Program 1

/**

To implement functions of Dictionary using Hashing (division method, Multiplication

method, Universal hashing)

**/

AIM: To implement functions of Dictionary using Hashing (division method, Multiplication method & Universal hashing)

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
void insert();
void display();
int search(int);
void deleted();
int HT[20],size,index,key,flag=0,s1=0,i1=0,j;
int main()
{
int i,ch;
//clrscr();
printf("Enter Hash Table size");
scanf("%d",&size);
for(i=0;i<size;i++)
{

HT[i]=-1;
}
do
{
printf("\n_______\n");
printf("\n1 Insert\n");
printf("\n2 Remove\n");
printf("\n3 search\n");
printf("\n4 Display\n");
printf("\n5Exit\n");
printf("\nEnter your choice\n");
scanf("%d",&ch);
switch(ch)
```

```
{

case 1:insert();break;
case 2:deleted();break;
case 3:printf("Enter searched key");
        scanf("%d",&key);
s1=search(key);
if(s1==1)
{
printf("\n Key is found\n");
}
else
printf("\n Key is not found\n");
break;
case 4:display();break;
case 5: exit(0); break;
default:printf("\n invalid choice");
}
}while(ch!=5);
return 0;
}
void insert()
{
printf("Enter key");
        scanf("%d",&key);
index=key%size;
flag=0;
i1=search(key);
if(i1==1)
{
printf("duplicate\n");
return;
}
else
{
if(HT[index]==-1)
{
HT[index]=key;
flag=1;
}
else
{
j=index+1;
while(1)
{
if(HT[j]==-1)
{
flag=1;
index=j;
break;
}
if(j==size)
j=0;
else if(j==index)
break;
else
j++;
}
}
if(flag==1)
{
printf("The key element value inserted in%d",index);
HT[index]=key;
}
else
printf("Hash Table is FuLL....!");
}
}
void deleted()
{
printf("Enter key to be Delete");
          scanf("%d",&key);
index=key%size;
flag=0;
if(HT[index]==key)
{
HT[index]=-1;
flag=1;
}
else
{
j=index+1;
while(1)
{
if(HT[j]==key)
{
flag=1;
index=j;
break;
}
if(j==size)
j=0;
```

```
else if(j==index)
break;
else
j++;
}
}
if(flag==1)
{
printf("The key element value  to be deleted\n");
HT[index]=-1;
}
else
printf("Hash Table is Empty....!");
}

int search(int key)
{
index=key%size;
flag=0;
if(HT[index]==key)
{

flag=1;
return(1);
}
else
{
j=index+1;
while(1)
{
if(HT[j]==key)
{
flag=1;
return(1);
}
if(j==size)
j=0;
else if(j==index)
return(0);
else
j++;
}
}
}
void display()
{
    int i;
printf("Hash Table elements are\n");
for(i=0;i<size;i++)
{
printf("%d\t",HT[i]);
}
printf("\n");
}
```

OUTPUT:

```
Enter Hash Table Size5

-----------------------

    1   Insert

    2   Remove

    3   Search

    4   Display

    5   Exit

    Enter your choice

1

Enter key 5

The key element value inserted in0

-----------------------

    1   Insert

    2   Remove

    3   Search

    4   Display

    5   Exit

    Enter your choice

1

Enter key 9

The key element value inserted in4

-----------------------

-----------------------

    1   Insert

    2   Remove
    3   Search
    4   Display
    5   Exit
    Enter your choice
1
Enter key 3
The key element value inserted in3
-----------------------
-----------------------
    1   Insert
    2   Remove
    3   Search
    4   Display
    5   Exit
    Enter your choice
1
Enter key 7
The key element value inserted in1
-----------------------
-----------------------
    1   Insert
    2   Remove
    3   Search
    4   Display
    5   Exit
    Enter your choice
1
```

```
Enter key 6

The key element value inserted in2

-----------------------

    1   Insert

    2   Remove

    3   Search

    4   Display

    5   Exit

    Enter your choice

4

Hash Table elements are

5       6       7       3       9

-----------------------

    1   Insert

    2   Remove

    3   Search

    4   Display

    5   Exit

    Enter your choice

2

Enter Key to be De;eted7

The Key element value to be deleted


-----------------------

    1   Insert

    2   Remove

    3   Search
    4   Display

    5   Exit

    Enter your choice

4

Hash Table elements are

5       6       -1      4       9

-----------------------

    1   Insert

    2   Remove

    3   Search

    4   Display

    5   Exit

    Enter your choice

5
```

Program 2

/**

To perform various operations i.e, insertions and deletions on AVL trees

**/

AIM : To perform various operations i.e, insertions and deletions on AVL trees

SOURCE CODE:

```
#include<stdio.h>
#include<malloc.h>
#define CHANGED 0
#define BALANCED 1
int height;
struct bnode
{
int data,bfactor;
struct bnode *left,*right;
};
typedef struct bnode node;
node* getnode()
{
int size;
node* newnode;
size=sizeof(node);
newnode=(node*)malloc(size);
return(newnode);
}
void copynode(node *r,int data)
{
r->data=data;
r->left=NULL;
r->right=NULL;
r->bfactor=0;
}
void releasenode(node *p)
{
free(p);
}
node* searchnode(node *root,int data)
```

```
{
if(root!=NULL)
if(data<root->data)
root=searchnode(root->left,data);
else if(data>root->data)
root=searchnode(root->right,data);
return(root);
}
void lefttoleft(node **Pptr,node **Aptr)
{
node *p=*Pptr,*A=*Aptr;
printf("\nLeft to left AVL Rotation\n");
p->left=A->right;
A->right=p;
if(A->bfactor==0)
{
p->bfactor=1;
A->bfactor=-1;
height=BALANCED;
}
else
{
p->bfactor=0;
A->bfactor=0;
}
p=A;
*Pptr=p;
*Aptr=A;
}

void lefttoright(node **Pptr,node **Aptr, node **Bptr)
{
node *p=*Pptr, *A=*Aptr, *B=*Bptr;
printf("\n Left to Right AVL Rotation \n");
B=A->right;
A->right=B->left;
B->left=A;
p->left=B->right;
B->right=p;
if(B->bfactor==1)
p->bfactor=-1;
else
p->bfactor=0;
if(B->bfactor==-1)
A->bfactor=1;
else
```

```
A->bfactor=0;
B->bfactor=0;
p=B;
*Pptr=p;
*Aptr=A;
*Bptr=B;
}

void righttoright(node **Pptr, node **Aptr)
{
node *p=*Pptr, *A=*Aptr;
printf("\n Right to Right AVL Rotation \n");
p->right=A->left;
A->left=p;
if(A->bfactor==0)
{
p->bfactor=-1;
A->bfactor=1;
height=BALANCED;
}
else
{
p->bfactor=0;
A->bfactor=0;
}
p=A;
*Pptr=p;
*Aptr=A;
}

void righttoleft(node **Pptr, node **Aptr, node **Bptr)
{
node *p=*Pptr, *A=*Aptr, *B=*Bptr;
printf("\n Right to Left AVL Rotation \n");
B=A->left;
A->left=B->right;
B->right=A;
p->right=B->left;
B->left=p;
if(B->bfactor==-1)
p->bfactor=1;
else
p->bfactor=0;
if(B->bfactor==1)
A->bfactor=-1;
else
```

