LAB MANUAL

Design and Analysis of Algorithm (DAA)

2301CS402

B-Tech 5th Semester

A.Y. 2025-2026

(Darshan Institute of Engineering and Technology)

Darshan
UNIVERSITY

योग: कर्मसु कौशलम्

Table of Content

Program 1: Write recursive program for calculation of factorial of an integer.

Program 1: Code

```c
# include<stdio.h>
double fact(double n){
    if(n==1)
        return 1;
    else
        return(n*fact(n-1));
}
void main(){
    double n, ans;
    printf("Enter a number:");
    scanf("%lf",&n);
    ans=fact(n);
    printf("\nAns : %lf \n",ans);
}
```

Program 1: Output

```
Enter a number:20
Ans : 2432902008176640000.000000
```

Program 2. Write a program to calculate the sum of numbers from 1 to n using recursion.

Program 2: Code

```c
#include<stdio.h>
int sumOfRange(int);
void main(){
   int n1;
   int sum;
   printf(" Input the last number of the range starting from 1 : ");
   scanf("%d", &n1);
   sum = sumOfRange(n1);
   printf("\n The sum of numbers from 1 to %d : %d\n\n", n1, sum);
}
int sumOfRange(int n1)
{
   int res;
   if (n1 == 1)
   {
      return (1);
   } else
   {
      res = n1 + sumOfRange(n1 - 1); //calling the function sumOfRange itself
   }
```

```
    return (res);
}
```

Program 2: Output

```
Input the last number of the range starting from 1 : 20
The sum of numbers from 1 to 20 : 210
```

Program 3. Write a program to count the digits of a given number using recursion.

Program 3: Code

```c
#include<stdio.h>
int noOfDigits(int n1);
void main()
{
  int n1,ctr;
    printf(" Input  a number : ");
    scanf("%d",&n1);
    ctr = noOfDigits(n1);
    printf(" The number of digits in the number is :  %d \n\n",ctr);
}

int noOfDigits(int n1)
{
    static int ctr=0;
     if(n1!=0)
     {
          ctr++;
         noOfDigits(n1/10);
    }
    return ctr;
}
```

Program 3: Output

```
Input  a number : 12345
The number of digits in the number is :  5
```

Program 4. Write a program to calculate the power of any number using recursion.

Program 4: Code

```c
#include <stdio.h>
long int CalcuOfPower(int x,int y)
{
    long int result=1;
    if(y == 0)
    return result;
    result=x*(CalcuOfPower(x,y-1));
}
void main()
{
    int bNum,pwr;
    long int result;
    printf(" Input the base  value : ");
    scanf("%d",&bNum);
    printf(" Input the value of power : ");
    scanf("%d",&pwr);
    result=CalcuOfPower(bNum,pwr);
    printf(" The value of %d to the power of %d is : %ld\n\n",bNum,pwr,result);
}
```

Program 4: Output

```
Input the base  value : 2
Input the value of power : 8
The value of 2 to the power of 8 is : 256
```

Program 1. Write a program to implement stack operations (PUSH, POP, PEEP, CHANGE & DISPLAY)

Program 1: Code

```c
#include<stdio.h>
#define size 5
struct stack{
   int a[size],top;
   int temp[size], tos;
}s;
// Push operation....
void push(int item){
        s.a[++s.top] = item;
}
// Pop operation....
int pop(){
    return s.a[s.top--];
}
// Display operation....
void display(){
    int i;
    printf("\nThe stack contains: ");
    for(i = s.top; i>=0; i--){
        printf("\n\t%d", s.a[i]);
    }
}
// Peep operation....
void peep(){
    printf("\n\tTop : %d", s.top);
    printf("\n\tValue: %d",s.a[s.top]);
}
void change(int row, int new_element){
    int i;
    int j = -1;
    printf("\n\tTop: %d", s.top);
    for(i=s.top; i>row; i--){
        s.temp[++j] = s.a[s.top--];
    }
    s.a[s.top] = new_element;

    for(i = j; i>-1; i--){
        s.a[++s.top] = s.temp[j--];
    }
}
void main(){
    s.top = -1;
    int item, choice, row, new_element;
    char ans;
    do{
        printf("\n 1. Push\n 2. Pop\n 3. Display\n 4. Peep\n 5. Change\n6.
Exit\n");
        printf("-----------------------------\n");
        printf("\n  Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
```

```
        case 1:
            if(s.top >= size-1){
                 printf("\nStack overflow..\n");
                 break;
            }
            printf("\nEnter item to be pushed:  ");
            scanf("%d", &item);
            push(item);
            break;
        case 2:
            if(s.top == -1){
                printf("\n..Stack underflow..\n");
                break;
            }
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            peep();
            break;
        case 5:
            printf("\n\tEnter row no : ");
            scanf("%d",&row);
            printf("\n\tEnter new element: ");
            scanf("%d", &new_element);
            change(row, new_element );
            break;
        case 6:
            return 0;
        }
    }while(choice != 6);
}
```

Program 2. Write a program to implement queue operations (INSERT, DELETE, DISPLAY)

Program 2: Code

```
#include <stdio.h>
#define MAX 5
void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(1);
            default:
            printf("Wrong choice \n");
        }
    }
}
void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
}
void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
}
void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}
```

Program 3. Write a program to implement singly linked list operations (INSERT, DELETE, DISPLAY).

