20MCA135 – DATA STRUCTURES LAB

LABORATORY RECORD

Submitted in partial fulfilment of the requirements for the award of

Masters of Computer Applications

At

COLLEGE OF ENGINEERING POONJAR

Managed by I.H.R.D., A Govt. of Kerala undertaking

(Affiliated to APJ Abdul Kalam Technological University)

SUBMITTED BY

SHAMEER SABEER (PJR24MCA-2017)

DEPARTMENT OF COMPUTER SCIENCE

COLLEGE OF ENGINEERING POONJAR

COLLEGE OF ENGINNERING POONJAR

Managed by I.H.R.D., A Govt. of Kerala undertaking

(Affiliated to APJ Abdul Kalam Technological University)

CERTIFICATE

Certified that this is a Bonafide record of practical work done in Data Structures Lab (20MCA135) Laboratory by SHAMEER SABEER, Reg No: PJR24MCA-2017 of College of Engineering Poonjar during the academic year 2024 – 2026.

Dr. Annie Julie Joseph	Dr. Annie Julie Joseph
Head of the department	Professor of CSE

Submitted to the University examination held on:

INTERNAL EXAMINER	EXTERNAL EXAMINER

INDEX

PROGRAM NO:1

AIM:

Write a c program to implement Perform sorting operations in an array.

ALGORITHM:

1.Input the Array:

- Accept the size of the array (n) from the user.
- Input the n elements of the array.

2.Bubble Sort Logic:

- Outer Loop: Repeat the process n-1 times (i from 0 to n-2).
- Inner Loop:
- Compare adjacent elements (arr[j] and arr[j+1]) for all indices j from 0 to n-i-2.
- If arr[j] > arr[j+1], swap the two elements.
- After each pass of the inner loop, the largest unsorted element is moved to its correct position.

3.Output the Sorted Array:

- Print the array elements after the sorting process.

4.End.

CODE:

```
#include <stdio.h>

void bubbleSort(int arr[], int n)
{
   int i, j, temp;
   for(i = 0; i < n-1; i++)
    {
      for(j = 0; j < n-i-1; j++)
       {
         if(arr[j] > arr[j+1])
          {
```

```
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
          }
        }
      }
    }

int main()
 {
   int n, i;


   printf("Enter the number of elements: ");
   scanf("%d", &n);


   int arr[n];


   printf("Enter %d elements:\n", n);
   for(i = 0; i < n; i++)
    {
      scanf("%d", &arr[i]);
   }


   bubbleSort(arr, n);


   printf("Sorted array: ");
```

```
    for(i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");

}
```

OUTPUT:

```
Enter the number of elements: 5
Enter 5 elements:
23 21 12 67 87
Sorted array: 12 21 23 67 87
```

RESULT:

The program is executed successfully and the output is verified.

PROGRAM NO:2

AIM:

Write a C program to perform linear search and binary search.

ALGORITHM:

1. Input the Array and Target Value:

1.1. Prompt the user to enter the size of the array (size).

1.2. If size <= 0, print an error message and terminate the program.

1.3. Allocate an array of size size.

1.4. Prompt the user to enter the size elements of the array.

1.5. Prompt the user to enter the target value to search (target).

2. Choose the Search Type:

2.1. Prompt the user to select the search type:

1 for Linear Search

2 for Binary Search

2.2. If an invalid search type is entered, print an error message and terminate the program.

3. Handle Binary Search (if selected):

3.1. If the user selects Binary Search (searchType == 2):

Sort the array using Bubble Sort.

Print the sorted array for user reference.

4. Perform the Search:

4.1. Call the search function with the appropriate arguments:

The array (arr).

The size of the array (size).

The target value (target).

The chosen search type (searchType).

4.2. Linear Search Algorithm (if searchType == 1):

Traverse the array from index 0 to size-1.

If any element matches the target, return its index.

If no match is found after the traversal, return -1.

4.3. Binary Search Algorithm (if searchType == 2):

Initialize left = 0 and right = size-1.

Repeat until left > right:

Compute the middle index: mid = left + (right - left) / 2.

If arr[mid] == target, return mid.

If arr[mid] < target, set left = mid + 1.

Otherwise, set right = mid - 1.

If no match is found, return -1.

5. Output the Result:

5.1. If the search function returns a valid index (result != -1):

Print the index where the element was found.

5.2. If the search function returns -1:

Print that the element was not found.

6. End the Program:

CODE;

```
#include <stdio.h>

int search(int arr[], int size, int target, int searchType) {
  if (searchType == 1)
   {

     for (int i = 0; i < size; i++)
     {
        if (arr[i] == target)
```

```
            {
                return i;
            }
        }
        return -1;
    } else if (searchType == 2)
     {

        int left = 0, right = size - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == target)
            {
                return mid;
            }
            if (arr[mid] < target)
             {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return -1;
    }
    return -1;
}

void bubbleSort(int arr[], int size)
 {

    for (int i = 0; i < size - 1; i++)
    {
```

```
        for (int j = 0; j < size - i - 1; j++)
         {
           if (arr[j] > arr[j + 1])
            {

               int temp = arr[j];
               arr[j] = arr[j + 1];
               arr[j + 1] = temp;
           }
        }
    }
}

int main()
{
    int size, target, searchType;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &size);

    if (size <= 0) {
        printf("Invalid array size. Size should be greater than 0.\n");
        return 1;
    }

    int arr[size];
    printf("Enter the elements of the array: ");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
```

```
printf("Enter the target value to search for: ");
scanf("%d", &target);



printf("Enter search type (1 for Linear Search, 2 for Binary Search): ");
scanf("%d", &searchType);

if (searchType == 2)
 {

   printf("Sorting array for Binary Search...\n");
   bubbleSort(arr, size);
   printf("Sorted array: ");
   for (int i = 0; i < size; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
} else if (searchType != 1)
 {
   printf("Invalid search type. Choose 1 or 2.\n");
   return 1;
}



int result = search(arr, size, target, searchType);

if (result != -1)
 {
   printf("Element found at index: %d\n", result);
} else {
   printf("Element not found.\n");
}
```

```
}
```

OUTPUT:

```
Enter the number of elements in the array: 5
Enter the elements of the array: 21 34 54 65 11
Enter the target value to search for: 54
Enter search type (1 for Linear Search, 2 for Binary Search): 1
Element found at index: 2
```

```
Enter the number of elements in the array: 5
Enter the elements of the array: 21 34 56 76 21
Enter the target value to search for: 34
Enter search type (1 for Linear Search, 2 for Binary Search): 2
Sorting array for Binary Search...
Sorted array: 21 21 34 56 76
Element found at index: 2
```

RESULT:

The program is executed successfully and the output is verified.

