EXPT 1 : DICTIONARIES USING HASHING

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
class hashtab {
 private:
 struct DCT{
 int k; int val;
}a[10];
public:
int hashdivmethod(int);
int hashmulmethod(int);
void init();
void insert(int,int,int);
void display();
void size();
void search(int);
};
void hashtab::init()
{ for(int i=0;i<10;i++)
 { a[i].k=-1;
  a[i].val=-1;
 }}
 int hashtab::hashdivmethod(int num)
 { int hkey;
  hkey=num%10;
  return hkey; }
 int hashtab::hashmulmethod(int num)
 { int hkey;
  double A=0.6180;
  int p=1;
  hkey=floor(num*p*A);
  return hkey; }
  void hashtab::insert(int index,int key,int value)
  { int flag,i,count=0;
   flag=0;
   if(a[index].k==-1)
   { a[index].k=key;
    a[index].val=value; }
   else
   { i=0;
    while(i<10)
    { if(a[i].k!=-1)
     count++;
     i++; }
    if(count==10)
    { cout<<"\nHash Table is Full"; }
```

```
 for(i=index+1;i<10;i++)
 if(a[i].k==-1)
 { a[i].k=key;
  a[i].val=value;
  flag=1;
  break; }
 for(i=0;i<index&&flag==0;i++)
 if(a[i].k==-1)
 { a[i].k=key;
  a[i].val=value;
  flag=1;
  break;
 } } }
void hashtab::display()
{ int i;
 cout<<"\nThe Hash Table is \n";
 cout<<"\n_____";
 for(i=0;i<10;i++)
 {  cout<<"\n "<<i<<" "<<a[i].k<<" "<<a[i].val; }
 cout<<"\n_____"; }
 void hashtab::size()
 {  int len=0,i;
  for(i=0;i<10;i++)
  {  if(a[i].k!=-1)
   len++; }
  cout<<"\nThe size of dictionary is ";
  cout<<len;  }
 void hashtab::search(int search_key)
 {  int i,j;
  i=hashdivmethod(search_key);
  if(a[i].k==search_key)
  {  cout<<"\nThe Record is present at locaton "<<i;
   return; }
   if(a[i].k!=search_key)
   { for(j=i;j<10;j++)
         { if(a[j].k==search_key)
          { cout<<"\nThe Record is present ar location "<<j;
           return;
          } }
          for(j=0;j<i;j++)
          { if(a[j].k==search_key)
           { cout<<"\nThe Record is present at location "<<j;
            return;
           } }}
          else
          cout<<"\nThe Record is not present in the hash table";
         }
          void main()
```

```
{
 int key,value,hkey,search_key,choice;
 char ans;
 hashtab obj;
 clrscr();
 cout<<"\nDictionary Funcitons using Hashing";
 obj.init();
 do
 {
  cout<<"\n1.Insertion (Division Method) \n2.Insertion (Multiplication Method) \n3.Display
\n4.Search by Division Method \n5.Exit";
  cout<<"\nEnter your choice";
  cin>>choice;
  switch(choice)
  {
   case 1: cout<<"\nEnter the key";
           cin>>key;
           cout<<"\nEnter the value";
           cin>>value;
           hkey=obj.hashdivmethod(key);
           obj.insert(hkey,key,value);
           break;
   case 2: cout<<"\nEnter the key";
           cin>>key;
           cout<<"\nEnter the value";
           cin>>value;
           hkey=obj.hashmulmethod(key);
           obj.insert(hkey,key,value);
           break;
   case 3: obj.display(); obj.size(); break;
   case 4: cout<<"\nEnter the key for searching the record";
           cin>>search_key;
           obj.search(search_key);
           break;
   case 5: exit(0);
  }
  cout<<"\nDo u wish to continue (y/n) : ";
  ans=getch();
 }while(ans=='y');
 getch();
}
```

UNIVERSAL HASHING

```
#include<stdio.h>
#include<limits.h>
#include<stdlib.h>
#include<conio.h>
#define max 11
#define ver 2
```