Program 3: Code

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *start;
void insert_begin();
void insert_last();
void insert_locc();
void delete_begin();
void delete_last();
void delete_locc();
void print();
void main ()
{
    int ch=0;
    while(ch!=8)
    {
        printf("\nEnter the operation to be performed\n");
             printf("\n1.Insert in the begining\n2.Insert at last\n3.Insert at
     any specified position\n  4.Delete from Beginning\n5.Delete from
     last\n6.Delete node after specified location\n 7.Show\n
     8.Exit\n");
        scanf("\n%d",&ch);
        switch(ch)
        {        /*function calls of all the operations */
            case 1:
            insert_begin();
            break;
            case 2:
            insert_last();
            break;
            case 3:
            insert_locc();
            break;
            case 4:
            delete_begin();
            break;
            case 5:
            delete_last();
            break;
            case 6:
            delete_locc();
            break;
            case 7:
            print();
            break;
            case 8:
            exit(0);
            break;
            default:
```

```
            printf("Enter valid option");
        }
    }
}           /*function definition*/
void insert_begin()                  //to insert the node at the beginnning of
linked list
{
    struct node *p;
    int value;
    p=(struct node *) malloc(sizeof(struct node *));
    if(p==NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&value);
        p->data=value;
        p->next=start;
        start=p;
    }
}
void insert_last()                //to insert the node at the last of linked
list
{
    struct node *p,*temp;
    int value;
    p=(struct node*)malloc(sizeof(struct node));
    if(p==NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&value);
        p->data=value;
        if(start==NULL)
        {
            p->next=NULL;
            start=p;
        }
        else
        {
            temp=start;
            while(temp->next!=NULL)
            {
                temp=temp->next;
            }
            temp->next=p;
            p->next=NULL;
        }
    }
}
void insert_locc()               //to insert the node at the specified location
of linked list
{
    int i,loc,value;
    struct node *p, *temp;
```

```c
    p=(struct node *)malloc(sizeof(struct node));
    if(p==NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter element value");
        scanf("%d",&value);
        p->data=value;
        printf("\nEnter the location after which you want to insert ");
        scanf("\n%d",&loc);
        temp=start;
        for(i=0;i<loc;i++)
        {
            temp=temp->next;
            if(temp==NULL)
            {
                printf("\ncan't insert\n");
                return;
            }
        }
        p->next=temp->next;
        temp->next=p;
    }
}
void delete_begin()          //to delete the node present in the beginning of
the linked list
{
    struct node *p;
    if(start==NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        p=start;
        start=p->next;
        free(p);
    }
}
void delete_last()          //to delete the node present in the last of the
linked list
{
    struct node *p,*p1;
    if(start==NULL)
    {
        printf("\nlist is empty");
    }
    else if(start->next==NULL)
    {
        start=NULL;
        free(start);
        printf("\nOnly node of the list deleted ...\n");
    }
    else
    {
        p=start;
        while(p->next!=NULL)
        {
            p1=p;
            p=p->next;
        }
        p1->next=NULL;
        free(p);
    }
}
void delete_locc()    //to delete the node present at the specified of the 
linked list
{
    struct node *p,*p1;
    int loc,i;
    printf("\n Enter the location of the node after which you want to perform 
deletion \n");
    scanf("%d",&loc);
    p=start;
    for(i=0;i<loc;i++)
    {
        p1=p;
        p=p->next;

        if(p==NULL)
        {
            printf("\nCan't delete");
            return;
        }
    }
    p1->next=p->next;
    free(p);
    printf("\nDeleted node %d ",loc+1);
}
void print()    //to print the values in the linked list
{
    struct node *p;
    p=start;
    if(p==NULL)
    {
        printf("Nothing to print");
    }
    else
    {
        printf("\nprinting values\n");
        while (p!=NULL)
        {
            printf("\n%d",p->data);
            p=p->next;
        }
    }
}
```

Program 1. Write a program to sort array elements using bubble sort.

Program 1: Code

```c
#include<stdio.h>
void main()
{
    int array[100], n, i, j, swap;
    printf("Enter number of 
elements:");
    scanf("%d", &n);
    printf("\nEnter %d Numbers:\n", 
n);
    for(i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }
    for(i = 0 ; i < n - 1; i++)
    {
        for(j = 0 ; j < n-i-1; j++)
        {
            if(array[j] > 
array[j+1])
            {
                swap=array[j];
                array[j]=array[j+1];
                array[j+1]=swap;
            }
        }
    }
    printf("Sorted Array:\n");
    for(i = 0; i < n; i++)
    {
        printf("%d\n", array[i]);
    }
}
```

Program 1: Output

```
Enter number of elements:5
Enter 5 Numbers:
9
8
7
6
5
Sorted Array:
5
6
7
8
9
```

Program 2. Write a program to sort array elements using insertion sort.

Program 2: Code