```
A->bfactor=0;
B->bfactor=0;
p=B;
*Pptr=p;
*Aptr=A;
*Bptr=B;
}

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

```
if(height==CHANGED)
{
switch(p->bfactor)
{
case 1:
p->bfactor=0;    //left heavy trees
height=BALANCED;
break;
case 0:
p->bfactor=-1;   //balanaced trees
break;
case -1:
A=p->right;        //right heavy trees
if(A->bfactor==-1)
righttoright(&p,&A);
else
righttoleft(&p,&A,&B);
height=CHANGED;
break;
}
}
}
return(p);
}

void del(node **N,node **C)
{
node *T,*A,*B;
node **p;
T=(*N);
if((*N)->right!=NULL)
{
del(&((*N)->right),C);
if(height==CHANGED)
{
p=N;
switch((*p)->bfactor)
{
case -1:
(*p)->bfactor=0;
break;
case 0:
(*p)->bfactor=1;
height=BALANCED;
break;
case 1:
```

```
A=(*p)->left;
if(A->bfactor>=0)
lefttoleft(p,&A);
else
lefttoright(p,&A,&B);
break;
}
}
}
else
{
(*C)->data=(*N)->data;
(*N)=(*N)->left;
releasenode(T);
height=CHANGED;
}
}

void deletenode(int data,node **p)
{
node *A,*B,*C;
if(*p==NULL)
{
printf("\nAVL Tree is empty");
return;
}
if(data<(*p)->data)
{
deletenode(data,&((*p)->left));
if(height==CHANGED)
{
switch((*p)->bfactor)
{
case 1:
(*p)->bfactor=0;
break;
case 0:
(*p)->bfactor=1;
height=BALANCED;
break;
case -1:
A=(*p)->right;
if(A->bfactor<=0)
righttoright(p,&A);
else
righttoleft(p,&A,&B);
```

```
break;
}
}
}
else if(data>(*p)->data)
{
deletenode(data,&((*p)->right));
if(height==CHANGED)
{
switch((*p)->bfactor)
{
case -1:
(*p)->bfactor=0;
break;
case 0:
(*p)->bfactor=1;
height=BALANCED;
break;
case 1:
A=(*p)->left;
if(A->bfactor>=0)
lefttoleft(p,&A);
else
lefttoright(p,&A,&B);
break;
}
}
}
else
{
C=*p;
if(C->right==NULL)
{
*p=C->left;
height=CHANGED;
releasenode(C);
}
else if(C->left==NULL)
{
*p=C->right;
height=CHANGED;
releasenode(C);
}
else
{
del(&(C->left),&C);
```

```
if(height==CHANGED)
{
switch((*p)->bfactor)
{
case 1:
(*p)->bfactor=0;
break;
case 0:
(*p)->bfactor=-1;
height=BALANCED;
break;
case -1:
A=(*p)->right;
if(A->bfactor<=0)
righttoright(p,&A);
else
righttoleft(p,&A,&B);
break;
}
}
}
}
}

void inorder(node *root)
{
if(root==NULL)
return;
inorder(root->left);
printf("%4d",root->data);
inorder(root->right);
}


int main()
{
int data,ch,choice='y';
node *root=NULL;
printf("\nBasic operations in an AVL Tree...");
printf("\n1.Insert a node in the AVL Tree");
printf("\n2.Delete a node in the AVL Tree");
printf("\n3.View the AVL Tree");
printf("\n4.Exit");
while((choice=='y')||(choice=='Y'))
{
printf("\n");
```

```
fflush(stdin);
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nEnter the value to be inserted:");
scanf("%d",&data);
if(searchnode(root,data)==NULL)
root=insertnode(data,root);
else
printf("\nData already exists");
break;
case 2:
printf("Enter the value to be deleted:");
scanf("%d",&data);
if(searchnode(root,data)!=NULL)
deletenode(data,&root);
else
printf("\nElement to be deleted is not found");
break;
case 3:
if(root==NULL)
{
printf("\nAVL Tree is Emoty\n");
continue;
}
printf("\nInorder Traversal of the AVL Tree:");
inorder(root);
break;
default:
printf("\nEnd of run of your program...");
releasenode(root);
return 0;
}
}
}
```

OUTPUT:

```
Basic operations in an AVL Tree...
1.Insert a node in the AVL Tree
2.Delete a node in the AVL Tree
3.View the AVL Tree
4.Exit
1
Enter the value to be inserted:23
1
Enter the value to be inserted:15
3
Inorder Traversal of the AVL Tree:  15  23

Enter the value to be inserted:13
Left to left AVL Rotation
3
Inorder Traversal of the AVL Tree:  13  15  23

Enter the value to be inserted:27
1
Enter the value to be inserted:20
1
Enter the value to be inserted:18
Right to Left AVL Rotation
Enter the value to be inserted:32
Right to Right AVL Rotation
3
Inorder Traversal of the AVL Tree:  13  15  18  20  23  27  32
Enter the value to be inserted:9
1
Enter the value to be inserted:12
Left to Right AVL Rotation
3
Inorder Traversal of the AVL Tree:   9  12  13  15  18  20  23  27  32
```

Program 3

/**

To perform various operations i.e., insertions and deletions on 2-3 trees

**/

AIM: To perform various operations i.e., insertions and deletions on 2-3 trees.