```
int hashtable[ver][max];
int pos[ver];
void initable()
{
  for(int i=0;i<ver;i++)
  for(int j=0;j<max;j++)
  hashtable[i][j]=INT_MIN;
}
 int hash(int function,int key)
{
 switch(function)
{
 case 1: return (key%max);
 case 2: return (key/max)%max;
}
}
void place(int key,int tableId,int cnt,int n)
{
 if(cnt==n)
{
 printf("%d Un postitioned\n",key);
 printf("Cycle present Rehash \n");
 return;
}
 for(int i=0;i<ver;i++)
{
 pos[i]=hash(i+1,key);
 if(hashtable[i][pos[i]]==key)
 return;
}
if(hashtable[tableId][pos[tableId]]!=INT_MIN)
{
 int dis=hashtable[tableId][pos[tableId]];
 hashtable[tableId][pos[tableId]]=key;
 place(dis,(tableId+1)%ver,cnt+1,n);
}
else
 hashtable[tableId][pos[tableId]]=key;
}
 void printtable()
{
 printf("Final hashtables\n");
 for(int i=0;i<ver;i++)
 {
 for(int j=0;j<max;j++)
 (hashtable[i][j]==INT_MIN)?printf("-"):printf("%d",hashtable[i][j]);
 printf("\n");
}
```