```c
#include <stdio.h>
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
void main()
{
    int arr[100], n, i;
    printf("Enter number of elements:");
    scanf("%d", &n);
    printf("\nEnter %d Numbers:\n", n);
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    insertionSort(arr, n); printf("Sorted Array:\n");
    printArray(arr, n);
}
```

Program 2: Output

```
Enter number of elements:5
Enter 5 Numbers:
9
8
7
6
5
```

```
Sorted Array:
5
6
7
8
9
```

Program 1. Write a program to sort array elements using selection sort.

Program 1: Code

```c
#include <stdio.h>
void main()
{
    int x,arr[100],i,n,pos,count,val,j,minx,minj;

    printf("Enter the number of elements to be sorted: ");
    scanf("%d",&n);
    printf("\nEnter %d Numbers:\n", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d",&arr[i]);
    }

    for(i=0; i < n; i++)
    {
     minj=i;
     minx=arr[i];
        for(j=i+1;j<n;j++)
        {
            if(arr[j]<minx)
            {
              minj=j;
              minx=arr[j];
            }
        }
     arr[minj]=arr[i];
     arr[i]=minx;
    }
    printf("Sorted Array:\n");
    for(i=0;i<n;++i)
    {
        printf("%d  ",arr[i]);
    }
}

```

Program 1: Output

```
Enter number of elements:5
Enter 5 Numbers:
9
8
7
6
5
```

```
Sorted Array:
5
6
7
8
9
```

Program 2. Write a program to sort array elements using heap sort.

Program 2: Code

```c
#include <stdio.h>
void main()
{
    int arr[100], n,  i, j, c, heap_root, temp;
    printf("Enter the number of elements to be sorted: ");
    scanf("%d",&n);
    printf("\nEnter %d Numbers:\n", n);
    for (i=0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }
    for (i=1; i<n; i++)
    {
        c = i;
        do
        {
            heap_root  = (c-1)/2;
            if(arr[heap_root] < arr[c])
            {
                temp =  arr[heap_root];
                arr[heap_root] = arr[c];
                arr[c]  = temp;
            }
            c =  heap_root;
        } while (c !=  0);
    }
    printf("Heap  array : ");
    for (i=0; i<n; i++)
        printf("%d ", arr[i]);
    for (j=n-1; j>=0; j--)
    {
        temp = arr[0];
        arr[0] = arr[j];
        arr[j] = temp;
        heap_root = 0;
        do
        {
            c = 2 *  heap_root + 1;
            if  ((arr[c] < arr[c + 1]) && c < j-1)
                c++;
            if  (arr[heap_root]<arr[c] && c<j)
            {
                temp =  arr[heap_root];
                arr[heap_root] = arr[c];
                arr[c]  = temp;
            }
            heap_root  = c;
        } while (c<j);
    }
    printf("\nSorted  array : ");
    for (i=0; i<n; i++)
        printf("%d ", arr[i]);
}

```

Program 2: Output

```
Enter the number of elements to be sorted: 5
Enter 5 Numbers:
9
8
7
6
5
Heap  array : 9 8 7 6 5
Sorted  array : 5 6 7 8 9
```

Program 1. Write a program to implement linear search algorithm.

Program 1: Code

```c
#include<stdio.h>
void main()
{
    int a[20],i,x,n;
    printf("How many elements?");
    scanf("%d",&n);
    printf("Enter array elements:n");
    for(i=0;i<n;++i)
        scanf("%d",&a[i]);

    printf("nEnter element to search:");
    scanf("%d",&x);
    for(i=0;i<n;++i)
        if(a[i]==x)
            break;
    if(i<n)
        printf("Element found at index %d",i);
    else
        printf("Element not found");
}
```

Program 1: Output

```
How many elements?5
Enter array elements:
9
8
7
6
5
Enter element to search:5
Element found at index 4
```

Program 2. Write a program to implement binary search algorithm.

Program 2: Code

```c
#include<stdio.h>
#include<stdlib.h>
int binsearch(int[], int, int, int);
void main()
{
    int num, i, key, position;
    int low, high, list[100];
    printf("Enter the total number of elements:");
    scanf("%d", &num);
    printf("\nEnter the elements of list:\n");
    for (i = 0; i < num; i++)
    {
      scanf("%d", &list[i]);
    }
    for (i = 0; i < num; i++)
    {
        if(list[i] < list[i - 1])
        {
            printf("Given input is not sorted\n");
            return 0;
        }
    }
    low = 0;
    high = num - 1;
    printf("\nEnter element to be searched: ");
    scanf("%d", &key);
    position = binsearch(list, key, low, high);
    if (position != -1)
    {
      printf("\nNumber present at %d", (position + 1));
    }
    else
    {
      printf("\n The number is not present in the list");
    }
    return (0);
    }
int binsearch(int a[], int x, int low, int high)
{
    int mid;
    if (low > high)
      return -1;
    mid = (low + high) / 2;
    if (x == a[mid])
    {
      return (mid);
    }
    else if (x < a[mid])
    {
      binsearch(a, x, low, mid - 1);
    }
    else
    {
      binsearch(a, x, mid + 1, high);
    }
}
```

Program 2: Output