PROGRAM NO: 3

AIM:

Write a C program to implement Singly linked list with functions Insertion, Deletion and Display.

ALGORITHM:

INSERTION AT END

1. Start
2. Input the data to be inserted
3. Create a new node
4. if(head==NULL) head=newnode;
5. else
   while(temp->next!=NULL) temp=temp->next temp->next=newnode; newnode->next=NULL;
   ++count;
6. Stop

DELETION AT END

1. Start
2. if(head==NULL) print invalid
3.else

   while(temp->next->next!=NULL) temp=temp->next;

   free(temp->next);

   temp->next=NULL;

   --count;

4. Stop

DISPLAY

1. if(temp==NULL) print Empty Linked list
2. else

   while(temp!=NULL)

   printf("%d-->",temp->data);

   temp=temp->next;

CODE:

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

struct node
{
    int data;
    struct node *link;
};

struct node *head = NULL;

void insert_at_end(int n)
{
    struct node *temp, *t;
    t = (struct node *)malloc(sizeof(struct node));
    t->data = n;

    if (head == NULL)
    {
        head = t;
        head->link = NULL;
    }
    else
    {
        temp = head;
        while (temp->link != NULL)
        {
            temp = temp->link;
        }
        temp->link = t;
        t->link = NULL;
```

```
    }
}
void delete(int value)
{
    struct node *temp = head, *prev = NULL;

    if (temp == NULL)
    {
        printf("\nList is empty. Cannot delete.\n");
        return;
    }

    if (temp != NULL && temp->data == value)
    {
        head = temp->link;
        free(temp);
        printf("Deleted node with value %d\n", value);
        return;
    }

    while (temp != NULL && temp->data != value)
    {
        prev = temp;
        temp = temp->link;
    }

    if (temp == NULL)
    {
        printf("Value %d not found in the list.\n", value);
        return;
    }
```

```
    prev->link = temp->link;
    free(temp);
    printf("Deleted node with value %d\n", value);
}

void display()
{
    struct node *t = head;
    if (t == NULL)
    {
        printf("List is empty\n");
        return;
    }
    while (t != NULL)
    {
        printf("%d -> ", t->data);
        t = t->link;
    }
    printf("NULL\n");
}

void main()
{
    int d, data;
    for (;;)
    {
        printf("Enter 1 to insert, 2 to display, 3 to  delete, 4 to Exit: ");
        scanf("%d", &d);

        if (d == 1)
        {
            printf("Enter the data to insert: ");
```

```
            scanf("%d", &data);
            insert_at_end(data);
        }
        else if (d == 2)
        {
            display();
        }
        else if (d == 3)
        {
            printf("Enter the value to delete: ");
            scanf("%d", &data);
            delete (data);
        }
        else if (d == 4)
        {
            printf("Exiting");
            break;
        }
        else
        {
            printf("Invalid option. Please try again.\n");
        }
    }
}
```

OUTPUT:

```
Enter 1 to insert, 2 to display, 3 to  delete, 4 to Exit: 1
Enter the data to insert: 23
Enter 1 to insert, 2 to display, 3 to  delete, 4 to Exit: 1
Enter the data to insert: 56
Enter 1 to insert, 2 to display, 3 to  delete, 4 to Exit: 1
Enter the data to insert: 67
Enter 1 to insert, 2 to display, 3 to  delete, 4 to Exit: 2
23 -> 56 -> 67 -> NULL
Enter 1 to insert, 2 to display, 3 to  delete, 4 to Exit: 3
Enter the value to delete: 56
Deleted node with value 56
Enter 1 to insert, 2 to display, 3 to  delete, 4 to Exit: 2
23 -> 67 -> NULL
Enter 1 to insert, 2 to display, 3 to  delete, 4 to Exit: 4
Exiting
```

RESULT:

The program is executed successfully and the output is verified

PROGRAM NO:4

AIM:

Write a c program to implement Singly linked stack-push, pop and display.

ALGORITHM;

1.Initialize the Stack:

a)Set top pointer to NULL, indicating the stack is empty.

2.Push Operation:

a)Create a new node with the given value.

b)If memory allocation fails, print an overflow message and exit.

c)Set the new node's next pointer to the current top.

d)Update the top pointer to point to the new node.

e)Print a success message.

3.Pop Operation:

a)Check if the stack is empty (top == NULL).

b)If empty, print an underflow message and return -1.

c)Save the value of the top node to a variable.

d)Update top to point to the next node.

e)Free the memory of the popped node.

f)Return the popped value.

4.Display Operation:

a) If the stack is empty, print a message and exit.

b)Traverse the stack using a temporary pointer.

c) Print the data of each node.

Main Menu:

Use a loop to allow the user to:

a)Push values onto the stack.

b)Pop values from the stack.

c)Display all stack elements.

d) Exit the program when the user selects "4".

e) This algorithm efficiently handles the dynamic nature of stacks using a linked list structure.

CODE:

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

struct Node
{
   int data;
   struct Node* next;
};

void push(struct Node** top, int value);
int pop(struct Node** top);
void display(struct Node* top);

int main()
{
   struct Node* top = NULL;
   int choice, value;

   do
   {
      printf("\nStack Operations:\n");
      printf("1. Push\n");
      printf("2. Pop\n");
      printf("3. Display\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
```

```
        switch (choice)
        {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &value);
                push(&top, value);
                break;
            case 2:
                value = pop(&top);
                if (value != -1)
                    printf("Popped value: %d\n", value);
                break;
            case 3:
                display(top);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);



}

void push(struct Node** top, int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL)
    {
```

```
        printf("Stack overflow! Memory allocation failed.\n");
        return;
    }
    newNode->data = value;
    newNode->next = *top;
    *top = newNode;
    printf("Value pushed: %d\n", value);
}

int pop(struct Node** top)
{
    if (*top == NULL)
    {
        printf("Stack underflow! The stack is empty.\n");
        return -1;
    }
    struct Node* temp = *top;
    int poppedValue = temp->data;
    *top = temp->next;
    free(temp);
    return poppedValue;
}
void display(struct Node* top)
{
    if (top == NULL)
    {
        printf("The stack is empty.\n");
        return;
    }
    printf("Stack elements:\n");
    struct Node* temp = top;
    while (temp != NULL)
```

```
    {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}
```

