Sorting, Searching and Hashing algorithms	A3

CO - PO MATRIX

Course Outcomes	Programme Outcome (POs)											
	K3	K4	K4	K5	K3,K4, K5	A3	A2	A3	A3	A3	A3	A2
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	2	2	-	-	2	1	1	-	-	-	-
CO2	2	2	2	1	-	1	1	1	-	-	-	-
CO3	1	3	2	-	-	2	1	1	-	-	-	-
CO4	2	2	2	2	-	2	2	1	-	-	-	-
CO5	3	2	1	1	-	2	1	1	-	-	-	-
CO6	2	1	1	1	-	1	2	1	-	-	-	-
CO7	-	-	-	-	-	-	-	3	-	-	-	-
CO8	-	-	-	-	-	-	-	-	3	-	3	-
CO9	-	-	-	-	-	-	-	-		3	-	-
CO10												3
	2	2	2	2	-	2	2	1	3	3	3	3

CO - PSO MATRIX

	PSO1	PSO2	PSO3
CO1	2	1	1
CO2	2	1	1
CO3	2	1	1
CO4	2	1	1
CO5	2	1	1
CO6	2	1	1
CO7	-	-	-
CO8	-	-	-
CO9	-	-	-
CO10			
	2	1	1

MODE OF ASSESSMENT

EVALUATION PROCEDURE FOR EACH EXPERIMENT

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	20
2.	Observation	20
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	20
Total		**100**

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Observation	05
2.	Performance	05
3.	Viva voce	05
4.	Record	05
Total		**20**

ABOUT THE SOFTWARE

C / C++

C is a structural or procedural oriented programming language which is machine-independent and extensively used in various applications. C is the basic programming language that can be used to develop from the operating systems (like Windows) to complex programs like Oracle database, Git, Python interpreter, and many more. C programming language can be called a god's programming language as it forms the base for other programming languages. If we know the C language, then we can easily learn other programming languages. C language was developed by the great computer scientist Dennis Ritchie at the Bell Laboratories. It contains some additional features that make it unique from other programming languages.

C++ is a special-purpose programming language developed by **Bjarne Stroustrup** at Bell Labs circa 1980. C++ language is very similar to C language, and it is so compatible with C that it can run 99% of C programs without changing any source of code though C++ is an object-oriented programming language, so it is safer and well-structured programming language than C.

Differences between C and C++:

- **Definition**
  C is a structural programming language, and it does not support classes and objects, while C++ is an object-oriented programming language that supports the concept of classes and objects.
- **Subset**
  C++ is a superset of C programming language. C++ can run 99% of C code but C language cannot run C++ code.
- **Type of approach**
  C follows the top-down approach, while C++ follows the bottom-up approach. The top-down approach breaks the main modules into tasks; these tasks are broken into sub-tasks, and so on. The bottom-down approach develops the lower level modules first and then the next level modules.

- **Security**

  In C, the data can be easily manipulated by the outsiders as it does not support the encapsulation and information hiding while C++ is a very secure language, i.e., no outsiders can manipulate its data as it supports both encapsulation and data hiding. In C language, functions and data are the free entities, and in C++ language, all the functions and data are encapsulated in the form of objects.
- **Function Overloading**

  Function overloading is a feature that allows you to have more than one function with the same name but varies in the parameters. C does not support the function overloading, while C++ supports the function overloading.
- **Function Overriding**

  Function overriding is a feature that provides the specific implementation to the function, which is already defined in the base class. C does not support the function overriding, while C++ supports the function overriding.
- **Keywords**

  C contains 32 keywords, and C++ supports 52 keywords.
- **Namespace feature**

  A namespace is a feature that groups the entities like classes, objects, and functions under some specific name. C does not contain the namespace feature, while C++ supports the namespace feature that avoids the name collisions.
- **Exception handling**

  C does not provide direct support to the exception handling; it needs to use functions that support exception handling. C++ provides direct support to exception handling by using a try-catch block.
- **Input/Output functions**

  In C, scanf and printf functions are used for input and output operations, respectively, while in C++, cin and cout are used for input and output operations, respectively.
- **Memory allocation and de-allocation**

  C supports calloc() and malloc() functions for the memory allocation, and free() function for the memory de-allocation. C++ supports a new operator for the memory allocation and delete operator for the memory de-allocation.

LIST OF EXPERIMENTS

Ex. No	List of Experiments	Page No
1. a	Stack using Array	
1. b	Queue using Array	
1.c	Circular Queue using Array	
2	Singly Linked List	
3. a	Stack Using Linked List	
3. b	Queue Using Linked List	
4	Polynomial Manipulation Using Linked List	
5. a	Infix to Postfix Conversion	
5. b	Postfix Expression Evaluation	
6	Binary Search Tree	
7	AVL trees	
8	Binary Heap	
9	Dijkstra's Shortest Path	
10	Prim’s Algorithm	
11. a	Linear Search	
11. b	Binary Search	
12. a	Insertion Sort	
12. b	Selection Sort	
13	Merge Sort	
14	Open Addressing Hashing Technique	

Ex. No. 1a **Stack Array**
Date:

Aim

To implement stack operations using array.

Algorithm

1. **Start**
2. **Define a array *stack* of size *max* = 5**
3. **Initialize *top* = -1**
4. **Display a menu listing stack operations**
5. **Accept choice**
6. **If choice = 1 then**
   - **If top < max -1**
     - **Increment top**
     - **Store element at current position of top**
   - **Else**
     - **Print Stack overflow**

   Else If choice = 2 then
   - **If top < 0 then**
     - **Print Stack underflow**
   - **Else**
     - **Display current top element**
     - **Decrement top**

   Else If choice = 3 then
   - **Display stack elements starting from top**
7. **Stop**

Program

```
/* Stack Operation using Arrays */

#include <stdio.h>
#include <conio.h>

#define max 5

static int stack[max];
int top = -1;

void push(int x)
{
   stack[++top] = x;
}

int pop()
{
   return (stack[top--]);
}

void view()
{
   int i;
   if (top < 0)
      printf("\n Stack Empty \n");
   else
   {
      printf("\n Top-->");
      for(i=top; i>=0; i--)
      {
         printf("%4d", stack[i]);
      }
      printf("\n");
   }
}

main()
{
   int ch=0, val;
   clrscr();

   while(ch != 4)
   {
      printf("\n STACK   OPERATION \n");
      printf("1.PUSH  ");
      printf("2.POP  ");
      printf("3.VIEW  ");
      printf("4.QUIT \n");
      printf("Enter Choice : ");
```

```
        scanf("%d", &ch);

        switch(ch)
        {
           case 1:
              if(top < max-1)
              {
                 printf("\nEnter Stack element : ");
                 scanf("%d", &val);
                 push(val);
              }
              else
                 printf("\n Stack Overflow \n");
              break;
           case 2:
              if(top < 0)
                 printf("\n Stack Underflow \n");
              else
              {
                 val = pop();
                 printf("\n Popped element is %d\n", val);
              }
              break;
           case 3:
              view();
              break;
           case 4:
              exit(0);
           default:
              printf("\n Invalid Choice \n");
        }
    }
}
```

Output

```
STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 12

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 23
```

```
STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 34

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 45

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 3

Top-->  45  34  23  12

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 2

Popped element is 45

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 3

Top-->  34  23  12

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 4
```

Result

Thus push and pop operations of a stack was demonstrated using arrays.