```
Enter the total number of elements:5
Enter the elements of list:
10
20
30
```

```
40
50
Enter element to be searched: 20
Number present at 2
```

Program 1. Write a program to implement quick sort algorithm.

Program 1: Code

```c
#include<stdio.h>
int pivot(int[],int,int);
void swap(int[],int,int);
void quicksort(int[],int,int);
int pivot(int arr[],int i,int j)
{
    int p = arr[i],temp;
    int k = i;
    int l = j+1;
    do
    {
        k++;
    }   while(arr[k]<=p && k<j);
    do
    {
        l--;
    }   while(arr[l]>p);
    while(k<l)
    {
        temp=arr[k];
        arr[k]=arr[l];
        arr[l]=temp;
        do
        {
            k++;
        }   while(arr[k] <= p);
        do{
            l--;
        }   while(arr[l] > p);
    }
    temp=arr[i];
    arr[i]=arr[l];
    arr[l]=temp;
    return l;
}
void quicksort(int arr[],int i,int j)
{
    if(i<j)
    {
        int position = pivot(arr,i,j);
        quicksort(arr,i,position-1);
        quicksort(arr,position+1,j);
    }
}
void main()
{
    int arr[1000],n,i;

    printf("Enter the number of elements to be sorted: ");
    scanf("%d",&n);
    printf("\nEnter %d Numbers:\n", n);
    for (i=0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }
    quicksort(arr,0,n-1);
    printf("Sorted Array:\n");
    for (i=0; i<n; i++)
    {
        printf("%d ",arr[i]);
    }
}
```

Program 1: Output

```
Enter the number of elements to be sorted: 5
Enter 5 Numbers:
9
8
7
6
5
Sorted Array:
5 6 7 8 9
```

Program 2. Write a program to implement merge sort algorithm.

Program 2: Code

```c
#include<stdio.h>
void disp( );
void mergesort(int,int,int);
void msortdiv(int,int);
int arr[100],n;
void main( )
{
    int i;
    printf("Enter the number of elements to be sorted: ");
    scanf("%d",&n);
    printf("\nEnter %d Numbers:\n", n);
    for (i=0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }
    printf("\nBefore Sorting the elements are:");
    disp( );
    msortdiv(0,n-1);
    printf("\nAfter Sorting the elements are:");
    disp( );
}
void disp( )
{
    int i;
    for(i=0;i<n;i++)
    printf("%d ",arr[i]);
}
void mergesort(int low,int mid,int high)
```

```
{
    int t[50],i,j,k;
    i=low;
    j=mid+1;
    k=low;
    while((i<=mid) && (j<=high))
    {
        if(arr[i]>=arr[j])
            t[k++]=arr[j++];
        else
            t[k++]=arr[i++];
    }
    while(i<=mid)
    {
        t[k++]=arr[i++];
    }
    while(j<=high)
    {
        t[k++]=arr[j++];
    }
    for(i=low;i<=high;i++)
    {
        arr[i]=t[i];
    }
}
void msortdiv(int low,int high)
{
    int mid;
    if(low!=high)
    {
        mid=((low+high)/2);
        msortdiv(low,mid);
        msortdiv(mid+1,high);
        mergesort(low,mid,high);
    }
}
```

Program 2: Output

```
Enter the number of elements to be sorted: 8
Enter 8 Numbers:
9
8
7
6
5
4
3
2
Before Sorting the elements are:9 8 7 6 5 4 3 2
After Sorting the elements are:2 3 4 5 6 7 8 9
```

Program 1. Write a program to study and implement minimum spanning tree using Kruskal’s algorithm.

Program 1: Code

```c
#include <stdio.h>
#define MAX 30
typedef struct edge {
  int u, v, w;
} edge;
typedef struct edge_list {
  edge data[MAX];
  int n;
} edge_list;
edge_list elist;
int Graph[MAX][MAX], n;
edge_list spanlist;
void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();
void kruskalAlgo() {
  int belongs[MAX], i, j, cno1, cno2;
  elist.n = 0;
  for (i = 1; i < n; i++)
    for (j = 0; j < i; j++) {
      if (Graph[i][j] != 0) {
        elist.data[elist.n].u = i;
        elist.data[elist.n].v = j;
        elist.data[elist.n].w = Graph[i][j];
        elist.n++;
      }
    }
    sort();
  for (i = 0; i < n; i++)
    belongs[i] = i;
  spanlist.n = 0;
  for (i = 0; i < elist.n; i++) {
    cno1 = find(belongs, elist.data[i].u);
    cno2 = find(belongs, elist.data[i].v);

    if (cno1 != cno2) {
      spanlist.data[spanlist.n] = elist.data[i];
      spanlist.n = spanlist.n + 1;
      applyUnion(belongs, cno1, cno2);
    }
  }
}
int find(int belongs[], int vertexno) {
  return (belongs[vertexno]);
}
void applyUnion(int belongs[], int c1, int c2) {
  int i;

  for (i = 0; i < n; i++)
    if (belongs[i] == c2)
```

```
      belongs[i] = c1;
}
void sort() {
  int i, j;
  edge temp;

  for (i = 1; i < elist.n; i++)
    for (j = 0; j < elist.n - 1; j++)
      if (elist.data[j].w > elist.data[j + 1].w) {
        temp = elist.data[j];
        elist.data[j] = elist.data[j + 1];
        elist.data[j + 1] = temp;
      }
}
void print() {
  int i, cost = 0;

  for (i = 0; i < spanlist.n; i++) {
    printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v,
spanlist.data[i].w);
    cost = cost + spanlist.data[i].w;
  }
  printf("\nSpanning tree cost: %d", cost);
}
void main()
{
  int i, j, total_cost;
  n = 6;
  Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 4;
  Graph[0][3] = 0;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;