OUTPUT:

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 20
Value pushed: 20
```

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 30
Value pushed: 30
```

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 10
Value pushed: 10
```

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements:
10
30
20
```

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped value: 10
```

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements:
30
20
```

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
Exiting...
PS C:\Users\sajin\OneDrive\Desktop\Project\Python\cprogram>
```

RESULT:

The Program is executed successfully and the output is verified.

PROGRAM NO: 5

AIM:

Doubly linked list-Insertion, Deletion and display.

ALGORITHM:

INSERTION AT BEGINNING

1. Start
2. Input the data to be inserted
3. Create a new node
4. If (head==NULL) , head=newnode; , newnode->next=NULL; , newnode->pre=NULL;
5. Else, newnode->next=head; , head=newnode; , newnode->pre=NULL; ++count
6. Stop

INSERTION AT END

1. Start
2. Input the data to be inserted
3. Create a new node
4. if(head==NULL) head=newnode;
5. else

while(temp->next!=NULL)

temp=temp->next

newnode->pre=temp;

temp->next=newnode;

newnode->next=NULL;

++count

6. Stop

INSERTION AT POSITION

1. Start
2. input the data and pos
3. initialize temp = start;
4. If (head==NULL) Print list is empty
5. else if(head==NULL) head=newnode;
6. else if(pos==1) in_beg();
7. else if(count+1==pos) in_end();
8. Else

Print enter the data

for(int i=2;inext;

newnode->pre= temp newnode->next=temp->next;

temp->next->pre=newnode;

temp->next =newnode;

++count;

DELETION AT BEGINNING

1. Start
2. if(count==0) print Empty linked list
3. else head=head->next; free(temp); --count;
4. Stop

DELETION AT END

1. Start 2. if(head==NULL) print invalid

3. else while(temp->next->next!=NULL) temp=temp->next; free(temp->next); temp->next=NULL; --count;

4. Stop

DELETION AT POSITION

1. Start
2. if(head==NULL) print invalid
3. else if(pos==0) del_in()
4. else if(count==pos) del_end()
5. else for(int i=2;inext;

ptr=temp->next->next;

node *del=temp->next;

free(del); ptr->pre=temp; temp->next=ptr;

--count;

6. Stop

DISPLAY

1. if(temp==NULL) print Empty Linked list
2. else while(temp!=NULL) printf("%d-->",temp->data);

temp=temp->next;

<u>Code:</u>

```
#include<stdio.h>

#include<stdlib.h>


typedef struct node {

   int data;

   struct node* next;

   struct node* pre;

} node;
```

```
node* head = NULL;
int count = 0;

void display() {
    node* temp = head;
    if (temp == NULL) {
        printf("Empty Linked List\n");
    } else {
        while (temp != NULL) {
            printf("%d-->", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

void in_beg() {
    node* newnode = (node*)malloc(sizeof(node));
    printf("\nEnter the data: ");
    scanf("%d", &newnode->data);
    newnode->next = head;
    newnode->pre = NULL;

    if (head != NULL) {
        head->pre = newnode;
    }
    head = newnode;
    printf("\nNode inserted at the beginning..\n");
    count++;
}

void in_end() {
```

```
    node* newnode = (node*)malloc(sizeof(node));
    node* temp = head;
    printf("\nEnter the data: ");
    scanf("%d", &newnode->data);
    newnode->next = NULL;

    if (head == NULL) {
        newnode->pre = NULL;
        head = newnode;
    } else {
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newnode;
        newnode->pre = temp;
    }
    printf("\nNode inserted at the end..\n");
    count++;
}

void in_pos() {
    int pos;
    node* newnode = (node*)malloc(sizeof(node));
    node* temp = head;

    printf("Enter the position at which the data is to be inserted: ");
    scanf("%d", &pos);
    if (pos <= 0 || pos > count + 1) {
        printf("Invalid position\n");
        return;
    }
```

```
    printf("\nEnter the data: ");
    scanf("%d", &newnode->data);

    if (pos == 1) {
        newnode->next = head;
        newnode->pre = NULL;
        if (head != NULL) {
            head->pre = newnode;
        }
        head = newnode;
    } else {
        for (int i = 1; i < pos - 1; i++) {
            temp = temp->next;
        }
        newnode->next = temp->next;
        newnode->pre = temp;
        if (temp->next != NULL) {
            temp->next->pre = newnode;
        }
        temp->next = newnode;
    }
    printf("\nNode inserted at position %d..\n", pos);
    count++;
}

void del_in() {
    if (head == NULL) {
        printf("Empty Linked List\n");
        return;
    }
    node* temp = head;
    head = head->next;
```

```
    if (head != NULL) {
        head->pre = NULL;
    }
    free(temp);
    printf("\nNode deleted from the beginning..\n");
    count--;
}

void del_end() {
    if (head == NULL) {
        printf("Empty Linked List\n");
        return;
    }
    node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    if (temp->pre != NULL) {
        temp->pre->next = NULL;
    } else {
        head = NULL;
    }
    free(temp);
    printf("\nNode deleted from the end..\n");
    count--;
}

void del_pos() {
    int pos;
    printf("Enter the position at which the node has to be deleted: ");
    scanf("%d", &pos);
```

```
    if (pos <= 0 || pos > count) {
        printf("Invalid position\n");
        return;
    }

    node* temp = head;
    if (pos == 1) {
        del_in();
    } else {
        for (int i = 1; i < pos; i++) {
            temp = temp->next;
        }
        if (temp->pre != NULL) {
            temp->pre->next = temp->next;
        }
        if (temp->next != NULL) {
            temp->next->pre = temp->pre;
        }
        free(temp);
        printf("\nNode deleted from position %d..\n", pos);
        count--;
    }
}

int main() {
    int ch;
    do {
        printf("\n*****LINKED LIST*****");
        printf("\n1.Display");
        printf("\n2.Insert at beginning");
        printf("\n3.Insert at end");
        printf("\n4.Insert at a position");
```