Ex. No. 1b **Queue Array**
Date:

Aim

To implement queue operations using array.

Algorithm

1. **Start**
2. **Define a array *queue* of size *max* = 5**
3. **Initialize *front* = *rear* = –1**
4. **Display a menu listing queue operations**
5. **Accept choice**
6. **If choice = 1 then**
   If rear < max -1
      Increment rear
      Store element at current position of rear
   Else
      Print Queue Full
   Else If choice = 2 then
   If front = –1 then
      Print Queue empty
   Else
      Display current front element
      Increment front
   Else If choice = 3 then
      Display queue elements starting from front to rear.
7. **Stop**

Program

```
/* Queue Operation using Arrays */

#include <stdio.h>
#include <conio.h>

#define max 5

static int queue[max];
int front = -1;
int rear  = -1;

void insert(int x)
{
   queue[++rear] = x;
   if (front == -1)
      front = 0;
}

int remove()
{
   int val;
   val = queue[front];
   if (front==rear && rear==max-1)
      front = rear = -1;
   else
      front ++;
   return (val);
}

void view()
{
   int i;

   if (front == -1)
      printf("\n Queue Empty \n");
   else
   {
      printf("\n Front-->");
      for(i=front; i<=rear; i++)
         printf("%4d", queue[i]);
      printf("  <--Rear\n");
   }
}

main()
{
   int ch= 0,val;
   clrscr();
```

```
    while(ch != 4)
    {
       printf("\n QUEUE OPERATION \n");
       printf("1.INSERT  ");
       printf("2.DELETE  ");
       printf("3.VIEW  ");
       printf("4.QUIT\n");
       printf("Enter Choice : ");
       scanf("%d", &ch);

       switch(ch)
       {
          case 1:
             if(rear < max-1)
             {
                printf("\n Enter element to be inserted : ");
                scanf("%d", &val);
                insert(val);
             }
             else
                printf("\n Queue Full \n");
             break;
          case 2:
             if(front == -1)
                printf("\n Queue Empty \n");
             else
             {
                val = remove();
                printf("\n Element deleted : %d \n", val);
             }
             break;
          case 3:
             view();
             break;
          case 4:
             exit(0);
          default:
             printf("\n Invalid Choice \n");
       }
    }
}
```

Output

```
 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 12

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 23

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 34

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 45

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 56

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Queue Full

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 3

Front-->  12  23  34  45  56  <--Rear
```

Result

Thus insert and delete operations of a queue was demonstrated using arrays.

Ex. No. 1c **Circular Queue using Array**
Date:

Aim

To implement circular queue operations using array.

Algorithm

Program

```
/*static circular queue*/
#include <stdio.h>
#include <conio.h>
#define size 5

void insertq(int[], int);
void deleteq(int[]);
void display(int[]);

int front =  - 1;
int rear =  - 1;

int main()
{
    int n, ch;
    int queue[size];
    clrscr();
    do
    {
        printf("\n\n Circular Queue:\n1. Insert \n2.
Delete\n3. Display\n0. Exit");
        printf("\nEnter Choice 0-3? : ");
        scanf("%d", &ch);
        switch (ch)
        {
```

```
            case 1:
                printf("\nEnter  number: ");
                scanf("%d", &n);
                insertq(queue, n);
                break;
            case 2:
                deleteq(queue);
                break;
            case 3:
                display(queue);
                break;
        }
    }while (ch != 0);
}


void insertq(int queue[], int  item)
{
    if((front == 0 && rear==size - 1) || (front == rear + 1))
    {
        printf("queue is full");
        return;
    }
    else if (rear ==  - 1)
    {
        rear++;
```

```
        front++;
    }
    else if (rear == size - 1 && front > 0)
    {
        rear = 0;
    }
    else
    {
        rear++;
    }
    queue[rear] = item;
}

void display(int queue[])
{
    int i;
    printf("\n");
    if (front > rear)
    {
        for (i = front; i < size;
i++)
        {
            printf("%d ", queue[i]);
        }
        for (i = 0; i <= rear; i++)
            printf("%d ", queue[i]);
```

```
    }
    else
    {
        for (i = front; i <= rear; i++)
            printf("%d ", queue[i]);
    }
}

void deleteq(int queue[])
{
    if (front ==  - 1)
    {
        printf("Queue is empty");
    }
    else if (front == rear)
    {
        printf("\n %d deleted", queue[front]);
        front =  - 1;
        rear =  - 1;
    }
    else
    {
        printf("\n %d deleted", queue[front]);
        front++;
    }
}
```

Output

Result

Thus insert and delete operations of a circular queue was demonstrated using arrays.

Ex. No. 2 **Singly Linked List**
Date:

Aim

To define a singly linked list node and perform operations such as insertions and deletions dynamically.

Algorithm

1. **Start**
2. **Define single linked list *node* as self referential structure**
3. **Create *Head* node with label = -1 and next = NULL using**
4. **Display menu on list operation**
5. **Accept user choice**
6. **If choice = 1 then**
   Locate node after which insertion is to be done
   Create a new node and get data part
   Insert new node at appropriate position by manipulating addressElse if choice = 2
   Get node's data to be deleted. Locate
   the node and delink the nodeRearrange
   the links
   Else
   Traverse the list from Head node to node which points to null
7. **Stop**

Program

```
/* Single Linked List */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>
#include <string.h>

struct node
{
   int label;
   struct node *next;
};

main()
{
   int ch, fou=0;
   int k;
   struct node *h, *temp, *head, *h1;

   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->label = -1;
   head->next = NULL;

   while(-1)
   {
      clrscr();
      printf("\n\n SINGLY LINKED LIST OPERATIONS \n");
      printf("1->Add ");
      printf("2->Delete  ");
      printf("3->View ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         /* Add a node at any intermediate location */
         case 1:
            printf("\n Enter label after which to add : ");
            scanf("%d", &k);

            h = head;
            fou = 0;

            if (h->label == k)
               fou = 1;
```

```
   while(h->next != NULL)
   {
      if (h->label == k)
      {
            fou=1;
            break;
      }
      h = h->next;
   }
   if (h->label == k)
      fou = 1;

   if (fou != 1)
      printf("Node not found\n");
   else
   {
   temp=(struct node *)(malloc(sizeof(struct node)));
      printf("Enter label for new node : ");
      scanf("%d", &temp->label);
      temp->next = h->next;
      h->next = temp;
   }
   break;

/* Delete any intermediate node */
case 2:
   printf("Enter label of node to be deleted\n");
   scanf("%d", &k);
   fou = 0;
   h = h1 = head;
   while (h->next != NULL)
   {
      h = h->next;
      if (h->label == k)
      {
            fou = 1;
            break;
      }
   }

   if (fou == 0)
      printf("Sorry Node not found\n");
   else
      {
      while (h1->next != h)
            h1 = h1->next;
      h1->next = h->next;
      free(h);
      printf("Node deleted successfully \n");
   }
   break;
```

```
         case 3:
            printf("\n\n HEAD -> ");
            h=head;
            while (h->next != NULL)
            {
               h = h->next;
               printf("%d -> ",h->label);
            }
            printf("NULL");
            break;

         case 4:
            exit(0);
      }
   }
}
```

Output

```
SINGLY LINKED LIST OPERATIONS
1->Add  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label after which new node is to be added : -1
Enter label for new node : 23

SINGLY LINKED LIST OPERATIONS
1->Add  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label after which new node is to be added : 23
Enter label for new node : 67

SINGLY LINKED LIST OPERATIONS
1->Add  2->Delete  3->View  4->Exit
Enter your choice : 3
 HEAD -> 23 -> 67 -> NULL
```

Result

Thus operation on single linked list is performed.