```
}
void cuckoo(int keys[],int n)
{
 initable();
 int cnt=0;
 for(int i=0;i<n;i++)
 place(keys[i],0,cnt,n);
 printtable();
}
void main()
{
 int keys_1[]={20,50,53,75,100,67,105,3,36,39};
 int n=sizeof(keys_1)/sizeof(int);
 cuckoo(keys_1,n);
 int keys_2[]={20,50,53,75,100,67,105,3,36,39,6};
 int m=sizeof(keys_2)/sizeof(int);
 cuckoo(keys_2,m);
 getch();
}
```

EXPT 2 : AVL TREES

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#define FALSE 0
#define TRUE 1
typedef struct Node
{ int data;
 int BF;
 struct Node *left;
 struct Node *right;
}node;
class AVL
{ node *root;
 public:
 AVL()
 { root=NULL; }
 node *insert(int data,int *current)
 { root=create(root,data,current);
  return root; }
  node *create(node *root,int data,int *current);
  node *right_rotation(node *root,int *current);
  node *left_rotation(node *root,int *current);
  void display(node *root);
 };
  node *AVL::create(struct Node *root,int data,int *current)
  { node *temp1,*temp2;
   if(root==NULL)
```

```
{ root=new node;
 root->data=data;
 root->left=root->right=NULL;
 root->BF=0;
 *current=TRUE;
 return(root);
}
if(data<root->data)
{ root->left=create(root->left,data,current);
 if(*current)
 { switch(root->BF)
  {
   case 1: temp1=root->left;
                    if(temp1->BF==1)
                    { cout<<"\nSingle Rotation : LL";
                     root->left=temp1->right;
                     temp1->right=root;
                     root->BF=0;
                     root=temp1;
                    }
                     else
                    { cout<<"\nDouble rotation : LR";
                     temp2=temp1->right;
                     temp1->right=temp2->left;
                     temp2->left=temp1;
                     root->left=temp2->right;
                     temp2->right=root;
                    if(temp2->BF==1)
                    root->BF=-1;
                    else
                    root->BF=0;
                    if(temp2->BF==-1)
                    temp1->BF=1;
                    else
                    temp1->BF=0;
                    root=temp2;
                    }
                    root->BF=0;
                    *current=FALSE;
                    break;
   case 0: root->BF=1; break;
   case -1: root->BF=0;
                    *current=FALSE; break;
   } } }
  if(data>root->data)
  {
   root->right=create(root->right,data,current);
   if(*current!=NULL)
```

```
{
 switch(root->BF)
 {
      case 1: root->BF=0;
              *current=FALSE; break;
      case 0: root->BF=-1; break;
      case -1: temp1=root->right;
                        if(temp1->BF==-1)
                        {
                        cout<<"\nSingle rotation : RR";
                        root->right=temp1->left;
                        temp1->left=root;
                        root->BF=0;
                        root=temp1;
                        }
                        else
                        {
                         cout<<"\nDouble Rotation : RL ";
                         temp2=temp1->left;
                         temp1->left=temp2->right;
                         temp2->right=temp1;
                         root->right=temp2->left;
                         temp2->left=root;
                         if(temp2->BF==-1)
                         root->BF=1;
                         else
                         root->BF=0;
                         if(temp2->BF==1)
                         temp1->BF=-1;
                         else
                         temp1->BF=0;
                         root=temp2;
                        }
                        root->BF=0;
                        *current=FALSE;
      } } }
      return(root);
}
 void AVL::display(node *root)
 {
      if(root!=NULL)
      {
       display(root->left);
       cout<<root->data<<" ";
       display(root->right);
      } }
```

```
node *AVL::right_rotation(node *root,int *current)
{ node *temp1,*temp2;
 switch(root->BF)
 {
  case 1: root->BF=0; break;
  case 0: root->BF=-1; *current=FALSE; break;
  case -1: temp1=root->right;
              if(temp1->BF<=0)
              {
                    cout<<"\nSingle Rotation : RR ";
                    root->right=temp1->left;
                    temp1->left=root;
                    if(temp1->BF==0)
                    { root->BF=-1;
                     temp1->BF=1;
                     *current=FALSE;
                    }
                     else
                     {
                      root->BF=temp1->BF=0;
                     } root=temp1;
                     } else
                     {
                      cout<<"\nDouble Rotation : RL ";
                      temp2=temp1->left;
                      temp1->left=temp2->right;
                      temp2->right=temp1;
                      root->right=temp2->left;
                      temp2->left=root;
                     if(temp2->BF==-1)
                     root->BF=1;
                     else
                     root->BF=0;
                     if(temp2->BF==1)
                     temp1->BF=-1;
                     else
                     temp1->BF=0;
                     root=temp2;
                     temp2->BF=0;
                    } }
              return(root);
             }
 node *AVL::left_rotation(node *root,int *current)
 {
  node *temp1,*temp2;
  switch(root->BF)
  {
        case -1: root->BF=0; break;
```

```
            case 0: root->BF=1; *current=FALSE; break;
            case 1: temp1=root->left;
                    if(temp1->BF>=0)
                    {
                     cout<<"\nSingle Rotation : LL";
                     root->left=temp1->right;
                     temp1->right=root;
                     if(temp1->BF==0)
                     {
                      root->BF=1;
                      temp1->BF=-1;
                      *current=FALSE;
                     } else
                     { root->BF=temp1->BF=0;
                     } root=temp1;
                     } else
                     {
                      cout<<"\nDouble Roation : LR";
                      temp2=temp1->right;
                      temp1->right=temp2->left;
                      temp2->left=temp1;
                      root->left=temp2->right;
                      temp2->right=root;
                      if(temp2->BF==1)
                      root->BF=-1;
                      else
                      root->BF=0;
                      if(temp2->BF==-1)
                      temp1->BF=1;
                      else
                      temp1->BF=0;
                      root=temp2;
                      temp2->BF=0;
                     } }
                     return root;
                     }
void main()
{
 AVL obj;
 node *root=NULL;
 int current;
 clrscr();
 root=obj.insert(40,&current);
 root=obj.insert(50,&current);
 root=obj.insert(70,&current);
 cout<<endl;
 obj.display(root);
 cout<<endl;
```

```
root=obj.insert(30,&current);
cout<<endl;
obj.display(root);
root=obj.insert(20,&current);
cout<<endl;
obj.display(root);
root=obj.insert(45,&current);
cout<<endl;
obj.display(root);
root=obj.insert(25,&current);
cout<<endl;
obj.display(root);
root=obj.insert(10,&current);
cout<<endl;
obj.display(root);
root=obj.insert(5,&current);
cout<<endl;
obj.display(root);
root=obj.insert(22,&current);
cout<<endl;
obj.display(root);
root=obj.insert(1,&current);
cout<<endl;
obj.display(root);
root=obj.insert(35,&current);
cout<<"\n\nFinal AVL tree is : \n";
obj.display(root);
getch();
}
```