  Graph[1][0] = 4;
  Graph[1][1] = 0;
  Graph[1][2] = 2;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 0;
  Graph[1][6] = 0;

  Graph[2][0] = 4;
  Graph[2][1] = 2;
  Graph[2][2] = 0;
  Graph[2][3] = 3;
  Graph[2][4] = 4;
  Graph[2][5] = 0;
  Graph[2][6] = 0;

  Graph[3][0] = 0;
  Graph[3][1] = 0;
  Graph[3][2] = 3;
  Graph[3][3] = 0;
  Graph[3][4] = 3;
  Graph[3][5] = 0;
  Graph[3][6] = 0;

```

```
    Graph[4][0] = 0;
    Graph[4][1] = 0;
    Graph[4][2] = 4;
    Graph[4][3] = 3;
    Graph[4][4] = 0;
    Graph[4][5] = 0;
    Graph[4][6] = 0;

    Graph[5][0] = 0;
    Graph[5][1] = 0;
    Graph[5][2] = 2;
    Graph[5][3] = 0;
    Graph[5][4] = 3;
    Graph[5][5] = 0;
    Graph[5][6] = 0;
    kruskalAlgo();
    print();
  }
```

Program 1: Output

```
2 - 1 : 2
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4
Spanning tree cost: 14
```

Program 2. Write a program to study and implement minimum spanning tree using Prim's algorithm.

Program 2: Code

```
#include<stdio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main(){
    printf("\nEnter the number of nodes:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
        }
       visited[1]=1;
    printf("\n");
    while(ne < n)    {
        for(i=1,min=999;i<=n;i++)
        for(j=1;j<=n;j++)
        if(cost[i][j]< min)
        if(visited[i]!=0)         {
            min=cost[i][j];
            a=u=i;
            b=v=j;         }
        if(visited[u]==0 || visited[v]==0)         {
            printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
            mincost+=min;
            visited[b]=1;         }
        cost[a][b]=cost[b][a]=999;
    }
    printf("\n Minimum cost=%d",mincost);
}
```

Program 2: Output

```
Enter the number of nodes: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
```

```
Edge 1:(1 3) cost:1
Edge 2:(1 2) cost:3
Edge 3:(2 5) cost:3
Edge 4:(3 6) cost:4
Edge 5:(6 4) cost:2
Minimum cost=13
```

Program 1. Write a program to study and implement Dijkstra's algorithm.

Program 1: Code

```
#include<stdio.h>
#define INFINITY 9999
#define MAX 10
void dijikstra(int G[MAX][MAX], int n, int startnode);
void main(){
    int G[MAX][MAX], i, j, n, u;
    printf("\nEnter the 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);
    dijikstra(G,n,u);
}
void dijikstra(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 %d = %d", i, distance[i]);
            printf("\nPath = %d", i);
            j=i;
            do
            {
                j=pred[j];
                printf(" <-%d", j);
            }
            while(j!=startnode);
        }
}
```

Program 1: Output

```
Enter the no. of vertices:: 4

Enter the adjacency matrix::
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0

Enter the starting node:: 1

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

Program 2. Write a program to implement Knapsack problem using greedy approach.

Program 2: Code

```
#include<stdio.h>
int max(int a, int b) {
   if(a>b){
      return a;
   } else {
      return b;
   }
}
int knapsack(int W, int wt[], int val[], int n) {
   int i, w;
   int knap[n+1][W+1];
   for (i = 0; i <= n; i++) {
      for (w = 0; w <= W; w++) {
         if (i==0 || w==0)
            knap[i][w] = 0;
         else if (wt[i-1] <= w)
            knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);
         else
            knap[i][w] = knap[i-1][w];
      }
   }
   return knap[n][W];
}
int main() {
   int val[] = {20, 25, 40};
   int wt[] = {25, 20, 30};
   int W = 50;
   int n = sizeof(val)/sizeof(val[0]);
   printf("The solution is : %d", knapsack(W, wt, val, n));
   return 0;
}
```

Program 2: Output

```
The solution is : 65
```

Program 1. Write a program to implement Huffman code algorithm.