```
printf("\n5.Deletion at beginning");
printf("\n6.Deletion from end");
printf("\n7.Deletion from a position");
printf("\n8.Exit");
printf("\n\nEnter your choice: ");
scanf("%d", &ch);

switch(ch) {
    case 1:
        display();
        break;
    case 2:
        in_beg();
        break;
    case 3:
        in_end();
        break;
    case 4:
        in_pos();
        break;
    case 5:
        del_in();
        break;
    case 6:
        del_end();
        break;
    case 7:
        del_pos();
        break;
    case 8:
        printf("Exiting program.\n");
        break;
```

```
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(ch != 8);
    return 0;
}
```

Output:

```
*****LINKED LIST*****
1.Display
2.Insert at beginning
3.Insert at end
4.Insert at a position
5.Deletion at beginning
6.Deletion from end
7.Deletion from a position
8.Exit

Enter your choice: 2

Enter the data: 4

Node inserted at the beginning..
```

```
Enter your choice: 2

Enter the data: 2

Node inserted at the beginning..
```

```
Enter your choice: 2

Enter the data: 1

Node inserted at the beginning..
```

```
Enter your choice: 1
1-->2-->4-->NULL
```

```
Enter your choice: 3

Enter the data: 5

Node inserted at the end..
```

```
Enter your choice: 1
1-->2-->4-->5-->NULL
```

```
Enter your choice: 4
Enter the position at which the data is to be inserted: 3

Enter the data: 3

Node inserted at position 3..
```

```
Enter your choice: 1
1-->2-->3-->4-->5-->NULL
```

```
Enter your choice: 5

Node deleted from the beginning..
```

```
Enter your choice: 1
2-->3-->4-->5-->NULL
```

```
Enter your choice: 6

Node deleted from the end..
```

```
Enter your choice: 1
2-->3-->4-->NULL
```

```
Enter your choice: 7
Enter the position at which the node has to be deleted: 2

Node deleted from position 2..
```

```
Enter your choice: 1
2-->4-->NULL
```

RESULT:

The program is executed successfully and the output is verified.

PROGRAM NO: 6

AIM:

Write a c Program to implement set data structure and set operations using bit strings.

ALGORITHM:

1. Input Universal Set:
   a) Read the size n of the universal set.
   b) Input n elements into the universal set array u.
2. Input Set A:
   a) Read the size m of set A and ensure it does not exceed n.
   b) Input m elements into set A.
3. Input Set B:
   a) Read the size o of set B and ensure it does not exceed n.
   b) Input o elements into set B.
4. Generate Bit Strings
   a) Create bit string bitA for set A:
      - Mark 1 in bitA if an element in the universal set is present in set A; otherwise, mark 0.
   b) Create bit string bitB for set B:
      - Mark 1 in bitB if an element in the universal set is present in set B; otherwise, mark 0.
5. Display Bit Strings:
   a) Print the bit string representation of sets A and B.
6. Perform Set Operations:
   a) Display a menu with the following operations:
      1. Union:
         - Compute and print the union of bitA and bitB using logical OR.
      2. Intersection:
         - Compute and print the intersection of bitA and bitB using logical AND.
      3. Difference
         - Compute and print the difference of bitA and bitB using logical AND with NOT.
      4. Exit:
7. Repeat Menu Until Exit.
8. Exit Program.

CODE:

```
#include <stdio.h>
void Union(int x[], int y[], int n)
{
   int z[50], i;
   printf("\nThe Union of both bit strings of Set A and B are: ");
   for (i = 0; i < n; i++)
      z[i] = x[i] == 1 || y[i] == 1;
   for (i = 0; i < n; i++)
      printf("%d ", z[i]);
}
void Intersection(int x[], int y[], int n)
{
   int z[50], i;
   printf("\nThe Intersection of both bit strings of Set A and B are: ");
   for (i = 0; i < n; i++)
      z[i] = x[i] == 1 && y[i] == 1;
   for (i = 0; i < n; i++)
      printf("%d ", z[i]);
}
void Difference(int x[], int y[], int n)
{
   int z[50], i;
   printf("\nThe difference of both bit strings of set A and B are: ");
   for (i = 0; i < n; i++)
      z[i] = x[i] == 1 && y[i] == 0;
   for (i = 0; i < n; i++)
      printf("%d ", z[i]);
}

void main()
{
```

```
    int n, m, o, i, j, u[20], a[20], b[20], bitA[50], bitB[50], ch;
    printf("Enter the size of the Universal set: ");
    scanf("%d", &n);
    printf("Enter the elements of the Universal set: ");
    for (i = 0; i < n; i++)
        scanf("%d", &u[i]);
    printf("the Elements in the Universal set  are:");
    for (i = 0; i < n; i++)
        printf(" %d ", u[i]);
    printf("\nEnter the size of set A: ");
    scanf("%d", &m);
    while (n < m)
    {
        printf("The size entered is bigger than the Universal set's size. Please Enter the size of
set A again.");
        scanf("%d", &m);
    }
    printf("Enter the elements of Set A: ");
    for (i = 0; i < m; i++)
        scanf("%d", &a[i]);
    printf("The set A is :");
    for (i = 0; i < m; i++)
        printf("%d ", a[i]);
    printf("\nEnter the size of set B: ");
    scanf("%d", &o);
    while (n < o)
    {
        printf("The size entered is bigger than the Universal set's size. Please Enter the size of
set B again.");
        scanf("%d", &o);
    }
    printf("Enter the elements of Set B: ");
    for (i = 0; i < o; i++)
```