Ex. No. 3a **Stack Using Linked List**
Date:

Aim

To implement stack operations using linked list.

Algorithm

1. **Start**
2. **Define a singly linked list node for stack**
3. **Create Head node**
4. **Display a menu listing stack operations**
5. **Accept choice**
6. **If choice = 1 then**
    Create a new node with data
    Make new node point to first node
    Make head node point to new node
   Else If choice = 2 then
    Make temp node point to first node
    Make head node point to next of temp node
    Release memory
   Else If choice = 3 then
    Display stack elements starting from head node till null
7. **Stop**

Program

```
/* Stack using Single Linked List */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>

struct node
{
   int label;
   struct node *next;
};

main()
{
   int ch = 0;
   int k;
   struct node *h, *temp, *head;

   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->next = NULL;

   while(1)
   {
      printf("\n Stack using Linked List \n");
      printf("1->Push  ");
      printf("2->Pop  ");
      printf("3->View ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         case 1:
            /* Create a new node */
            temp=(struct node *)(malloc(sizeof(struct node)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);
            h = head;
            temp->next = h->next;
            h->next = temp;
            break;

         case 2:
            /* Delink the first node */
            h = head->next;
            head->next = h->next;
```

```
            printf("Node %s deleted\n", h->label);
            free(h);
            break;

         case 3:
            printf("\n HEAD -> ");
            h = head;
            /* Loop till last node */
            while(h->next != NULL)
            {
               h = h->next;
               printf("%d -> ",h->label);
            }
            printf("NULL \n");
            break;

         case 4:
            exit(0);
      }
   }
}
```

Output

```
 Stack using Linked List
1->Push  2->Pop  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 23
New node added

 Stack using Linked List
1->Push  2->Pop  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 34

 Stack using Linked List
1->Push  2->Pop  3->View  4->Exit
Enter your choice : 3
HEAD -> 34 -> 23 -> NULL
```

Result

Thus push and pop operations of a stack was demonstrated using linked list.

Ex. No. 3b **Queue Using Linked List**
Date:

Aim

To implement queue operations using linked list.

Algorithm

1. **Start**
2. **Define a singly linked list node for stack**
3. **Create Head node**
4. **Display a menu listing stack operations**
5. **Accept choice**
6. **If choice = 1 then**
   - **Create a new node with data**
   - **Make new node point to first node**
   - **Make head node point to new node**

   Else If choice = 2 then
   - **Make temp node point to first node**
   - **Make head node point to next of temp node**
   - **Release memory**

   Else If choice = 3 then
   - **Display stack elements starting from head node till null**
7. **Stop**

Program

```
/* Queue using Single Linked List */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>

struct node
{
   int label;
   struct node *next;
};

main()
{
   int ch=0;
   int k;
   struct node *h, *temp, *head;

   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->next = NULL;

   while(1)
   {
      printf("\n Queue using Linked List \n");
      printf("1->Insert  ");
      printf("2->Delete  ");
      printf("3->View ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         case 1:
            /* Create a new node */
            temp=(struct node *)(malloc(sizeof(struct node)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);

            /* Reorganize the links */
            h = head;
            while (h->next != NULL)
               h = h->next;
            h->next = temp;
            temp->next = NULL;
            break;
```

```
        case 2:
           /* Delink the first node */
           h = head->next;
           head->next = h->next;
           printf("Node deleted \n");
           free(h);
           break;

        case 3:
           printf("\n\nHEAD -> ");
           h=head;
           while (h->next!=NULL)
           {
              h = h->next;
              printf("%d -> ",h->label);
           }
           printf("NULL \n");
           break;

        case 4:
           exit(0);
     }
  }
}
```

Output

```
 Queue using Linked List
1->Insert  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 12

 Queue using Linked List
1->Insert  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 23

 Queue using Linked List
1->Insert  2->Delete  3->View  4->Exit
Enter your choice : 3
HEAD -> 12 -> 23 -> NULL
```

Result

Thus insert and delete operations of a queue was demonstrated using linked list.

Ex. No. 4 **Polynomial Addition**
Date:

Aim

To add any two given polynomial using linked lists.

Algorithm

1. **Create a structure for polynomial with exp and coeff terms.**
2. **Read the coefficient and exponent of given two polynomials p and q.**
3. **While p and q are not null, repeat step 4.**
   If powers of the two terms are equal then
   Insert the sum of the terms into the sum Polynomial
   Advance p and q
   Else if the power of the first polynomial> power of second then
   Insert the term from first polynomial into sum polynomial
   Advance p
   Else
   Insert the term from second polynomial into sum polynomial
   Advance q
4. **Copy the remaining terms from the non empty polynomial into the sum polynomial**
5. **Stop**

Program

```
/* Polynomial Addition */

/* Add two polynomials */

#include <stdio.h>
#include <malloc.h>
#include <conio.h>

struct link
{
     int coeff;
     int pow;
     struct link *next;
};
struct link *poly1=NULL,*poly2=NULL,*poly=NULL;

void create(struct link *node)
{
     char ch;
     do
     {
          printf("\nEnter coefficient: ");
          scanf("%d", &node->coeff);
          printf("Enter exponent: ");
          scanf("%d", &node->pow);
          node->next = (struct link*)malloc(sizeof(struct
                                                   link));
          node = node->next;
          node->next = NULL;
          printf("\n continue(y/n): ");
          fflush(stdin);
          ch=getch();
     } while(ch=='y' || ch=='Y');
}

void show(struct link *node)
{
     while(node->next!=NULL)
    {
          printf("%dx^%d", node->coeff, node->pow);
          node=node->next;
          if(node->next!=NULL)
               printf(" + ");
     }
}

void polyadd(struct link *poly1, struct link *poly2, struct
                                             link *poly)
{
```

```
    while(poly1->next &&  poly2->next)
    {
        if(poly1->pow > poly2->pow)
       {
             poly->pow = poly1->pow;
         poly->coeff = poly1->coeff;
         poly1 = poly1->next;
        }
       else if(poly1->pow < poly2->pow)
        {
             poly->pow = poly2->pow;
             poly->coeff = poly2->coeff;
         poly2 = poly2->next;
        }
       else
       {
             poly->pow = poly1->pow;
             poly->coeff = poly1->coeff + poly2->coeff;
             poly1 = poly1->next;
             poly2 = poly2->next;
        }

       poly->next=(struct link *)malloc(sizeof(struct link));
       poly=poly->next;
       poly->next=NULL;
    }
    while(poly1->next || poly2->next)
    {
        if(poly1->next)
    {
             poly->pow = poly1->pow;
             poly->coeff = poly1->coeff;
             poly1 = poly1->next;
        }
       if(poly2->next)
        {
             poly->pow = poly2->pow;
             poly->coeff = poly2->coeff;
             poly2 = poly2->next;
        }
       poly->next = (struct link *)malloc(sizeof(struct
                                                        link));
        poly = poly->next;
        poly->next = NULL;
    }
}

main()
{
    poly1 = (struct link *)malloc(sizeof(struct link));
    poly2 = (struct link *)malloc(sizeof(struct link));
```

```
poly = (struct link *)malloc(sizeof(struct link));
```

```
     printf("Enter 1st Polynomial:");
     create(poly1);
     printf("\nEnter 2nd Polynomial:");
     create(poly2);
     printf("\nPoly1: ");
     show(poly1);
     printf("\nPoly2: ");
     show(poly2);
     polyadd(poly1, poly2, poly);
     printf("\nAdded Polynomial: ");
    show(poly);
}
```