Program 1: Code

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
        char ch;
        int freq;
        struct node *left;
        struct node *right;
}node;

node * heap[100];
int heapSize=0;

void Insert(node * element)
{
        heapSize++;
        heap[heapSize] = element;
        int now = heapSize;
        while(heap[now/2] -> freq > element -> freq)
        {
                heap[now] = heap[now/2];
                now /= 2;
        }
        heap[now] = element;
}
node * DeleteMin()
{
        node * minElement,*lastElement;
        int 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]->freq < heap[child] ->
freq )
                {
                        child++;
                }
                if(lastElement -> freq > heap[child] -> freq)
                {
                        heap[now] = heap[child];
                }
                else
                {
                        break;
                }
        }
        heap[now] = lastElement;
        return minElement;
}
void print(node *temp,char *code)
```

```
{
        if(temp->left==NULL && temp->right==NULL)
        {
                printf("char %c code %s\n",temp->ch,code);
                return;
        }
        int length = strlen(code);
        char leftcode[10],rightcode[10];
        strcpy(leftcode,code);
        strcpy(rightcode,code);
        leftcode[length] = '0';
        leftcode[length+1] = '\0';
        rightcode[length] = '1';
        rightcode[length+1] = '\0';
        print(temp->left,leftcode);
        print(temp->right,rightcode);
}
int main()
{

       heap[0] = (node *)malloc(sizeof(node));
        heap[0]->freq = 0;
        int n ;
        printf("Enter the no of characters: ");
        scanf("%d",&n);
        printf("Enter the characters and their frequencies: ");
        char ch;
        int freq,i;

        for(i=0;i<n;i++)
        {
                scanf(" %c",&ch);
                scanf("%d",&freq);
                node * temp = (node *) malloc(sizeof(node));
                temp -> ch = ch;
                temp -> freq = freq;
                temp -> left = temp -> right = NULL;
                Insert(temp);
        }
        if(n==1)
        {
                printf("char %c code 0\n",ch);
                return 0;
        }
        for(i=0;i<n-1 ;i++)
        {
                node * left = DeleteMin();
                node * right = DeleteMin();
                node * temp = (node *) malloc(sizeof(node));
                temp -> ch = 0;
                temp -> left = left;
                temp -> right = right;
                temp -> freq = left->freq + right -> freq;
                Insert(temp);
        }
        node *tree = DeleteMin();
        char code[10];
        code[0] = '\0';
        print(tree,code);
}
```

Program 1: Output

```
Enter the no of characters: 6
Enter the characters and their frequencies: A 5
B 9
C 12
D 13
E 16
F 45
char F code 0
char C code 100
char D code 101
char A code 1100
char B code 1101
char E code 111
```

Program 2. Write a program to implement job scheduling problem using greedy approach.

Program 2: Code

```c
#include <stdio.h>

typedef struct {
    char id;
    int deadline;
    int profit;
} Job;

void jobScheduling(Job jobs[], int n) {
    int i, j, maxDeadline = 0;

    // Find maximum deadline
    for (i = 0; i < n; i++) {
        if (jobs[i].deadline > maxDeadline)
            maxDeadline = jobs[i].deadline;
    }

    // Sort jobs by profit (descending order)
    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (jobs[j].profit > jobs[i].profit) {
                Job temp = jobs[i];
                jobs[i] = jobs[j];
                jobs[j] = temp;
            }
        }
    }

    // Create time slots
    char slot[maxDeadline + 1];
    for (i = 1; i <= maxDeadline; i++)
```

```c
        slot[i] = '-'; // empty slot

    // Fill slots using greedy method
    for (i = 0; i < n; i++) {
        for (j = jobs[i].deadline; j > 0; j--) {
            if (slot[j] == '-') {
                slot[j] = jobs[i].id;
                break;
            }
        }
    }

    // Print job sequence
    printf("\nOptimal Job Sequence: ");
    for (i = 1; i <= maxDeadline; i++) {
        if (slot[i] != '-')
            printf("%c ", slot[i]);
    }
}

int main() {
    Job jobs[] = {
        {'A', 2, 100},
        {'B', 1, 19},
        {'C', 2, 27},
        {'D', 1, 25},
        {'E', 3, 15}
    };

    int n = sizeof(jobs) / sizeof(jobs[0]);

    printf("Job Scheduling Problem using Greedy Approach\n");
    printf("Jobs: (ID, Deadline, Profit)\n");
    for (int i = 0; i < n; i++) {
        printf("(%c, %d, %d)\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);
    }

    jobScheduling(jobs, n);

    return 0;
}
```

Program 2: Output

```
Job Scheduling Problem using Greedy Approach
Jobs: (ID, Deadline, Profit)
(A, 2, 100)
(B, 1, 19)
(C, 2, 27)
(D, 1, 25)
(E, 3, 15)

Optimal Job Sequence: A C E
```

Program 1. Write a program to implement making a change problem using dynamic programming.

Program 1: Code