SOURCE CODE:

```
/*Program of insertion and deletion in B tree*/
#include<stdio.h>
#include <conio.h>
#include<stdlib.h>
#define M 5

struct node{
int n; /* n < M No. of keys in node will always less than order of B
tree */
int keys[M-1]; /*array of keys*/
struct node *p[M]; /* (n+1 pointers will be in use) */
}*root=NULL;

enum KeyStatus { Duplicate,SearchFailure,Success,InsertIt,LessKeys };

void insert(int key);
void display(struct node *root,int);
void DelNode(int x);
void search(int x);
enum KeyStatus ins(struct node *r, int x, int* y, struct node ** u);
int searchPos(int x,int *key_arr, int n);
enum KeyStatus del(struct node *r, int x);

int main()
{
int key;
int choice;
printf("Creation of B tree for node %d\n",M);
while(1)
{
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Search\n");
printf("4.Display\n");
printf("5.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);

switch(choice)
{
case 1:
printf("Enter the key : ");
scanf("%d",&key);
insert(key);
break;
case 2:
printf("Enter the key : ");
scanf("%d",&key);
DelNode(key);
break;
case 3:
printf("Enter the key : ");
scanf("%d",&key);
search(key);
break;
case 4:
printf("Btree is :\n");
display(root,0);
break;
case 5:
exit(1);
default:
printf("Wrong choice\n");
break;
}/*End of switch*/
}/*End of while*/
//return 0;
}/*End of main()*/

void insert(int key)
{
struct node *newnode;
int upKey;
enum KeyStatus value;
value = ins(root, key, &upKey, &newnode);
```

```
if (value == Duplicate)
printf("Key already available\n");
if (value == InsertIt)
{
struct node * uproot = root;
root=(struct node *)malloc(sizeof(struct node));
root->n = 1;
root->keys[0] = upKey;
root->p[0] = uproot;
root->p[1] = newnode;
}/*End of if */
}/*End of insert()*/

enum KeyStatus ins(struct node *ptr, int key, int *upKey,struct node **newnode)
{
struct node *newPtr, *lastPtr;
int pos, i, n,splitPos;
int newKey, lastKey;
enum KeyStatus value;
if (ptr == NULL)
{
*newnode = NULL;
*upKey = key;
return InsertIt;
}
n = ptr->n;
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
return Duplicate;
value = ins(ptr->p[pos], key, &newKey, &newPtr);
if (value != InsertIt)
return value;
/*If keys in node is less than M-1 where M is order of B tree*/
if (n < M - 1)
{
pos = searchPos(newKey, ptr->keys, n);
/*Shifting the key and pointer right for inserting the new key*/
for (i=n; i>pos; i--)
{
ptr->keys[i] = ptr->keys[i-1];
ptr->p[i+1] = ptr->p[i];
}
/*Key is inserted at exact location*/
ptr->keys[pos] = newKey;
ptr->p[pos+1] = newPtr;
++ptr->n; /*incrementing the number of keys in node*/
return Success;
}/*End of if */
/*If keys in nodes are maximum and position of node to be inserted is
last*/
if (pos == M - 1)
{
lastKey = newKey;
lastPtr = newPtr;
}
else /*If keys in node are maximum and position of node to be inserted
is not last*/
{
lastKey = ptr->keys[M-2];
lastPtr = ptr->p[M-1];
for (i=M-2; i>pos; i--)
{
ptr->keys[i] = ptr->keys[i-1];
ptr->p[i+1] = ptr->p[i];
}
ptr->keys[pos] = newKey;
ptr->p[pos+1] = newPtr;
}
splitPos = (M - 1)/2;
(*upKey) = ptr->keys[splitPos];

(*newnode)=(struct node *)malloc(sizeof(struct node));/*Right node after split*/
ptr->n = splitPos; /*No. of keys for left splitted node*/
(*newnode)->n = M-1-splitPos;/*No. of keys for right splitted node*/
for (i=0; i < (*newnode)->n; i++)
{
(*newnode)->p[i] = ptr->p[i + splitPos + 1];
if(i < (*newnode)->n - 1)
(*newnode)->keys[i] = ptr->keys[i + splitPos + 1];
else
(*newnode)->keys[i] = lastKey;
}
(*newnode)->p[(*newnode)->n] = lastPtr;