```
    scanf("%d", &b[i]);
printf("The Set B is: ");
for (i = 0; i < o; i++)
    printf("%d ", b[i]);
for (i = 0; i < n; i++)
{
    for (j = 0; j < m; j++)
    {
        if (u[i] == a[j])
        {
            bitA[i] = 1;
            break;
        }
        else
            bitA[i] = 0;
    }
}
for (i = 0; i < n; i++)
{
    for (j = 0; j < o; j++)
    {
        if (u[i] == b[j])
        {

            bitB[i] = 1;
            break;
        }

        else
            bitB[i] = 0;
    }
}
```

```
    printf("\nBit String A : ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", bitA[i]);
    }
    printf("\nBit String B : ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", bitB[i]);
    }
    do
    {
        printf("\nEnter the operation to be done:\n (1) For Union\n (2) For Intersection\n (3)
For Difference\n (4) To Exit ");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            Union(bitA, bitB, n);
            break;
        case 2:
            Intersection(bitA, bitB, n);
            break;
        case 3:
            Difference(bitA, bitB, n);
            break;
        case 4:
            printf("\nExiting");
            break;
        default:
            printf("\nInvalid choice!!");
        }
```

```
    } while (ch != 4);
}
```

OUTPUT:

```
Enter the size of the Universal set: 8
Enter the elements of the Universal set: 1 2 3 4 5 6 7 8
the Elements in the Universal set  are: 1  2  3  4  5  6  7  8
Enter the size of set A: 5
Enter the elements of Set A: 1 2 3 4 5
The set A is :1 2 3 4 5
Enter the size of set B: 4
Enter the elements of Set B: 3 4 5 6
The Set B is: 3 4 5 6
Bit String A : 1 1 1 1 1 0 0 0
Bit String B : 0 0 1 1 1 1 0 0
Enter the operation to be done:
```

```
 (1) For Union
 (2) For Intersection
 (3) For Difference
 (4) To Exit
Enter your choice: 1

The Union of both bit strings of Set A and B are: 1 1 1 1 1 1 0 0
```

```
Enter the operation to be done:
 (1) For Union
 (2) For Intersection
 (3) For Difference
 (4) To Exit
Enter your choice: 2

The Intersection of both bit strings of Set A and B are: 0 0 1 1 1 0 0 0
```

```
Enter the operation to be done:
 (1) For Union
 (2) For Intersection
 (3) For Difference
 (4) To Exit
Enter your choice: 3

The difference of both bit strings of set A and B are: 1 1 0 0 0 0 0 0
```

RESULT:

The program is executed successfully and the output is verified.

PROGRAM NO:7

AIM:

Disjoint sets and associated operations.

ALGORITHM:

1. Define Data Structures:
   - Define a structure node with rep, next, and data.
   - Declare arrays heads and tails to store pointers to disjoint sets.
2. Global Variables:
   - Declare countRoot to track the number of disjoint sets.
3. Function makeSet(x):
   - Allocate memory for a new node.
   - Initialize node fields and store it in arrays.
   - Increment countRoot.
4. Function find(a):
   - Iterate through sets to find the representative of element a.
5. Function unionSets(a, b):
   - Find representatives of elements a and b.
   - If representatives are different, merge sets.
6. Function search(x):
   - Check if element x is present in any set.
7. Main Program:
   - Display a menu for set operations.
   - Loop until the user chooses to exit.
   - Call corresponding functions based on user input.

CODES:

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node *rep;

struct node *next;

int data;

}*heads[50],*tails[50];

static int countRoot=0;

void makeSet(int x)
```

```
{
struct node *new=(struct node *)malloc(sizeof(struct node));
new->rep=new;
new->next=NULL;
new->data=x;
heads[countRoot]=new;
tails[countRoot++]=new;
}
struct node* find(int a)
{
int i;
struct node *tmp=(struct node *)malloc(sizeof(struct node));
for(i=0;i<countRoot;i++)
{
tmp=heads[i];
while(tmp!=NULL)
{
if(tmp->data==a)
return tmp->rep;
tmp=tmp->next;
}
}
return NULL;
}
void unionSets(int a,int b)
{
int i,pos,flag=0,j;
struct node *tail2=(struct node *)malloc(sizeof(struct node));
struct node *rep1=find(a);
struct node *rep2=find(b);
if(rep1==NULL||rep2==NULL)
{
```

```
printf("\nElement not present in the DS\n");
return;
}
if(rep1!=rep2)
{
for(j=0;j<countRoot;j++)
{
if(heads[j]==rep2)
{
pos=j;
flag=1;
countRoot-=1;
tail2=tails[j];
for(i=pos;i<countRoot;i++)
{
heads[i]=heads[i+1];
tails[i]=tails[i+1];
}
}
if(flag==1)
break;
}
for(j=0;j<countRoot;j++)
{
if(heads[j]==rep1)
{
tails[j]->next=rep2;
tails[j]=tail2;
break;
}
}
while(rep2!=NULL)
```

```
{
rep2->rep=rep1;
rep2=rep2->next;
}
}
}
int search(int x)
{
int i;
struct node *tmp=(struct node *)malloc(sizeof(struct node));
for(i=0;i<countRoot;i++)
{
tmp=heads[i];
if(heads[i]->data==x)
return 1;
while(tmp!=NULL)
{
if(tmp->data==x)
return 1;
tmp=tmp->next;
}
}
return 0;
}
void main()
{
int choice,x,i,j,y,flag=0;
do
{
printf("\n **** Disjoint set ****");
printf("\n1.Make Set");
printf("\n2.Display set representatives");
```

```
printf("\n3.Union");
printf("\n4.Find Set");
printf("\n5.Display sets");
printf("\n6.Exit\n");
printf("\nEnter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("\nEnter new element : ");
scanf("%d",&x);
if(search(x)==1)
printf("\nElement already present in the disjoint set\n");
else
makeSet(x);
break;
case 2:
printf("\n");
for(i=0;i<countRoot;i++)
printf("%d ",heads[i]->data);
printf("\n");
break;
case 3:
printf("\nEnter an element in first set you want to union : ");
scanf("%d",&x);
printf("\nEnter an element in the second set you want to union : ");
scanf("%d",&y);
unionSets(x,y);
break;
case 4:printf("\nEnter the element");
scanf("%d",&x);
struct node *rep=(struct node *)malloc(sizeof(struct node));
rep=find(x);
```

```
if(rep==NULL)
printf("\nElement not present in the Set\n");
else
printf("\nThe representative of %d is %d\n",x,rep->data);
break;
case 5:
for (i = 0; i <countRoot; i++)
{
printf("\nSet %d: ", i + 1);
struct node *temp =heads[i];
printf("{ ");
while (temp != NULL)
{
printf("%d ", temp->data);
temp = temp->next;
}
printf("}\n");
}
break;
case 6:exit(0);
default: printf("\nWrong choice\n");
break;
}
}while(1);
}
```