EXPT 4 : BINARY HEAP

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#define MAX 10
template<class T>
class Heap
{
 private:
 T arr[MAX];
 int n;
 public:
  Heap();
  void insert(T num);
  void makeheap();
  void heapsort();
  void display();
 };
```

```
template<class T>
Heap<T>::Heap()
{ n=0;
 for(int i=0;i<MAX;i++)
 arr[i]=0;
}
 template<class T>
 void Heap<T>::insert(T num)
 { if(n<MAX)
  { arr[n]=num;
   n++;
  }
   else
   cout<<"\Array is full";
  }
   template<class T>
   void Heap<T>::makeheap()
   { for(int i=1;i<n;i++)
    { T val=arr[i];
     int j=i;
     int f=(j-1)/2;
     while(j>0&&arr[f]<val)
     { arr[j]=arr[f];
      j=f;
      f=(j-1)/2;
     } arr[j]=val;
    } }
    template<class T>
    void Heap<T>::heapsort()
    { for(int i=n-1;i>0;i--)
     { T temp=arr[i];
      arr[i]=arr[0];
      int k=0;
      int j;
      if(i==1)
      j=-1;
      else
      j=1;
      if(i>2&&arr[2]>arr[1])
      j=2;
      while(j>=0&&temp<arr[j])
      {
            arr[k]=arr[j];
            k=j;
            j=2*k+1;
            if(j+1<=i-1&&arr[j]<arr[j+1])
            j++;
            if(j>i-1)
```

```
        j=-1;
}
        arr[k]=temp;
} }
template<class T>
void Heap<T>::display()
{
        for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";
        cout<<"\n";
}
        void main()
        {
         clrscr();
         Heap<int>Iobj;
         Heap<char>Cobj;
         Iobj.insert(10);
         Iobj.insert(25);
         Iobj.insert(30);
         Iobj.insert(90);
         Iobj.insert(7);
         Iobj.insert(21);
         Iobj.insert(3);
         Iobj.insert(23);
         cout<<"\nThe Elements are .."<<endl;
         Iobj.display();
         Iobj.makeheap();
         cout<<"\nHeapefied"<<endl;
         Iobj.display();
         Iobj.heapsort();
         cout<<"\nElements sorted by heap sort are ..."<<endl;
         Iobj.display();
         cout<<"\nSorting character type of elements";
         Cobj.insert('F');
         Cobj.insert('D');
         Cobj.insert('H');
         Cobj.insert('B');
         Cobj.insert('G');
         Cobj.insert('A');
         Cobj.insert('C');
         Cobj.insert('E');
         cout<<"\nThe Elements are .."<<endl;
         Cobj.display();
         Cobj.makeheap();
         cout<<"\nHeapefied"<<endl;
         Cobj.display();
         Cobj.heapsort();
         cout<<"\nElements sorted by heap sort are ..."<<endl;
```