return InsertIt;
}/*End of ins()*/

void display(struct node *ptr, int blanks)
{
```

```
if (ptr)
{
int i;
for(i=1;i<=blanks;i++)
printf(" ");
for (i=0; i < ptr->n; i++)
printf("%d ",ptr->keys[i]);
printf("\n");
for (i=0; i <= ptr->n; i++)
display(ptr->p[i], blanks+10);
}/*End of if*/
}/*End of display()*/

void search(int key)
{
int pos, i, n;
struct node *ptr = root;
printf("Search path:\n");
while (ptr)
{
n = ptr->n;
for (i=0; i < ptr->n; i++)
printf(" %d",ptr->keys[i]);
printf("\n");
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
{
printf("Key %d found in position %d of last dispalyednode\n",key,i);
return;
}
ptr = ptr->p[pos];
}
printf("Key %d is not available\n",key);
}/*End of search()*/

int searchPos(int key, int *key_arr, int n)
{
int pos=0;
while (pos < n && key > key_arr[pos])
pos++;
return pos;
}/*End of searchPos()*/

void DelNode(int key)
```

```
{
struct node *uproot;
enum KeyStatus value;
value = del(root,key);
switch (value)
{
case SearchFailure:
printf("Key %d is not available\n",key);
break;
case LessKeys:
uproot = root;
root = root->p[0];
free(uproot);
break;
}/*End of switch*/
}/*End of delnode()*/

enum KeyStatus del(struct node *ptr, int key)
{
int pos, i, pivot, n ,min;
int *key_arr;
enum KeyStatus value;
struct node **p,*lptr,*rptr;

if (ptr == NULL)
return SearchFailure;
/*Assigns values of node*/
n=ptr->n;
key_arr = ptr->keys;
p = ptr->p;
min = (M - 1)/2;/*Minimum number of keys*/

pos = searchPos(key, key_arr, n);
if (p[0] == NULL)
{
if (pos == n || key < key_arr[pos])
return SearchFailure;
/*Shift keys and pointers left*/
for (i=pos+1; i < n; i++)
{
key_arr[i-1] = key_arr[i];
p[i] = p[i+1];
}
return --ptr->n >= (ptr==root ? 1 : min) ? Success : LessKeys;
```

```
}/*End of if */

if (pos < n && key == key_arr[pos])
{
struct node *qp = p[pos], *qp1;
int nkey;
while(1)
{
nkey = qp->n;
qp1 = qp->p[nkey];
if (qp1 == NULL)
break;
qp = qp1;
}/*End of while*/
key_arr[pos] = qp->keys[nkey-1];
qp->keys[nkey - 1] = key;
}/*End of if */
value = del(p[pos], key);
if (value != LessKeys)
return value;

if (pos > 0 && p[pos-1]->n > min)
{
pivot = pos - 1; /*pivot for left and right node*/
lptr = p[pivot];
rptr = p[pos];
/*Assigns values for right node*/
rptr->p[rptr->n + 1] = rptr->p[rptr->n];
for (i=rptr->n; i>0; i--)
{
rptr->keys[i] = rptr->keys[i-1];
rptr->p[i] = rptr->p[i-1];
}
rptr->n++;
rptr->keys[0] = key_arr[pivot];
rptr->p[0] = lptr->p[lptr->n];
key_arr[pivot] = lptr->keys[--lptr->n];
return Success;
}/*End of if */
if (pos > min)
{
pivot = pos; /*pivot for left and right node*/
lptr = p[pivot];
rptr = p[pivot+1];
```

```
/*Assigns values for left node*/
lptr->keys[lptr->n] = key_arr[pivot];
lptr->p[lptr->n + 1] = rptr->p[0];
key_arr[pivot] = rptr->keys[0];
lptr->n++;
rptr->n--;
for (i=0; i < rptr->n; i++)
{
rptr->keys[i] = rptr->keys[i+1];
rptr->p[i] = rptr->p[i+1];
}/*End of for*/
rptr->p[rptr->n] = rptr->p[rptr->n + 1];
return Success;
}/*End of if */

if(pos == n)
pivot = pos-1;
else
pivot = pos;

lptr = p[pivot];
rptr = p[pivot+1];
/*merge right node with left node*/
lptr->keys[lptr->n] = key_arr[pivot];
lptr->p[lptr->n + 1] = rptr->p[0];
for (i=0; i < rptr->n; i++)
{
lptr->keys[lptr->n + 1 + i] = rptr->keys[i];
lptr->p[lptr->n + 2 + i] = rptr->p[i+1];
}
lptr->n = lptr->n + rptr->n +1;
free(rptr); /*Remove right node*/
for (i=pos+1; i < n; i++)
{
key_arr[i-1] = key_arr[i];
p[i] = p[i+1];
}
return --ptr->n >= (ptr == root ? 1 : min) ? Success : LessKeys;
}/*End of del()*/
```