Output

```
Enter 1st Polynomial:
Enter coefficient: 5
Enter exponent: 2
 continue(y/n): y
Enter coefficient: 4
Enter exponent: 1
 continue(y/n): y
Enter coefficient: 2
Enter exponent: 0
 continue(y/n): n

Enter 2nd Polynomial:
Enter coefficient: 5
Enter exponent: 1
 continue(y/n): y
Enter coefficient: 5
Enter exponent: 0
 continue(y/n): n

Poly1: 5x^2 + 4x^1 + 2x^0
Poly2: 5x^1 + 5x^0
Added Polynomial: 5x^2 + 9x^1 + 7x^0
```

Result

Thus the two given polynomials were added using lists.

Ex. No. 5a **Infix To Postfix Conversion**
Date:

Aim

To convert infix expression to its postfix form using stack operations.

Algorithm

1. **Start**
2. **Define a array *stack* of size *max* = 20**
3. **Initialize *top* = -1**
4. **Read the infix expression character-by-character**
    - **If character is an operand print it**
    - **If character is an operator**
        - **Compare the operator's priority with the stack[top] operator.**
        - **If the stack [top] has higher/equal priority than the input operator,**
            - **Pop it from the stack and print it.**
        - **Else**
            - **Push the input operator onto the stack**
    - **If character is a left parenthesis, then push it onto the stack.**
    - **If character is a right parenthesis, pop all operators from stack and print it until a left parenthesis is encountered. Do not print the parenthesis.**
    - **If character = $ then Pop out all operators, Print them and Stop**

.

Program

```
/* Conversion of infix to postfix expression */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 20

int top = -1;
char stack[MAX];
char pop();
void push(char item);

int prcd(char symbol)
{
   switch(symbol)
   {
      case '+':
      case '-':
         return 2;
         break;
      case '*':
      case '/':
         return 4;
         break;
      case '^':
      case '$':
         return 6;
         break;
      case '(':
      case ')':
      case '#':
         return 1;
         break;
   }
}

int isoperator(char symbol)
{
   switch(symbol)
   {
      case '+':
      case '-':
      case '*':
      case '/':
      case '^':
      case '$':
      case '(':
      case ')':
         return 1;
```

```
            break;
        default:
            return 0;
    }
}

void convertip(char infix[],char postfix[])
{
    int i,symbol,j = 0;
    stack[++top] = '#';
    for(i=0;i<strlen(infix);i++)
    {
        symbol = infix[i];
        if(isoperator(symbol) == 0)
        {
            postfix[j] = symbol;
            j++;
        }
        else
        {
            if(symbol == '(')
                push(symbol);
            else if(symbol == ')')
            {
                while(stack[top] != '(')
                {
                    postfix[j] = pop();
                    j++;
                }
                pop(); //pop out (.
            }
            else
            {
                if(prcd(symbol) > prcd(stack[top]))
                    push(symbol);
                else
                {
                    while(prcd(symbol) <= prcd(stack[top]))
                    {
                          postfix[j] = pop();
                          j++;
                    }
                    push(symbol);
                }
            }
        }
    }

    while(stack[top] != '#')
    {
        postfix[j] = pop();
```

```
        j++;
    }
    postfix[j] = '\0';
}

main()
{
    char  infix[20],postfix[20];
    clrscr();
    printf("Enter the valid infix string: ");
    gets(infix);
    convertip(infix, postfix);
    printf("The corresponding postfix string is: ");
    puts(postfix);
    getch();
}

void push(char item)
{
    top++;
    stack[top] = item;
}

char pop()
{
    char a;
    a = stack[top];
    top--;
    return a;
}
```

Output

```
Enter the valid infix string: (a+b*c)/(d$e)
The corresponding postfix string is: abc*+de$/

Enter the valid infix string: a*b+c*d/e
The corresponding postfix string is: ab*cd*e/+

Enter the valid infix string: a+b*c+(d*e+f)*g
The corresponding postfix string is: abc*+de*f+g*+
```

Result

Thus the given infix expression was converted into postfix form using stack.

Ex. No. 5b **Postfix Expression Evaluation**
Date:

Aim

To evaluate the given postfix expression using stack operations.

Algorithm

1. **Start**
2. **Define a array *stack* of size *max* = 20**
3. **Initialize *top* = -1**
4. **Read the postfix expression character-by-character**
   - **If character is an operand push it onto the stack**
   - **If character is an operator**
     - **Pop topmost two elements from stack.**
     - **Apply operator on the elements and push the result onto the stack,**
5. **Eventually only result will be in the stack at end of the expression.**
6. **Pop the result and print it.**
7. **Stop**

Program

```
/* Evaluation of Postfix expression using stack */

#include <stdio.h>
#include  <conio.h>

struct stack
{
   int top;
   float a[50];
}s;


main()
{
   char pf[50];
   float d1,d2,d3;
   int i;
   clrscr();

   s.top = -1;
   printf("\n\n Enter the postfix expression: ");
   gets(pf);
   for(i=0; pf[i]!='\0'; i++)
   {
      switch(pf[i])
      {
         case '0':
         case '1':
         case '2':
         case '3':
         case '4':
         case '5':
         case '6':
         case '7':
         case '8':
         case '9':
            s.a[++s.top] = pf[i]-'0';
            break;

         case '+':
            d1 = s.a[s.top--];
            d2 = s.a[s.top--];
            s.a[++s.top] = d1 + d2;
            break;

         case '-':
            d2 = s.a[s.top--];
            d1 = s.a[s.top--];
            s.a[++s.top] = d1 - d2;
```

```
break;
```

```
         case '*':
            d2 = s.a[s.top--];
            d1 = s.a[s.top--];
            s.a[++s.top] = d1*d2;
            break;

         case '/':
            d2 = s.a[s.top--];
            d1 = s.a[s.top--];
            s.a[++s.top] = d1 / d2;
            break;
      }
   }
   printf("\n Expression value is %5.2f", s.a[s.top]);
   getch();
}
```

Output

```
Enter the postfix expression: 6523+8*+3+*
Expression value is 288.00
```

Result

Thus the given postfix expression was evaluated using stack.