```
            Cobj.display();
            getch();
           }
```

EXPT 5 : GRAPHS

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10
class Graph
{ private:
 int choice,n;
 int G[MAX][MAX];
 public:
 Graph();
 void insert_vertex();
 void delete_vertex();
 void find_vertex();
 void insert_edge();
 void delete_edge();
 void display();
 void create();
};
 void main()
 { Graph obj;
  int choice;
  char ch='y';
  clrscr();
  cout<<"\nProgram for graph creation";
  obj.create();
  obj.display();
  do
  { cout<<"\nEnter your choice";
   cout<<"\n1.Insertion of vertex \n2.Deletion of vertex \n3.Finding vertex \n4.Edge addition \n5.Edge
deletion \n6.Exit";
   cin>>choice;
   switch(choice)
   {
    case 1: obj.insert_vertex();
            obj.display(); break;
    case 2: obj.delete_vertex();
            obj.display(); break;
    case 3: obj.find_vertex();
            break;
    case 4: obj.insert_edge();
            obj.display(); break;
    case 5: obj.delete_edge();
            obj.display(); break;
```

```
 case 6: exit(0);
 }
 cout<<"Do you want to go to main menu?";
 ch=getche();
 }while(ch=='y');
}
 Graph::Graph()
 { for(int i=0;i<MAX;i++)
 for(int j=0;j<MAX;j++)
 G[i][j]=0;
 }
 void Graph::create()
 {
  int v1,v2;
  char ans='y';
  do
  {
   cout<<"\nEnter vertex v1 and v2";
   cin>>v1>>v2;
   G[v1][v2]=1;
   G[v2][v1]=1;
   cout<<"\nDo you want to insert more?";
   ans=getch();
  }while(ans=='y');
  }
   void Graph::insert_vertex()
   {
    int v1,v2;
    char ans='y';
    cout<<"\nEnter the vertex to be inserted";
    cin>>v1;
    do
    { cout<<"\nEnter neighbouring vertex";
          cin>>v2;
          G[v1][v2]=1;
          G[v2][v1]=1;
          cout<<"\nMore neighbouring vertex?";
          ans=getch();
    }while(ans=='y');
   }
    void Graph::display()
    { cout<<"\n";
          for(int i=0;i<MAX;i++)
          { for(int j=0;j<MAX;j++)
           { cout<<" "<<G[i][j];
           } cout<<"\n";
           } }
           void Graph::delete_vertex()
```

```
{ int i,v;
 cout<<"\nEnter the vertex to be deleted";
 cin>>v;
 for(i=0;i<MAX;i++)
 { G[v][i]=0;
  G[i][v]=0;
 } cout<<"\nThe vertex is deleted";
 }
  void Graph::find_vertex()
  {
   int v,i,flag=1;
   cout<<"\nEnter the vertex to be searched in the graph";
   cin>>v;
   for(i=0;i<MAX;i++)
   {
    if(G[v][i]==1)
    {
     flag=0;
     cout<<"\nNeighbouring vertex is"<<i;
    } }
    if(flag==1)
    cout<<"\nVertex is not present in the graph";
   }
     void Graph::insert_edge()
     {
      int v1,v2;
      cout<<"\nEnter the edge to be inserted by v1 and v2";
      cin>>v1>>v2;
      G[v1][v2]=1;
      G[v2][v1]=1;
     }
      void Graph::delete_edge()
      {
            int v1,v2;
            cout<<"\nEnter the edge to be deleted by v1 and v2";
            cin>>v1>>v2;
            G[v1][v2]=0;
            G[v2][v1]=0;
      }
```

EXPT 6 : DFS

```
#include<iostream.h>
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int cost[10][10],i,j,k,m,n,v,stk[10],top,visit[10],visited[10];
main()
{
```

```
 int m;
 cout<<"Enter the no of vertices";
 cin>>n;
 cout<<"Enter the no of edges";
 cin>>m;
 cout<<"Edges\n";
 for(k=1;k<=m;k++)
{
 cin>>i>>j;
 cost[i][j]=cost[j][i]=1;
}
 cout<<"Enter initial vertex";
 cin>>v;
 cout<<"Order of the visited vertices";
 cout<<v<<" ";
 visited[v]=1;
 k=1;
 while(k<n)
{
 for(j=n;j>=1;j--)
 if(cost[v][j]!=0&&visited[j]!=1&&visit[j]!=1)
{
 visit[j]=1;
 stk[top]=j;
 top++;
}
v=stk[--top];
cout<<v<<" ";
k++;
visit[v]=0;
visited[v]=1;
}
getch();
}
```

EXPT 7 : BFS

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#define TRUE 1
#define FALSE 0
template<class T>
class Lgraph
{
 private:
 typedef struct Gnode
 {
  T vertex;
```