OUTPUT:

```
 **** Disjoint set ****
1.Make Set
2.Display set representatives
3.Union
4.Find Set
5.Display sets
6.Exit

Enter your choice : 1

Enter new element : 4
```

```
Enter your choice : 1

Enter new element : 5
```

```
Enter your choice : 2

4 5
```

```
Enter your choice : 3

Enter an element in first set you want to union : 4

Enter an element in the second set you want to union : 5
```

```
Enter your choice : 4

Enter the element5

The representative of 5 is 4
```

```
Enter your choice : 5

Set 1: { 4 5 }
```

RESULT:

The program is executed successfully and the output is verified.

PROGRAM NO:8

AIM:

Write a c program to implement Graph using Adjacency matrix.

ALGORITHM:

- **Create Graph:**

    1.Input number of nodes (n).

    2.Initialize adjacency matrix (adj) to 0.

    3.For each edge, input origin and destin until "0 0":

    4. Validate nodes, then set adj[origin][destin] = 1.

- **Insert Node:**

    1.Increment n.

    2.Add a new row and column to adj and initialize to 0.

- **Delete Node:**

    1.Input node u to delete.

    2.Validate u. If valid:Shift rows and columns in adj to remove the node.

    4.Clear the last row and column, then decrement n.

- **Insert Edge:**

    1.Input origin and destin.

    2.Validate nodes, then set adj[origin][destin] = 1.

- **Display Graph:**

    Print the adjacency matrix.

- **Main Menu:**

    Loop until exit:

        1.Choose operation: Insert node, delete node, insert edge, display, or exit.

2.Call respective function based on the choice.

CODE:

```
#include <stdio.h>
#define max 20

int adj[max][max] = {0};
int n;

void create_graph()
{
   int i, max_edges, origin, destin;

   printf("Enter number of nodes: ");
   scanf("%d", &n);
   max_edges = n * (n - 1);

   for (i = 1; i <= max_edges; i++)
   {
      printf("Enter edge %d (0 0 to quit): ", i);
      scanf("%d %d", &origin, &destin);

      if (origin == 0 && destin == 0)
         break;

      if (origin > n || destin > n || origin <= 0 || destin <= 0)
      {
         printf("Invalid edge! Try again.\n");
         i--;
      }
      else
```

```
        {
            adj[origin][destin] = 1;
        }
    }
}

void insert_edge()
{
    int origin, destin;

    printf("Enter an edge (origin destination): ");
    scanf("%d %d", &origin, &destin);

    if (origin > n || destin > n || origin <= 0 || destin <= 0)
    {
        printf("Invalid edge! Nodes should be between 1 and %d.\n", n);
        return;
    }

    adj[origin][destin] = 1;
    printf("Edge (%d -> %d) added successfully.\n", origin, destin);
}

void display()
{
    int i, j;
    printf("\nAdjacency Matrix:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            printf("%4d", adj[i][j]);
```

```
        }
        printf("\n");
    }
}

void insert_node()
{
    int i;
    n++;
    printf("The inserted node is %d\n", n);

    for (i = 1; i <= n; i++)
    {
        adj[i][n] = 0;
        adj[n][i] = 0;
    }
}

void delete_node(int u)
{
    int i, j;

    if (n == 0)
    {
        printf("Graph is empty!\n");
        return;
    }

    if (u > n || u <= 0)
    {
        printf("Node %d is not present in the graph.\n", u);
        return;
```

```
    }

    // Shift rows and columns to remove the node
    for (i = u; i < n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            adj[j][i] = adj[j][i + 1];
            adj[i][j] = adj[i + 1][j];
        }
    }

    // Clear the last row and column
    for (i = 1; i <= n; i++)
    {
        adj[i][n] = 0;
        adj[n][i] = 0;
    }

    n--;
    printf("Node %d deleted successfully.\n", u);
}

int main()
{
    int choice, node;

    create_graph();
    while (1)
    {
        printf("\n1. Insert a node\n");
        printf("2. Delete a node\n");
```

```
        printf("3. Insert an edge\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            insert_node();
            break;
        case 2:
            printf("Enter the node to be deleted: ");
            scanf("%d", &node);
            delete_node(node);
            break;
        case 3:
            insert_edge();
            break;
        case 4:
            display();
            break;
        case 5:
            return 0;
        default:
            printf("Invalid choice! Please try again.\n");
            break;
        }
    }
}
```

OUTPUT:

```
Enter number of nodes: 3
Enter edge 1 (0 0 to quit): 1 2
Enter edge 2 (0 0 to quit): 2 3
Enter edge 3 (0 0 to quit): 3 1
Enter edge 4 (0 0 to quit): 0 0
```

```
1. Insert a node
2. Delete a node
3. Insert an edge
4. Display
5. Exit
Enter your choice: 4

Adjacency Matrix:
   0   1   0
   0   0   1
   1   0   0
```

```
1. Insert a node
2. Delete a node
3. Insert an edge
4. Display
5. Exit
Enter your choice: 1
The inserted node is 4
```

```
1. Insert a node
2. Delete a node
3. Insert an edge
4. Display
5. Exit
Enter your choice: 3
Enter an edge (origin destination): 3 4
Edge (3 -> 4) added successfully.
```

```
1. Insert a node
2. Delete a node
3. Insert an edge
4. Display
5. Exit
Enter your choice: 4

Adjacency Matrix:
   0   1   0   0
   0   0   1   0
   1   0   0   1
   0   0   0   0
```

```
1. Insert a node
2. Delete a node
3. Insert an edge
4. Display
5. Exit
Enter your choice: 5
PS D:\coding\C\lab>
```

RESULT:

The Program is executed successfully and the output is verified.