Ex. No. 6 **Binary Search Tree**
Date:

Aim

To insert and delete nodes in a binary search tree.

Algorithm

1. **Create a structure with key and 2 pointer variable left and right.**
2. **Read the node to be inserted.**

   If (root==NULL)
   root=node
   else if (root->key<node->key)
   root->right=NULL
   else
   Root->left=node
3. **For Deletion**

   if it is a leaf node
   Remove immediately
   Remove pointer between del node & child
   if it is having one child
   Remove link between del node&child
   Link delnode is child with delnodes parent
   If it is a node with a children
   Find min value in right subtree
   Copy min value to delnode place
   Delete the duplicate
4. **Stop**

Program

```
/* Binary Search Tree */

#include <stdio.h>
#include <stdlib.h>

struct node
{
   int key;
   struct node *left;
   struct node *right;
};

struct node *newNode(int item)
{
   struct node *temp = (struct node *)malloc(sizeof(struct
                                                        node));
   temp->key = item;
   temp->left = temp->right = NULL;
   return temp;
}

void inorder(struct node *root)
{
   if (root != NULL)
   {
      inorder(root->left);
      printf("%d ", root->key);
      inorder(root->right);
   }
}

struct node* insert(struct node* node, int key)
{
   if (node == NULL)
      return newNode(key);
   if (key < node->key)
      node->left  = insert(node->left, key);
    else
      node->right = insert(node->right, key);
   return node;
}

struct node * minValueNode(struct node* node)
{
   struct node* current = node;
   while (current->left != NULL)
      current = current->left;
   return current;
}
```

```
struct node* deleteNode(struct node* root, int key)
{
   struct node *temp;
   if (root == NULL)
      return root;
   if (key < root->key)
      root->left = deleteNode(root->left, key);
   else if (key > root->key)
      root->right = deleteNode(root->right, key);
   else
   {
      if (root->left == NULL)
      {
         temp = root->right;
         free(root);
         return temp;
      }
      else if (root->right == NULL)
      {
         temp = root->left;
         free(root);
         return temp;
      }
      temp = minValueNode(root->right);
      root->key = temp->key;
      root->right = deleteNode(root->right, temp->key);
   }
   return root;
}

main()
{
   struct node *root = NULL;
   root = insert(root, 50);
   root = insert(root, 30);
   root = insert(root, 20);
   root = insert(root, 40);
   root = insert(root, 70);
   root = insert(root, 60);
   root = insert(root, 80);
   printf("Inorder traversal of the given tree \n");
   inorder(root);
   printf("\nDelete 20\n");
   root = deleteNode(root, 20);
   printf("Inorder traversal of the modified tree \n");
   inorder(root);
   printf("\nDelete 30\n");
   root = deleteNode(root, 30);
   printf("Inorder traversal of the modified tree \n");
   inorder(root);
   printf("\nDelete 50\n");
```

```
    root = deleteNode(root, 50);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);
}
```

Output

```
Inorder traversal of the given tree
20 30 40 50 60 70 80
Delete 20
Inorder traversal of the modified tree
30 40 50 60 70 80
Delete 30
Inorder traversal of the modified tree
40 50 60 70 80
Delete 50
Inorder traversal of the modified tree
40 60 70 80
```

Result

Thus nodes were inserted and deleted from a binary search tree.

Ex. No. 7 **AVL Trees**
Date:

Aim

To perform insertion operation on an AVL tree and to maintain balance factor.

Algorithm

1. **Start**
2. **Perform standard BST insert for w**
3. **Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.**
4. **Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways.**
   a) **y is left child of z and x is left child of y (Left Left Case)**
   b) **y is left child of z and x is right child of y (Left Right Case)**
   c) **y is right child of z and x is right child of y (Right Right Case)**
   d) **y is right child of z and x is left child of y (Right Left Case)**
5. **Stop**

Program

```
/* AVL Tree */

#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <stdlib.h>

#define CHANGED 0
#define BALANCED 1

typedef struct bnode
{
   int data,bfactor;
   struct bnode *left;
   struct bnode *right;
}node;

int height;

void displaymenu()
{
   printf("\nBasic Operations in AVL tree");
   printf("\n0.Display menu list");
   printf("\n1.Insert a node in AVL tree");
   printf("\n2.View AVL tree");
   printf("\n3.Exit");
}

node* getnode()
{
   int size;
   node *newnode;
   size = sizeof(node);
   newnode =  (node*)malloc(size);
   return(newnode);
}

void copynode(node *r, int data)
{
   r->data = data;
   r->left = NULL;
   r->right = NULL;
   r->bfactor = 0;
}

void releasenode(node *p)
{
   free(p);
}
```

```
node* searchnode(node *root, int data)
{
   if(root!=NULL)
      if(data < root->data)
         root = searchnode(root->left, data);
      else if(data > root->data)
         root = searchnode(root->right, data);
   return(root);
}

void lefttoleft(node **pptr, node **aptr)
{
   node *p = *pptr, *a = *aptr;
   printf("\nLeft to Left AVL rotation");
   p->left = a->right;
   a->right = p;
   if(a->bfactor == 0)
   {
      p->bfactor = 1;
      a->bfactor = -1;
      height = BALANCED;
   }
   else
   {
      p->bfactor = 0;
      a->bfactor = 0;
   }
   p = a;
   *pptr = p;
   *aptr = a;
}

void lefttoright(node **pptr, node **aptr, node **bptr)
{
   node *p = *pptr, *a = *aptr, *b = *bptr;
   printf("\nLeft to Right AVL rotation");
   b = a->right;
   b->right = p;
   if(b->bfactor == 1)
      p->bfactor = -1;
   else
      p->bfactor = 0;
   if(b->bfactor == -1)
      a->bfactor = 1;
   else
      a->bfactor = 1;
   b->bfactor = 0;
   p = b;
   *pptr = p;
   *aptr = a;
   *bptr = b;
```

```
}

void righttoright(node **pptr, node **aptr)
{
   node *p = *pptr, *a = *aptr;
   printf("\nRight to Right AVL rotation");
   p->right = a->left;
   a->left = p;
   if(a->bfactor == 0)
   {
      p->bfactor = -1;
      a->bfactor = 1;
      height = BALANCED;
   }
   else
   {
      p->bfactor = 0;
      a->bfactor = 0;
   }
   p = a;
   *pptr = p;
   *aptr = a;
}

void righttoleft(node **pptr, node **aptr, node **bptr)
{
   node *p = *pptr, *a = *aptr, *b = *bptr;
   printf("\nRight to Left AVL rotation");
   b = a->left;
   a->left = b->right;
   b->right = a;
   p->right = b->left;
   b->left = p;
   if(b->bfactor == -1)
      p->bfactor = 1;
   else
      p->bfactor = 0;
   if(b->bfactor == -1)
      a->bfactor = 0;
   b->bfactor = 0;
   p = b;
   *pptr = p;
   *aptr = a;
   *bptr = b;
}

void inorder(node *root)
{
   if(root == NULL)
      return;
   inorder(root->left);
```

```
   printf("\n%4d",  root->data);
   inorder(root->right);
}

void view(node *root, int level)
{
   int k;
   if(root == NULL)
      return;
   view(root->right,  level+1);
   printf("\n");
   for(k=0; k<level; k++)
      printf("  ");
   printf("%d", root->data);
   view(root->left,  level+1);
}

node* insertnode(int data, node *p)
{
   node *a,*b;
   if(p == NULL)
   {
      p=getnode();
      copynode(p,  data);
      height = CHANGED;
      return(p);
  }
  if(data < p->data)
  {
      p->left = insertnode(data,  p->left);
      if(height == CHANGED)
      {
         switch(p->bfactor)
         {
            case -1:
               p->bfactor = 0;
               height = BALANCED;
               break;
            case 0:
               p->bfactor = 1;
               break;
            case 1:
               a = p->left;
               if(a->bfactor == 1)
                    lefttoleft(&p,  &a);
               else
                    lefttoright(&p, &a, &b);
               height = BALANCED;
               break;
         }
      }
```

```
    }
    if(data > p->data)
    {
       p->right = insertnode(data, p->right);
       if(height == CHANGED)
       {
          switch(p->bfactor)
          {
             case 1:
                p->bfactor = 0;
                height = BALANCED;
                break;
             case 0:
                p->bfactor = -1;
                break;
             case -1:
                a=p->right;
                if(a->bfactor == -1)
                     righttoright(&p, &a);
                else
                     righttoleft(&p, &a, &b);
                height=BALANCED;
                break;
          }
       }
    }
    return(p);
 }

 main()
 {
    int data, ch;
    char choice = 'y';
    node *root = NULL;
    clrscr();
    displaymenu();
    while((choice == 'y') || (choice == 'Y'))
    {
       printf("\nEnter your choice: ");
       fflush(stdin);
       scanf("%d",&ch);
       switch(ch)
       {
          case 0:
             displaymenu();
             break;
          case 1:
             printf("Enter the value to be inserted ");
             scanf("%d", &data);
             if(searchnode(root, data) == NULL)
                root = insertnode(data, root);
```

```
            else
               printf("\nData already exists");
               break;
         case 2:
            if(root == NULL)
            {
               printf("\nAVL tree is empty");
               continue;
            }
            printf("\nInorder traversal of AVL tree");
            inorder(root);
            printf("\nAVL tree is");
            view(root, 1);
            break;
         case 3:
            releasenode(root);
            exit(0);
      }
   }
   getch();
}
```