```
 struct Gnode *next;
}node;
node *head[10];
int visited[10];
T Queue[10];
int front,rear;
public:
 void init();
 void create();
 void bfs(T);
};
template<class T>
void Lgraph<T>::init()
{
 int v1;
 for(v1=0;v1<10;v1++)
 visited[v1]=FALSE;
 front=rear=-1;
 for(v1=0;v1<10;v1++)
 head[v1]=NULL;
}
 template<class T>
 void Lgraph<T>::create()
 {
  T v1,v2;
  char ans='y';
  node *New,*first;
  cout<<"\nEnter the vertices no. beginning with 0";
  do
  {
   cout<<"\nEnter the edge of graph";
   cin>>v1>>v2;
   New=new node;
   if(New==NULL)
   cout<<"\nInsufficient Memory";
   New->vertex=v2;
   New->next=NULL;
   first=head[v1];
   if(first==NULL)
   head[v1]=New;
   else
   {
    while(first->next!=NULL)
    first=first->next;
    first->next=New;
   }
    New=new node;
    if(New==NULL)
```

```
 cout<<"\nInsufficient Memory";
 New->vertex=v1;
 New->next=NULL;
 first=head[v2];
 if(first==NULL)
 head[v2]=New;
else
 {
  while(first->next!=NULL)
  first=first->next;
  first->next=New;
 }
  cout<<"\nWant to add more edges (y/n) ?";
  ans=getche();
 }while(ans=='y');
}
 template<class T>
 void Lgraph<T>::bfs(T v1)
 {
  T i;
  node *first;
  Queue[++rear]=v1;
  while(front!=rear)
  {
   i=Queue[++front];
   if(visited[i]==FALSE)
   {
         cout<<endl<<i;
         visited[i]=TRUE;
   }
         first=head[i];
         while(first!=NULL)
         {
          if(visited[first->vertex]==FALSE)
          Queue[++rear]=first->vertex;
          first=first->next;
         } } }
  void main()
  {
         char ans;
         Lgraph<int>gr;
         int v1;
         clrscr();
         gr.init();
         gr.create();
         cout<<"\nEnter the vertex from which you want to traverse";
         cin>>v1;
         cout<<"\nThe BFS of Graph is ";
```

```
        gr.bfs(v1);
        getche();
    }
```

EXPT 8 : PRIMS

```
#include<iostream.h>
#include<conio.h>
#define sz 30
#define inf 32767
int vertex[sz];
class spt
{
 private:
 int g[sz][sz],nodes;
 public:
 spt();
 void prim();
 void getdata();
 void display();
};
struct mn
{
 int d,v;
}s[sz];
spt::spt()
{
 for(int i=0;i<sz;i++)
{
 vertex[i]=0;
 for(int j=0;j<sz;j++)
{
 g[i][j]=0;
}}
 vertex[0]=1;
}
 void spt::prim()
{
 int select[sz],i,j,k;
 int min,v1,v2,total=0;
 for(i=0;i<nodes;i++)
 select[i]=0;
 select[0]=1;
for(k=1;k<nodes;k++)
{
 min=inf;
 for(i=0;i<nodes;i++)
{
 for(j=0;j<nodes;j++)
```

```
{
 if(g[i][j]&&((select[i]&&!select[j])||(!select[i]&&select[j])))
{
 if(min>g[i][j])
{
 min=g[i][j];
 v1=i; v2=j;
}}}}
 select[v1]=select[v2]=1;
if(vertex[v2]==1)
{
 int t;
 t=v1; v1=v2; v2=t;
}
 vertex[v2]=1;
 s[v2].d=min;
 s[v2].v=v1;
 total=total+min;
}
cout<<"The total path length is ="<<total;
}
 void spt::display()
{
 s[0].d=0;
 s[0].v=0;
cout<<"\nVertex \t Known \t Weight \t Pr vt\n"<<endl;
for(int i=0;i<nodes;i++)
{
 cout<<i<<"\t1\t"<<s[i].d<<"\t"<<s[i].v<<endl;
}}
 void spt::getdata()
{
 int v1,v2,n,cost;
 cout<<"\nEnter the no of nodes";
 cin>>nodes;
 cout<<"\nEnter the no of edges";
 cin>>n;
 cout<<"\nEnter the edges and costs";
 for(int i=0;i<n;i++)
{
 cout<<"\nEnter edge v1 and v2 and cost:";
 cin>>v1>>v2>>cost;
 g[v1][v2]=g[v2][v1]=cost;
}}
 void main()
{
 spt obj;
 clrscr();
```