```c
#include <stdio.h>
#define INF 999
//total different denominations of coins available
#define N 3
//amount for which we are making change
#define A 8
void display(int arr[A+1]);
void coinChange(int d[N+1], int C[A+1], int S[A+1]);
void coinSet(int d[N+1], int S[A+1]);
void main() {
  //denomination d of the coins
  //we will start from index 1
  //so, index 0 is set to 0
  int d[N+1] = {0, 1, 4, 6};
  int C[A+1];
  int S[A+1];
  coinChange(d, C, S);
  printf("\nC[p]\n");
  display(C);
  printf("\nS[p]\n");
  display(S);
  printf("\nMin. no. of coins required to make change for amount %d = %d\n", A,
C[A]);
  printf("\nCoin Set\n");
  coinSet(d, S);
}
void coinChange(int d[N+1], int C[A+1], int S[A+1]) {
  int i, p, min, coin;
  C[0] = 0;
  S[0] = 0;
  for(p = 1; p <= A; p++) {
    min = INF;
    for(i = 1; i <= N; i++) {
      if(d[i] <= p) {
        if(1 + C[p - d[i]] < min) {
          min = 1 + C[p - d[i]];
          coin = i;
        }
      }
    }
    C[p] = min;
    S[p] = coin;
  }
}
void coinSet(int d[N+1], int S[A+1]) {
  int a = A;
  while(a > 0) {
    printf("Use coin of denomination: %d\n", d[S[a]]);
    a = a - d[S[a]];
  }
}
void display(int arr[A+1]) {
  int c;
  for(c = 0; c <= A; c++) {
    printf("%5d", arr[c]);
  }
  printf("\n");
}
```

Program 1: Output

```
C[p]
   0    1    2    3    1    2    1    2    2


S[p]
   0    1    1    1    2    1    3    1    2

Min. no. of coins required to make change for amount 8 = 2

Coin Set
Use coin of denomination: 4
Use coin of denomination: 4
```

Program 2. Write a program to Implement 0/1 knapsack problem using dynamic programming.

Program 2: Code

```c
#include<stdio.h>
int max(int a, int b)
{
    return (a > b)? a : b;
}
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n+1][W+1];
   for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0)
               K[i][w] = 0;
           else if (wt[i-1] <= w)
                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
           else
                 K[i][w] = K[i-1][w];
       }
   }
   return K[n][W];
}
void main()
{
    int i, n, val[20], wt[20], W;
    printf("Enter number of items:");
    scanf("%d", &n);
    printf("Enter value and weight of items:\n");
    for(i = 0;i < n; ++i){
     scanf("%d%d", &val[i], &wt[i]);
    }
    printf("Enter size of knapsack:");
    scanf("%d", &W);
    printf("%d", knapSack(W, wt, val, n));
}
```

Program 2: Output

```
Enter number of items:3
Enter value and weight of items:
100 20
50 10
150 30
Enter size of knapsack:50
250
```

Program 1. Write a program to solve all-pairs shortest paths problem using Floyd's algorithm

Program 1: Code

```c
#include <stdio.h>
#define INF 99999
#define V 4   // Number of vertices in the graph

void floydWarshall(int graph[V][V]) {
    int dist[V][V], i, j, k;

    // Initialize distance matrix same as input graph
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    // Floyd-Warshall algorithm
    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    // Print the shortest distance matrix
    printf("\nAll-Pairs Shortest Paths (Floyd's Algorithm):\n");
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

int main() {
    /* Example Graph */
    int graph[V][V] = {
        {0,   5,  INF, 10},
        {INF, 0,   3,  INF},
        {INF, INF, 0,   1},
        {INF, INF, INF, 0}
    };

    printf("Floyd's Algorithm - All Pairs Shortest Path\n");
    floydWarshall(graph);
    return 0;
}
```

Program 1: Output

```
Floyd's Algorithm - All Pairs Shortest Path

All-Pairs Shortest Paths (Floyd's Algorithm):
    0     5     8     9
  INF     0     3     4
  INF   INF     0     1
  INF   INF   INF     0
```

Program 2. Write a program to implement Largest Common Sub-sequence.

Program 2: Code

```c
#include <stdio.h>
#include <string.h>
int i, j, m, n, LCS_table[20][20];
char S1[20] = "ACADB", S2[20] = "CBDA", b[20][20];
void lcsAlgo() {
  m = strlen(S1);   n = strlen(S2);
  for (i = 0; i <= m; i++)
    LCS_table[i][0] = 0;
  for (i = 0; i <= n; i++)
    LCS_table[0][i] = 0;
  for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++) {
      if (S1[i - 1] == S2[j - 1]) {
        LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
      } else if (LCS_table[i - 1][j] >= LCS_table[i][j - 1]) {
        LCS_table[i][j] = LCS_table[i - 1][j];
      } else {
        LCS_table[i][j] = LCS_table[i][j - 1];      }
    }
  int index = LCS_table[m][n];
  char lcsAlgo[index + 1];
  lcsAlgo[index] = '\0';
  int i = m, j = n;
  while (i > 0 && j > 0) {
    if (S1[i - 1] == S2[j - 1]) {
      lcsAlgo[index - 1] = S1[i - 1];
      i--;      j--;      index--;
    }
    else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
      i--;
    else
```

```c
      j--;
  }
  printf("S1 : %s \nS2 : %s \n", S1, S2);
  printf("LCS: %s", lcsAlgo);
}
int main() {
  lcsAlgo();
  printf("\n");
}
```

Program 2: Output

```
S1 : ACADB
S2 : CBDA
LCS: CB
```

Program 1. Write a program to implement the DFS algorithm.

Program 1: Code