PROGRAM NO:9

AIM:

Graph traversal DFS and BFS

ALGORITHM:

Algorithm for DFS and BFS

1. Graph Creation:
    1. Prompt the user to enter the number of vertices and edges.
    2. Initialize the graph with the specified number of vertices.
    3. Prompt the user to enter the edges of the graph.
    4. Add each edge to the graph using the addEdge function.
2. Depth-First Search (DFS):
    1. Call the DFS function to perform DFS traversal with start and end times:
        - It iterates through all vertices and calls DFS_VISIT for unvisited vertices.
    2. DFS_VISIT function:
        1. Marks the current vertex as visited.
        2. Assigns a start time to the vertex.
        3. Prints the vertex.
        4. Recursively calls DFS_VISIT for its unvisited neighbors.
        5. Assigns an end time to the vertex.
    3. Breadth-First Search (BFS):
        1. Reset the visited array for BFS.
        2. Prompt the user to enter the starting vertex for BFS.
        3. Call the BFS function to perform BFS traversal:
            1. It uses a queue to maintain the order of vertices to be visited.
            2. It dequeues a vertex, marks it as visited, prints it, and adds its unvisited neighbors to the queue.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
struct Node
{
   int vertex;
   struct Node *next;
};

struct Graph
{
   int num_vertices;
   struct Node *adj_list[MAX_VERTICES];
   int visited[MAX_VERTICES];
   int start_time[MAX_VERTICES];
   int end_time[MAX_VERTICES];
   int time;
};
void initializeGraph(struct Graph *G, int num_vertices)
{
   G->num_vertices = num_vertices;
   for (int i = 0; i < num_vertices; ++i)
   {
      G->adj_list[i] = NULL;
      G->visited[i] = 0;
      G->start_time[i] = 0;
      G->end_time[i] = 0;
   }
   G->time = 0;
}
```

```
void addEdge(struct Graph *G, int src, int dest)
{
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->vertex = dest;
    newNode->next = G->adj_list[src];
    G->adj_list[src] = newNode;
}
void DFS_VISIT(struct Graph *G, int u)
{
    G->visited[u] = 1;
    G->start_time[u] = ++(G->time);
    printf("%d ", u);
    struct Node *temp = G->adj_list[u];
    while (temp != NULL)
    {
        int v = temp->vertex;
        if (!G->visited[v])
        {
            DFS_VISIT(G, v);
        }
        temp = temp->next;
    }
    G->end_time[u] = ++(G->time);
}
void DFS(struct Graph *G)
{
    printf("DFS Traversal: ");
    for (int i = 0; i < G->num_vertices; ++i)
    {
        if (!G->visited[i])
        {
```

```
            DFS_VISIT(G, i);
        }
    }
    printf("\n");
    printf("Node\tStart Time\tEnd Time\n");
    for (int i = 0; i < G->num_vertices; ++i)
    {
        printf("%d\t%d\t\t%d\n", i, G->start_time[i], G->end_time[i]);
    }
}
void BFS(struct Graph *G, int start_vertex)
{
    printf("BFS Traversal: ");
    int queue[MAX_VERTICES];
    int front = 0, rear = -1;
    G->visited[start_vertex] = 1;
    printf("%d ", start_vertex);
    queue[++rear] = start_vertex;
    while (front <= rear)
    {
        int u = queue[front++];
        struct Node *temp = G->adj_list[u];
        while (temp != NULL)
        {
            int v = temp->vertex;
            if (!G->visited[v])
            {
                G->visited[v] = 1;
                printf("%d ", v);
                queue[++rear] = v;
            }
```

```
            temp = temp->next;
        }
    }
    printf("\n");
}
int main()
{
    struct Graph G;
    int num_vertices, num_edges;

    printf("Enter number of vertices: ");
    scanf("%d", &num_vertices);
    printf("Enter the number of edges:");
    scanf("%d", &num_edges);
    initializeGraph(&G, num_vertices);
    printf("Enter edges (vertex u and v connected): \n");
    for (int i = 0; i < num_edges; ++i)
    {
        int u, v;
        scanf("%d %d", &u, &v);
        addEdge(&G, u, v);
    }
    DFS(&G);
    printf("\n");
    for (int i = 0; i < G.num_vertices; ++i)
    {
        G.visited[i] = 0;
    }
    int start_vertex;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &start_vertex);
```

```
    BFS(&G, start_vertex);
    return 0;
}
```

OUTPUT:

```
Enter number of vertices: 5
Enter the number of edges:4
Enter edges (vertex u and v connected):
0 1
1 2
2 3
3 4
DFS Traversal: 0 1 2 3 4
Node    Start Time      End Time
0       1               10
1       2               9
2       3               8
3       4               7
4       5               6

Enter the starting vertex for BFS: 0
BFS Traversal: 0 1 2 3 4
```

RESULT:

The program is executed and the output is obtained.

PROGRAM NO:10

AIM:

Write a c program to implement Prim's Algorithm to find the minimum cost of spanning tree.

ALGORITHM:

1. Associate with each vertex V for the graph no C[V] and edge E[V]. To initialize the values, set all value of C[V] to + infinity and set each E[V] to a special flag value indicating that there is no edge connecting V to earlier vertices.

2. Indicating an empty forest F and a set n of vertices that have not yet been included in F. Repeat the following steps until queue is empty.

   1.Find and remove G vertex V from 0 having the minimum possible value of E[V].

   2.Add V to F and if E[V] is not the special flag value, also add E[V] to F.

   3.Loop over the edges vw connecting v to other vertices for each v such edge, if

   w still belongs to G and vw has smallest weight when C[w], perform the following steps.

   a) Set C[w] to the cost of edge vw.
   Return F.

   b) Set E[w] to the point to edge vw.

   Return F

CODE:

```
#include<stdio.h>

#define MAX 10

int main()
{
        int vertex_array[MAX],counter;
        int vertex_count=0;
```