Output

```
Basic Operations in AVL tree
0.Display menu list
1.Insert a node in AVL tree
2.View AVL tree
3.Exit
Enter your choice: 1
Enter the value to be inserted 1

Enter your choice: 1
Enter the value to be inserted 2

Enter your choice: 1
Enter the value to be inserted 3

Right to Right AVL rotation

Enter your choice: 1
Enter the value to be inserted 4

Enter your choice: 1
Enter the value to be inserted 5

Right to Right AVL rotation
```

```
Enter your choice: 1
Enter the value to be inserted 6

Right to Right AVL rotation

Enter your choice: 1
Enter the value to be inserted 7

Right to Right AVL rotation

Enter your choice: 1
Enter the value to be inserted 8

Enter your choice: 2
Inorder traversal of AVL tree
   1
   2
   3
   4
   5
   6
   7
   8
AVL tree is
        8
      7
    6
      5
  4
      3
    2
      1
Enter your choice: 3
```

Result

Thus rotations were performed as a result of insertions to AVL Tree.

Ex. No. 8 **Binary Heap**
Date:

Aim

To build a binary heap from an array of input elements.

Algorithm

1. **Start**
2. **In a heap, for every node x with parent p, the key in p is smaller than or equal to the key in x.**
3. **For insertion operation**
   a. **Add the element to the bottom level of the heap.**
   b. **Compare the added element with its parent; if they are in the correct order, stop.**
   c. **If not, swap the element with its parent and return to the previous step.**
4. **For deleteMin operation**
   a. **Replace the root of the heap with the last element on the last level.**
   b. **Compare the new root with its children; if they are in the correct order, stop.**
   c. **If not, Swap with its smaller child in a min-heap**
5. **Stop**

Program

```
/* Binary Heap */

#include <stdio.h>
#include <limits.h>

int heap[1000000], heapSize;

void Init()
{
   heapSize = 0;
   heap[0] = -INT_MAX;
}

void Insert(int element)
{
   heapSize++;
   heap[heapSize] = element;
   int now = heapSize;
   while (heap[now / 2] > element)
   {
      heap[now] = heap[now / 2];
      now /= 2;
   }
   heap[now] = element;
}

int DeleteMin()
{
   int minElement, lastElement, child, now;
   minElement = heap[1];
   lastElement = heap[heapSize--];
   for (now = 1; now * 2 <= heapSize; now = child)
   {
      child = now * 2;
      if (child != heapSize && heap[child + 1] < heap[child])
         child++;
      if (lastElement > heap[child])
         heap[now] = heap[child];
      else
         break;
   }
   heap[now] = lastElement;
   return minElement;
}

main()
{
   int number_of_elements;
   printf("Program to demonstrate Heap:\nEnter the number of
```

```
                                                   elements: ");
   scanf("%d",  &number_of_elements);
   int iter, element;
   Init();
   printf("Enter the elements: ");
   for (iter = 0; iter < number_of_elements; iter++)
   {
      scanf("%d",  &element);
      Insert(element);
   }
   for (iter = 0; iter < number_of_elements; iter++)
      printf("%d ", DeleteMin());
   printf("\n");
}
```

Output

```
Program to demonstrate Heap:
Enter the number of elements: 6
Enter the elements: 3 2 15 5 4 45

2 3 4 5 15 45
```

Result

Thus a binary heap is constructed for the given elements.

Ex. No. 9 **Dijkstra's Shortest Path**
Date:

Aim

To find the shortest path for the given graph from a specified source to all other vertices using Dijkstra's algorithm.

Algorithm

1. **Start**
2. **Obtain no. of vertices and adjacency matrix for the given graph**
3. **Create cost matrix from adjacency matrix. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity**
4. **Initialize visited[] to zero**
5. **Read source vertex and mark it as visited**
6. **Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex**
   distance[i]=cost[0][i];
7. **Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.**
8. **Recalculate the shortest distance of remaining vertices from the source.**
9. **Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v**
   if(visited[v]==0)
   distance[v]=min(distance[v]
   distance[w]+cost[w][v])
10. **Stop**