```
 cout<<"\nPrims Algorithm";
 obj.getdata();
 obj.prim();
 obj.display();
 getch();
}
```

EXPT 9 : KRUSKAL

```
#include<iostream.h>
#include<conio.h>
template<class T>
class Kruskal
{
 typedef struct graph
{
 int v1,v2;
 T cost,weight;
 char r;
}GR;
GR G[10];
public:
 int n,e,i;
 void create();
 void spt();
 void input();
 int min(int);
};
  int find(int v2,int parent[])
  {
   while(parent[v2]!=v2)
   {
    v2=parent[v2];
   }
   return v2;
  }
   void unionfn(int i,int j,int parent [])
   {
    if(i<j)
    parent[j]=i;
    else
    parent[i]=j;
   }
   template<class T>
   void Kruskal<T>::input()
   {
    cout<<"\nEnter total no of nodes ";
    cin>>n;
    cout<<"\nEnter total no of edges ";
```

```
 cin>>e;
}
template<class T>
void Kruskal<T>::create()
{
 for(int k=0;k<e;k++)
 {
  cout<<"\nEnter edge in (v1 v2) form ";
  cin>>G[k].v1>>G[k].v2;
  cout<<"\nEnter corresponding cost ";
  cin>>G[k].cost;
  G[k].weight=G[k].cost;
  G[k].r='r';
 }
}
template<class T>
int Kruskal<T>::min(int n)
{
 int i,small,pos;
 small=999;
 pos=-1;
 for(i=0;i<n;i++)
 {
 if(G[i].cost<small)
  {
   small=G[i].cost;
   pos=i;
  }
 }
  return pos;
 }
 template<class T>
 void Kruskal<T>::spt()
 {
  int count,v1,v2,i,j,pos,parent[10],m;
  T sum;
  count=0;
  sum=0;
  for(i=0;i<n;i++)
  parent[i]=i;
  while(count!=n-1)
  {
   pos=min(e);
   if(pos==-1)
   break;
   v1=G[pos].v1;
   v2=G[pos].v2;
   i=find(v1,parent);
```

```
    j=find(v2,parent);
    if(i!=j)
    {
     count++;
     sum+=G[pos].cost;
     unionfn(i,j,parent);
    }
     G[pos].r='a';
     G[pos].cost=999;
    }
     if(count==n-1)
     {
           cout<<"\nThe cost of Spanning tree is "<<sum;
           cout<<"\nSpanning Tree is \n Edge \t Weight \tAction \n";
           for(i=0;i<e;i++)
            {
             cout<<G[i].v1<<"-"<<G[i].v2<<"\t"<<G[i].weight;
              if(G[i].r=='a')
              cout<<"\tAccepted"<<endl;
             else
              cout<<"\tRejected"<<endl;
            }
           }
           }
    void main()
    {
           Kruskal<int>obj;
           clrscr();
           cout<<"Graph creation";
           obj.input();
           obj.create();
           obj.spt();
    }
```

EXPT 10 : DIJKSTRA

```
#include<iostream.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
#define max 32767
#define m 20
int V;
int mindistance(int dist[],int sptset[])
{
 int min=max,min_index;
 for(int v=0;v<V;v++)
  if(sptset[v]==0&&dist[v]<=min)
   min=dist[v], min_index=v;
```