```
        int row,column;
        int cost_matrix[MAX][MAX];
        int visited[MAX]={0};
        int edge_count=0,count=1;
        int sum_cost=0,min_cost=0;
        int row_no,column_no,vertex1,vertex2;

        printf("Total no of vertex :: ");
        scanf("%d",&vertex_count);

        printf("\n-- Enter vertex -- \n\n");
        for(counter=1;counter<=vertex_count;counter++){
                printf("vertex[%d] :: ",counter);
                scanf("%d",&vertex_array[counter]);
}

printf("\n--- Enter Cost matrix of size %d x %d  ---\n\n",vertex_count,vertex_count);
printf("\n\t-- format is --\n");
for(row=1;row<=vertex_count;row++)
{
        for(column=1;column<=vertex_count;column++)
        {
                printf("x ");
        }
        printf("\n");
}
        printf("\n-- MATRIX --\n\n");
        for(row=1;row<=vertex_count;row++)
        {
                for(column=1;column<=vertex_count;column++)
                {
                        scanf("%d",&cost_matrix[row][column]);
```

```
                    if(cost_matrix[row][column] == 0)
                    {
                        cost_matrix[row][column] = 999;
                    }
                }
            }
            printf("\n");
            visited[1]=1;
            edge_count = vertex_count-1;
            while(count <= edge_count)
            {
                for(row=1,min_cost=999;row<=vertex_count;row++)
                {
                    for(column=1;column<=vertex_count;column++)
                    {
                        if(cost_matrix[row][column] < min_cost)
                        {
                            if(visited[row] != 0)
                            {
                                min_cost = cost_matrix[row][column];
                                vertex1 = row_no = row;
                                vertex2 = column_no = column;
                            }
                        }
                    }
                }

                if(visited[row_no] == 0 || visited[column_no] ==0)
                {
                    printf("\nEdge %d is (%d -> %d)with
        cost:%d",count++,vertex_array[vertex1],vertex_array[vertex2],min_cost);
                    sum_cost = sum_cost + min_cost;
```

```
                visited[column_no]=1;
            }
            cost_matrix[vertex1][vertex2] = cost_matrix[vertex2][vertex1] = 999;
        }
        printf("\n\nMinimum cost=%d",sum_cost);
        return 0;
}
```

OUTPUT:

```
Total no of vertex :: 4

-- Enter vertex --

vertex[1] :: 1
vertex[2] :: 2
vertex[3] :: 3
vertex[4] :: 4

--- Enter Cost matrix of size 4 x 4  ---

        -- format is --
x x x x
x x x x
x x x x
x x x x

-- MATRIX --

0 2 0 6
2 0 3 8
0 3 0 0
6 8 0 0


Edge 1 is (1 -> 2) with cost : 2
Edge 2 is (2 -> 3) with cost : 3
Edge 3 is (1 -> 4) with cost : 6

Minimum cost=11
```

RESULT:

The program is executed successfully and the output is verified.

PROGRAM NO:11

AIM:

Write a c program implement Kruskal's Algorithm using disjoint set data structure.

ALGORITHM:

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree.
2. Create a set E containing all the edges in the graph
3. Repeat Steps 4 and 5 while E is NOT EMPTY and F is not spanning
4. Remove an edge from E with minimum weight
5. If the edge obtained in Step4 connects two different trees, then add it to the forest (F) Else, discard the edge.
6. End

CODE:

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

#define MAX 10

int parent[MAX];

int find(int i)
{
   while(parent[i])
      i=parent[i];
   return i;
```

```
}

int uni(int i,int j)
{
   if(i!=j)
   {
      parent[j]=i;
      return 1;
   }
   return 0;
}


int main(){
   int vertex_count=0;
   int row,column;
   int cost_matrix[MAX][MAX];
   int edge_count=0,count=1;
   int sum_cost=0,min_cost;
   int row_no,column_no,edge1,edge2;

   printf("Implementation of Kruskal's algorithm\n\n");
   printf("Total no of vertex :: ");
   scanf("%d",&vertex_count);

   for(row=1;row<=vertex_count;row++)
   {
      for(column=1;column<=vertex_count;column++)
      {
         scanf("%d",&cost_matrix[row][column]);
         if(cost_matrix[row][column] == 0)
         {
```

```
            cost_matrix[row][column] = 999;
        }
    }
}

edge_count = vertex_count-1;

while(count <= edge_count)
{
    for(row=1,min_cost=999;row<=vertex_count;row++)
    {
        for(column=1;column<=vertex_count;column++)
        {
            if(cost_matrix[row][column] < min_cost)
            {
                    min_cost = cost_matrix[row][column];
                    edge1 = row_no    =  row;
                    edge2 = column_no =  column;
            }
        }
    }

    row_no    = find(row_no);
    column_no = find(column_no);

    if(uni(row_no,column_no))
    {
        printf("\nEdge %d is (%d -> %d) with cost : %d ",count++,edge1,edge2,min_cost);
        sum_cost = sum_cost + min_cost;
    }
    cost_matrix[edge1][edge2] = cost_matrix[edge2][edge1] = 999;
}
```

```
    printf("\n Minimum cost=%d",sum_cost);
    return 0;
}
```

OUTPUT:

```
Total no of vertex :: 4
0 10 20 30
10 0 40 50
20 40 0 60
30 50 60 0

Edge 1 is (1 -> 2) with cost : 10
Edge 2 is (1 -> 3) with cost : 20
Edge 3 is (1 -> 4) with cost : 30
 Minimum cost=60
```

RESULT:

The program is executed successfully and the output is verified.

PROGRAM NO: 12

AIM:

Write a c program to find the shortest path using Dijkstra’s algorithm.

ALGORITHM:

1. It takes the number of vertices (n), the adjacency matrix, and the starting node (u) as input.
2. It initializes the cost matrix (cost), distance array (distance), predecessor array (pred), and visited array (visited).
3. The cost matrix is created based on the adjacency matrix, and the distance and predecessor arrays are initialized with values from the starting node.
4. The main loop of Dijkstra's algorithm is executed to find the shortest path and distance to all nodes.
5. After the algorithm completes, the program prints the distance and path from the starting node to every other node in the graph.

CODE:

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10

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

int main()
{
   int G[MAX][MAX], i, j, n, u;
   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]);

    printf("\nEnter 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 of 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:4

Enter the adjacency matrix:
0 2 0 1
2 0 3 0
0 3 0 7
1 0 7 0

Enter the starting node:0
```

```
Distance of node 1 = 2
Path = 1 <- 0
Distance of node 2 = 5
Path = 2 <- 1 <- 0
Distance of node 3 = 1
Path = 3 <- 0
```

RESULT:

The Program is executed successfully and the output is verified.