Program

```
/* Dijkstra's Shortest Path */

#include <stdio.h>
#include <conio.h>

#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

main()
{
    int G[MAX][MAX], i, j, n, u;
    printf("Enter no. of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            scanf("%d", &G[i][j]);
    printf("Enter the starting node: ");
    scanf("%d", &u);
    dijkstra(G, n, u);
}

void dijkstra(int G[MAX][MAX], int n,int startnode)
{
    int cost[MAX][MAX], distance[MAX], pred[MAX];
    int visited[MAX],count, mindistance, nextnode, i, j;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            if(G[i][j] == 0)
                cost[i][j] = INFINITY;
            else
                cost[i][j] = G[i][j];
    for(i=0; i<n; i++)
    {
        distance[i] = cost[startnode][i];
        pred[i] = startnode;
        visited[i] = 0;
    }
    distance[startnode] = 0;
    visited[startnode] = 1;
    count = 1;
    while(count < n-1)
    {
        mindistance = INFINITY;
        for(i=0; i<n; i++)
            if(distance[i] < mindistance && !visited[i])
            {
```

```
                mindistance = distance[i];
                nextnode=i;
            }
            visited[nextnode] = 1;
            for(i=0; i<n; i++)
                if(!visited[i])
                    if(mindistance + cost[nextnode][i] <
                                             distance[i])
                    {
                        distance[i] = mindistance +
                                        cost[nextnode][i];
                        pred[i] = nextnode;
                    }
        count++;
    }

    for(i=0; i<n; i++)
        if(i != startnode)
        {
            printf("\nDistance to node%d = %d", i,
                                          distance[i]);
            printf("\nPath = %d", i);
            j = i;
            do
            {
                j = pred[j];
                printf("<-%d", j);
            } while(j != startnode);
    }
}
```

Output

```
Enter no. of vertices: 5
Enter the adjacency matrix:
0   10  0  30  100
10  0   50 0   0
0   50  0  20  10
30  0   20 0   60
100 0   0  60  0
Enter the starting node: 0
Distance to node1 = 10
Path = 1<-0
Distance to node2 = 50
Path = 2<-3<-0
Distance to node3 = 30
Path = 3<-0
Distance to node4 = 60
Path = 4<-2<-3<-0
```

Result

Thus Dijkstra's algorithm is used to find shortest path from a given vertex.

Ex. No. 10 **Prim's Algorithm**
Date:

Aim

To find the minimum cost spanning tree for the given graph using Prim's algorithm.

Algorithm

1. **Begin**
2. **Create edge list of given graph, with their weights.**
3. **Draw all nodes to create skeleton for spanning tree.**
4. **Select an edge with lowest weight and add it to skeleton and delete edge from edge list.**
5. **Add other edges. While adding an edge take care that the one end of the edge should always be in the skeleton tree and its cost should be minimum.**
6. **Repeat step 5 until n-1 edges are added.**
7. **Return.**

Program

```
/* Prim's Algorithm */

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()
{
int i,j,total_cost;
clrscr();
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
total_cost=prims();
printf("\nspanning tree matrix:\n");
for(i=0;i<n;i++)
{
printf("\n");
for(j=0;j<n;j++)
printf("%d\t",spanning[i][j]);
}
printf("\n\nTotal cost of spanning tree=%d",total_cost);
return 0;
}

int prims()
{
int cost[MAX][MAX];
int u,v,min_distance,distance[MAX],from[MAX];
int visited[MAX],no_of_edges,i,min_cost,j;
//create cost[][] matrix,spanning[][]
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(G[i][j]==0)
cost[i][j]=infinity;
else
cost[i][j]=G[i][j];
spanning[i][j]=0;
```

```
}
//initialise visited[],distance[] and from[]
distance[0]=0;
visited[0]=1;
for(i=1;i<n;i++)
{
distance[i]=cost[0][i];
from[i]=0;
visited[i]=0;
}
min_cost=0; //cost of spanning tree
no_of_edges=n-1; //no. of edges to be added
while(no_of_edges>0)
{
//find the vertex at minimum distance from the tree
min_distance=infinity;
for(i=1;i<n;i++)
if(visited[i]==0&&distance[i]<min_distance)
{
v=i;
min_distance=distance[i];
}
u=from[v];
//insert the edge in spanning tree
spanning[u][v]=distance[v];
spanning[v][u]=distance[v];
no_of_edges--;
visited[v]=1;
//updated the distance[] array
for(i=1;i<n;i++)
if(visited[i]==0&&cost[i][v]<distance[i])
{
distance[i]=cost[i][v];
from[i]=v;
}
min_cost=min_cost+cost[u][v];
}
return(min_cost);
}
```

Output

```
Enter no. of vertices: 6
Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
spanning tree matrix:
0 3 1 0 0 0
3 0 0 0 3 0
1 0 0 0 0 4
0 0 0 0 0 2
0 3 0 0 0 0
0 0 4 2 0 0
Total cost of spanning tree=13
```

Result

Thus Prim′s algorithm is used to find the minimum spanning tree of given graph.

Ex. No. 11a **Linear Search**
Date:

Aim

To perform linear search of an element on the given array.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i, $i = 0,1,2,\ldots n-1$**
4. **Read *search* value**
5. **Assign 0 to *found***
6. **Check each array element against *search***
   If A_i = *search* then
      ***found* = 1**
      Print "Element found"
      Print position *i*
      Stop
7. **If *found* = 0 then**
   print "Element not found"
8. **Stop**

Program

```
/* Linear search on a sorted array */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[50],i, n, val, found;
   clrscr();

   printf("Enter number of elements : ");
   scanf("%d", &n);
   printf("Enter Array Elements : \n");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);

   printf("Enter element to locate : ");
   scanf("%d", &val);
   found = 0;
   for(i=0; i<n; i++)
   {
      if (a[i] == val)
      {
         printf("Element found at position %d", i);
         found = 1;
         break;
      }
   }
   if (found == 0)
      printf("\n Element not found");
   getch();
}
```

Output

```
Enter number of elements : 7
Enter Array Elements :
23  6  12  5  0  32  10
Enter element to locate : 5
Element found at position 3
```

Result

Thus an array was linearly searched for an element's existence.

Ex. No. 11b **Binary Search**
Date:

Aim

To locate an element in a sorted array using Binary search method

Algorithm

1. **Start**
2. **Read number of array elements, say *n***
3. **Create an array *arr* consisting *n* sorted elements**
4. **Get element, say *key* to be located**
5. **Assign 0 to *lower* and *n* to *upper***
6. **While (*lower* < *upper*)**

   Determine middle element *mid* = (upper+lower)/2
   If key = arr[mid] then
   Print mid
   Stop
   Else if key > arr[mid] then
   lower = mid + 1
   else
   upper = mid – 1

7. **Print "Element not found"**
8. **Stop**

Program

```
/* Binary Search on a sorted array */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[50],i, n, upper, lower, mid, val, found;
   clrscr();

   printf("Enter array size  : ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
      a[i] = 2 * i;

   printf("\n Elements in Sorted Order \n");
   for(i=0; i<n; i++)
      printf("%4d", a[i]);

   printf("\n Enter element to locate : ");
   scanf("%d", &val);
   upper = n;
   lower = 0;
   found = -1;
   while (lower <= upper)
   {
      mid = (upper + lower)/2;
      if (a[mid] == val)
      {
         printf("Located at position %d", mid);
         found = 1;
         break;
      }
      else if(a[mid] > val)
         upper = mid - 1;
      else
         lower = mid + 1;
   }

   if (found == -1)
      printf("Element not found");
   getch();
}
```

Output

```
Enter array size : 9
 Elements in Sorted Order
   0   2   4   6   8  10  12  14  16
 Enter element to locate : 12
 Located at position 6



Enter array size : 10
 Elements in Sorted Order
   0   2   4   6   8  10  12  14  16  18
 Enter element to locate : 13
 Element not found
```

Result

Thus an element is located quickly using binary search method.