```
#include<stdio.h>
int a[20][20],reach[20],n;
void dfs(int v) {
    int i;
    reach[v]=1;
    for (i=1;i<=n;i++)
      if(a[v][i] && !reach[i]) {
        printf("\n %d->%d",v,i);
        dfs(i);
    }
}
void main() {
    int i,j,count=0;
    printf("\n Enter number of vertices:");
    scanf("%d",&n);
    for (i=1;i<=n;i++) {
        reach[i]=0;
        for (j=1;j<=n;j++)
           a[i][j]=0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
       scanf("%d",&a[i][j]);
    dfs(1);
    printf("\n");
    for (i=1;i<=n;i++) {
        if(reach[i])
           count++;
    }
    if(count==n)
      printf("\n Graph is connected"); else
      printf("\n Graph is not connected");
}
```

Program 1: Output

```
Enter number of vertices:4
Enter the adjacency matrix:
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
```

```
 1->2
 2->3
 3->4

Graph is connected
```

Program 2. Write a program to implement the BFS algorithm.

Program 2: Code

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

```c
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v) {
    for (i=1;i<=n;i++)
      if(a[v][i] && !visited[i])
       q[++r]=i;
    if(f<=r) {
        visited[q[f]]=1;
        bfs(q[f++]);
    }
}
void main() {
    int v;
    printf("\n Enter the number of vertices:");
    scanf("%d",&n);
    for (i=1;i<=n;i++) {
        q[i]=0;
        visited[i]=0;
    }
    printf("\n Enter graph data in matrix form:\n");
    for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
       scanf("%d",&a[i][j]);
    printf("\n Enter the starting vertex:");
    scanf("%d",&v);
    bfs(v);
    printf("\n The node which are reachable are:\n");
    for (i=1;i<=n;i++)
      if(visited[i])
       printf("%d\t",i); else
       printf("\n Bfs is not possible");
}
```

Program 2: Output

```
Enter the number of vertices:4
Enter graph data in matrix form:
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
Enter the starting vertex:1
The node which are reachable are:
1	2	3	4
```

Program 1: Write a Program for N Queen's problem using Backtracking.

Program 1: Code

```c
#include <stdio.h>
#include <stdbool.h>

#define N 4   // Change N to any size of chessboard

// Function to print solution
void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
    printf("\n");
}

// Check if queen can be placed on board[row][col]
bool isSafe(int board[N][N], int row, int col) {
    int i, j;

    // Check row on left
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    // Check upper diagonal on left
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    // Check lower diagonal on left
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

// Solve N-Queens using backtracking
bool solveNQUtil(int board[N][N], int col) {
    // If all queens placed
    if (col >= N)
        return true;

    // Try placing queen in each row of this column
    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;

            if (solveNQUtil(board, col + 1))
                return true;

            // BACKTRACK
            board[i][col] = 0;
        }
    }
    return false;  // No place found
}

void solveNQ() {
    int board[N][N] = {0};

    if (!solveNQUtil(board, 0)) {
        printf("Solution does not exist\n");
        return;
    }

    printSolution(board);
}

int main() {
    printf("N-Queens Problem Solution (N = %d)\n\n", N);
    solveNQ();
    return 0;
}
```

Program 1: Output

```
N-Queens Problem Solution (N = 4)

 0  0  1  0
 1  0  0  0
 0  0  0  1
 0  1  0  0
```

Program 1: Write a program to implement Rabin-Karp method for pattern searching.

Program 2: Code

```c
#include <stdio.h>
#include <string.h>

#define d 256  // number of characters in input alphabet

// Rabin-Karp search function
void rabinKarp(char text[], char pattern[], int q) {
    int m = strlen(pattern);
    int n = strlen(text);
    int i, j;
    int p = 0; // hash value for pattern
    int t = 0; // hash value for text
    int h = 1;

    // h = pow(d, m-1) % q
    for (i = 0; i < m - 1; i++)
        h = (h * d) % q;

    // Calculate hash value for pattern and first window of text
    for (i = 0; i < m; i++) {
        p = (d * p + pattern[i]) % q;
        t = (d * t + text[i]) % q;
    }

    // Slide the pattern over text one by one
    for (i = 0; i <= n - m; i++) {
        // Check the hash values
        if (p == t) {
            // Check characters one by one
            for (j = 0; j < m; j++) {
                if (text[i + j] != pattern[j])
                    break;
            }
            if (j == m)
                printf("Pattern found at index %d\n", i);
        }

        // Calculate hash value for next window of text
        if (i < n - m) {
            t = (d * (t - text[i] * h) + text[i + m]) % q;

            // We might get negative value of t, convert it
            if (t < 0)
                t = (t + q);
        }
    }
}

int main() {
    char text[] = "ABCCDDAEFG";
    char pattern[] = "CDD";
    int q = 101; // A prime number

    printf("Text: %s\n", text);
    printf("Pattern: %s\n", pattern);

    rabinKarp(text, pattern, q);

    return 0;
}
```

Program 1: Output

```
Text: ABCCDDAEFG
Pattern: CDD
Pattern found at index 2
```