```
 return min_index;
}
void printsolution(int dist[],int n)
{
 cout<<"Vertex Distance from source \n";
 for(int i=0;i<n;i++)
 cout<<i<<"\t\t"<<dist[i]<<endl;
 }
void dijkstra(int graph[m][m],int src)
{
 int dist[m];
 int sptset[m];
 for(int i=0;i<V;i++)
 { dist[i]=max; sptset[i]=0;
  dist[src]=0;
 }
 for(int count=0;count<V-1;count++)
{
  int u=mindistance(dist,sptset);
  sptset[u]=1;
  for(int v=0;v<V;v++)
   if(!sptset[v]&&graph[u][v]&&dist[u]!=max&&dist[u]+graph[u][v]<dist[v])
    dist[v]=dist[u]+graph[u][v];
}
printsolution(dist,V);
}
int main()
{
 cout<<"Enter number of nodes in graph  ";
 int graph[m][m];
 cin>>V;
 cout<<"\nEnter the adjacency matrix of the graph \n";
 for(int i=0;i<V;i++)
{
 for(int j=0;j<V;j++)
 cin>>graph[i][j];
}
 dijkstra(graph,0);
return 0;
}
```

EXPT 11 : MOOYRE BOOYRE

```
#include<iostream.h>
#include<string.h>
#include<stdlib.h>
#define MAX 50
void init(char *str,int size, int a[])
{
```

```
int i;
for(i=0;i<MAX;i++)
a[i]=-1;
for(i=0;i<size;i++)
a[(int)str[i]]=i;
}
int maximum(int a,int b)
{
 return(a>b)?a:b;
}
void search(char *txt,char *pat)
{
int m=strlen(pat);
int n=strlen(txt);
int a[MAX];
init(pat,m,a);
int s=0;
while(s<=(n-m))
{
 int j=m-1;
 while(j>=0&&pat[j]==txt[s+j])
 j--;
if(j<0)
{
 cout<<"Pattern occurs at position "<<s+1;
 s+=(s+m<n)?m-a[txt[s+m]]:1;
}
 else
 s+=maximum(1,j-a[txt[s+j]]);
}
}
void main()
{
 char txt[50],pat[10];
 cout<<"\nEnter the text ";
 cin>>txt;
 cout<<"\nEnter the pattern ";
 cin>>pat;
 search(txt,pat);
}
```

EXPT 12 : KNUTH MORIS PRATT

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
int *create_prefix_table(char p[10])
{
```

```
int *prefix_table;
int m=strlen(p);
int k=0;
prefix_table=(int *)malloc(m*sizeof(int));
prefix_table[0]=0;
for(int q=1;q<m;q++)
{
 while(k>0&&p[k]!=p[q])
 k=prefix_table[k-1];
 if(p[k]==p[q])
 k++;
 prefix_table[q]=k;
}
 return prefix_table;
}
 void kmp_match(char t[50],char p[10])
 {
  int *prefix_table;
  int i;
  int j=0;
  int n=strlen(t);
  int m=strlen(p);
  prefix_table=create_prefix_table(p);
  cout<<"\n_____";
  cout<<endl<<"Prefix array is : \n";
  for(i=0;i<m;i++)
  cout<<prefix_table[i]<<" ";
  cout<<"\n_____";
  cout<<endl;
  for(i=0;i<n;i++)
  {
   cout<<"\ni= "<<i<<"j= "<<j;
   cout<<"\nComparing "<<t[i]<<" and "<<p[j];
   while(j>0&&p[j]!=t[i])
   j=prefix_table[j-1];
   if(p[j]==t[i])
   {
    cout<<"\nMatch found at "<<"i = "<<i<<"j = "<<j;
    j++;
   }
   if(j==m)
   {
    cout<<"\n\tPattern is present in the text at : ";
    cout<<i-m+1<<" position. "<<endl;
    j=prefix_table[j-1];
   }
  }
 }
```

```
int main()
{
 char t[50];
 char p[10];
 clrscr();
 cout<<"Enter the T string : ";
 cin>>t;
 cout<<"\nEnter the P string : ";
 cin>>p;
 kmp_match(t,p);
 return 0;
}
```