Ex. No. 12a **Insertion Sort**
Date:

Aim

To sort an array of N numbers using Insertion sort.

Algorithm

1. **Start**
2. ***2.*** **Read number of array elements *n***
3. **Read array elements A_i**
4. **Sort the elements using insertion sort**
   In pass p, move the element in position p left until its correct place is found among the first p + 1 elements.
   Element at position p is saved in temp, and all larger elements (prior to position p) are moved one spot to the right. Then temp is placed in the correct spot.
5. **Stop**

Program

```
/* Insertion Sort */

main()
{
   int i, j, k, n, temp, a[20], p=0;

   printf("Enter total elements: ");
   scanf("%d",&n);
   printf("Enter array elements: ");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);

   for(i=1; i<n; i++)
   {
      temp = a[i];
      j = i - 1;
      while((temp<a[j]) && (j>=0))
      {
         a[j+1] = a[j];
         j = j - 1;
      }
      a[j+1] = temp;

      p++;
      printf("\n After Pass %d: ", p);
      for(k=0; k<n; k++)
         printf("  %d", a[k]);
   }

   printf("\n Sorted List : ");
   for(i=0; i<n; i++)
      printf(" %d", a[i]);
}
```

Output

```
Enter total elements: 6
Enter array elements: 34 8 64 51 32 21
 After Pass 1:   8  34  64  51  32  21
 After Pass 2:   8  34  64  51  32  21
 After Pass 3:   8  34  51  64  32  21
 After Pass 4:   8  32  34  51  64  21
 After Pass 5:   8  21  32  34  51  64
 Sorted List :  8 21 32 34 51 64
```

Result

Thus array elements was sorted using insertion sort.

Ex. No. 13 **Merge Sort**
Date:

Aim

To sort an array of N numbers using Merge sort.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i**
4. **Divide the array into sub-arrays with a set of elements**
5. **Recursively sort the sub-arrays**
6. **Merge the sorted sub-arrays onto a single sorted array.**
7. **Stop**

Program

```
/* Merge sort */

#include <stdio.h>
#include <conio.h>

void merge(int [],int ,int ,int );
void part(int [],int ,int );
int size;

main()
{
   int i, arr[30];
   printf("Enter total no. of elements : ");
   scanf("%d", &size);
   printf("Enter array elements : ");
   for(i=0; i<size; i++)
      scanf("%d", &arr[i]);
   part(arr, 0, size-1);
   printf("\n Merge sorted list : ");
   for(i=0; i<size; i++)
      printf("%d ",arr[i]);
   getch();
}

void part(int arr[], int min, int max)
{
   int i, mid;
   if(min < max)
   {
      mid = (min + max) / 2;
      part(arr, min, mid);
      part(arr, mid+1, max);
      merge(arr, min, mid, max);
   }
   if (max-min == (size/2)-1)
   {
      printf("\n Half sorted list : ");
      for(i=min; i<=max; i++)
         printf("%d ", arr[i]);
   }
}

void merge(int arr[],int min,int mid,int max)
{
   int tmp[30];
   int i, j, k, m;
   j = min;
   m = mid + 1;
```

```
    for(i=min; j<=mid && m<=max; i++)
    {
       if(arr[j] <= arr[m])
       {
          tmp[i] = arr[j];
          j++;
       }
       else
       {
          tmp[i] = arr[m];
          m++;
       }
    }
    if(j > mid)
    {
       for(k=m; k<=max; k++)
       {
          tmp[i] = arr[k];
          i++;
       }
    }
    else
    {
       for(k=j; k<=mid; k++)
       {
          tmp[i] = arr[k];
          i++;
       }
    }
    for(k=min; k<=max; k++)
       arr[k] = tmp[k];
}
```

Output

```
Enter total no. of elements : 8
Enter array elements : 24 13 26 1 2 27 38 15

 Half sorted list : 1 13 24 26
 Half sorted list : 2 15 27 38
 Merge sorted list : 1 2 13 15 24 26 27 38
```

Result

Thus array elements was sorted using merge sort's divide and conquer method.

Ex. No. 12 **Open Addressing Hashing Technique**
Date:

Aim

To implement hash table using a C program.

Algorithm

1. **Create a structure, data (hash table item) with key and value as data.**
2. **Now create an array of structure, data of some certain size (10, in this case). But, the size of array must be immediately updated to a prime number just greater than initial array capacity (i.e 10, in this case).**
3. **A menu is displayed on the screen.**
4. **User must choose one option from four choices given in the menu**
5. **Perform all the operations**
6. **Stop**

Program

```
/* Open hashing */

#include <stdio.h>
#include <stdlib.h>

#define MAX 10

main()
{
   int a[MAX], num, key, i;
   char ans;
   int create(int);
   void linearprobing(int[], int, int);
   void display(int[]);

   printf("\nCollision handling by linear probing\n\n");
   for(i=0; i<MAX; i++)
      a[i] = -1;
   do
   {
      printf("\n Enter number:");
      scanf("%d", &num);
      key = create(num);
      linearprobing(a, key, num);
      printf("\nwish to continue?(y/n):");
      ans = getch();
   } while( ans == 'y');
   display(a);
}

int create(int num)
{
   int key;
   key = num % 10;
   return key;
}

void linearprobing(int a[MAX], int key, int num)
{
   int flag, i, count = 0;
   void display(int a[]);
   flag = 0;
   if(a[key] == -1)
      a[key] = num;
   else
   {
      i=0;
```

```
        while(i < MAX)
        {
           if(a[i] != -1)
           count++;
           i++;
        }
        if(count == MAX)
        {
           printf("hash table is full");
           display(a);
           getch();
           exit(1);
        }
        for(i=key+1; i<MAX; i++)
           if(a[i] == -1)
           {
              a[i] = num;
              flag = 1;
              break;
           }
        for(i=0; i<key && flag==0; i++ )
        if(a[i] == -1)
        {
           a[i] = num;
           flag = 1;
           break;
        }
    }
}

void display(int a[MAX])
{
   int i;
   printf("\n Hash table is:");
   for(i=0; i<MAX; i++)
      printf("\n  %d\t\t%d",i,a[i]);
}
```

Output

```
Collision handling by linear probing
 Enter number:1
wish to continue?(y/n):
 Enter number:26
wish to continue?(y/n):
 Enter number:62
wish to continue?(y/n):
 Enter number:93
wish to continue?(y/n):
```

```
 Enter number:84
wish to continue?(y/n):
 Enter number:15
wish to continue?(y/n):
  Enter number:76
wish to continue?(y/n):
 Enter number:98
wish to continue?(y/n):
 Enter number:26
wish to continue?(y/n):
 Enter number:199
wish to continue?(y/n):
 Enter  number:1234
wish to continue?(y/n):
 Enter number:5678
hash table is full
 Hash table is:
 0              1234
 1              1
 2              62
 3              93
 4              84
 5              15
 6              26
 7              76
 8              98
 9              199
```

Result

Thus hashing has been performed successfully.