OUTPUT:

```
Creation of B tree for node 5
1.Insert
2.Delete
3.Search
4.Display
5.Quit
Enter your choice : 1
Enter the key : 10
1.Insert
2.Delete
3.Search
4.Display
5.Quit
Enter your choice : 1
Enter the key : 20
1.Insert
2.Delete
3.Search
4.Display
5.Quit
Enter your choice : 4
Btree is :
10 20
1.Insert
2.Delete
3.Search
4.Display
5.Quit
```

```
Enter your choice : 2
Enter the key : 10
1.Insert
2.Delete
3.Search
4.Display
5.Quit
Enter your choice : 4
Btree is :
20
1.Insert
2.Delete
3.Search
4.Display
5.Quit
Enter your choice :5
```

Program 4

/**

To implement operations on binary heap.

**/

AIM: To implement operations on binary heap

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<stdlib.h>
#define arraylength 20
int heapsize=0,heap[arraylength];
void enqueue(int element)
{
int currentnode;
if(heapsize==arraylength-1)
{
printf("Array is full");
return;
}
currentnode=++heapsize;
while(currentnode!=0&&heap[currentnode/2]<element)
{
heap[currentnode]=heap[currentnode/2];
currentnode/=2;
}
heap[currentnode]=element;
}
void dequeue()
{
int lastelement,currentnode,child;
if(heapsize==0)
{
printf("Priority queue is empty");
return;
}
heap[0]=0;
lastelement=heap[heapsize--];
```

```
currentnode=1;
child=2;
while(child<=heapsize)
{
if(child<heapsize&&heap[child]<heap[child+1])
child++;
if(lastelement>=heap[child])
break;
heap[currentnode]=heap[child];
currentnode=child;
child*=2;
}
heap[currentnode]=lastelement;
}
void display()
{
int i;
for(i=0;i<arraylength;i++)
{
printf("%5d",heap[i]);
}
}
int main()
{
int opt,ele;
//clrscr();
while(1)
{
printf("\n............");
printf("\n1.Enqueue");
printf("\n2.Dequeue");
printf("\n3.Display");
printf("\n4.Exit");
printf("\n...........");
printf("\nEnter ur option:");
scanf("%d",&opt);
switch(opt)
{
case 1:
printf("Enter the element to insert:");
scanf("%d",&ele);
enqueue(ele);
break;
case 2:
```

```
dequeue();
printf("Element is deleted");
break;
case 3:
display();
break;
case 4:
exit(0);
}
getch();
}
return 0;
}
```

OUTPUT:

```
.............
1.Enqueue
2.Dequeue
3.Display
4.Exit
............
Enter ur option:1
Enter the element to insert: 10
.............
1.Enqueue
2.Dequeue
3.Display
4.Exit
............
Enter ur option:1
Enter the element to insert: 5
.............
1.Enqueue
2.Dequeue
3.Display
4.Exit
............
Enter ur option:1
Enter the element to insert: 18
.............
1.Enqueue
2.Dequeue
3.Display
4.Exit
............
Enter ur option:1
Enter the element to insert: 24
.............
1.Enqueue
2.Dequeue
3.Display
4.Exit
............
Enter ur option:3
24   18   10    5    0    0    0    0    0     0    0    0    0    0    0    0     0    0    0    0
```

Program 5

/***

To implement operations on graphs

i) vertex insertion

ii) Vertex deletion

iii) finding vertex

iv)Edge addition and deletion

***/

AIM: To implement operations on graphs

i) vertex insertion   ii) Vertex deletion   iii) finding vertex   iv)Edge addition and deletion

Source Code:

```
/*Program to create,insert,delete and view the adjecency matrix*/
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define VSIZE 20

int checkWt();

int checkDir();

void insertVertex ();

void deleteVertex(int vDel);

void insertEdge(int vStart,int vEnd);

void deleteEdge(int vStart, int vEnd);
void createGraph();
void viewGraph();
void display_menu();
int nVertex,adjMat[VSIZE][VSIZE];
int main()
{
char choice='y';
int ch,vs,ve,vd;
//clrscr();
display_menu();
while((choice=='y')||(choice=='y'))
{        printf("\n?");
fflush(stdin);
scanf("%d",&ch);
switch(ch)
{
case 0 :display_menu();
break;
case 1 :createGraph();
break;
case 2 :insertVertex();
break;
case 3 :printf("\n enter the starting & ending vertex to insert an edge :");
scanf("%d %d",&vs,&ve);
insertEdge(vs,ve);
break;
case 4 :printf("\n enter the vertex to delete :");
scanf("%d",&vd);
deleteVertex(vd);
break;
case 5 :printf("\n enter the starting & ending vertex to delete an edge :");
scanf("%d%d",&vs,&ve);
break;
case 6 :viewGraph();
break;
case 7 :printf("\n end of run of your program...........");
exit(0);
}
}
return 0;
}
void insertVertex()
{	int rc;
nVertex++;
for(rc=0;rc<nVertex;rc++)
adjMat[rc] [nVertex-1]=adjMat[nVertex-1] [rc]=0;
```

```
}
void insertEdge(int vStart,int vEnd)
{	int ie;
if(vStart>nVertex||vStart<1||vEnd>nVertex||vEnd<1)
return;
printf("enter weight of the Edge from v%d to v%d :", vStart ,vEnd);
scanf("%d",&adjMat [vStart-1] [vEnd-1]);
}
void deleteVertex(int vDel)
{	int r,c;
if(vDel>nVertex || vDel<1)
return;
for(r=vDel-1;r<nVertex; r++)
for(c=0;c<nVertex;c++)
adjMat[r][c]=adjMat[r+1][c];
for(c=vDel-1;r<nVertex;c++)
for(r=0;r<nVertex;r++)
adjMat[r][c]=adjMat[r][c+1];
nVertex--;
}
void deleteEdge(int vStart,int vEnd)
{	if(vStart>nVertex || vStart<1 || vEnd>nVertex|| vEnd<1)
return;
```

```
if(!checkDir()) adjMat [vStart-1] [vEnd-1] =0;
}
int checkDir()
{         int r,c;
for(r=0;r<nVertex;r++)
for(c=0;c<nVertex;c++)
if(adjMat[r][c]!=adjMat[c][r])
return 1;
return 0;
}
int checkWt()
{         int r,c;
for(r=0;r<nVertex;r++)
for(c=0;c<nVertex;c++)
if(adjMat[r][c]>1)
return 1;
return 0;
}
void createGraph()
{         int r,c;
printf("\n enter the no. of vertices : ");
scanf("%d",&nVertex);
for(r=0;r<nVertex;r++)
for(c=0;c<nVertex;c++)
{       adjMat[r][c]=0;
if(r!=c)
insertEdge(r+1,c+1);
}
}
void viewGraph()
{
int v,r,c,edge,inDeg[VSIZE],outDeg[VSIZE];
for(v=0;v<nVertex;v++)
printf(" v%d",v+1);
for(r=0;r<nVertex;r++)
{       printf("\nv%-2d         ",r+1);
for(c=0;c<nVertex;c++)
printf("%-2d    ",adjMat[r][c]);
}
for(v=0;v<nVertex;v++)
inDeg[v]=outDeg[v]=0;
edge=0;
for(r=0;r<nVertex;r++)
for(c=0;c<nVertex;c++)
if(adjMat[r][c]!=0)
{       edge++;
outDeg[r]++;
inDeg[c]++;
}
if(!checkDir())
edge=edge/2;
printf("n %s Graph",(checkDir())?"DIRECTED" : "UNDIRECTED");
printf("n %s Graph",(checkDir())?"Weighted" : "UNWeighted");
printf("total no. of vertices :%d",nVertex);
printf("\ntotal no.of Edges : %d",edge);
printf("Vertex Indegree Outdegree");
for(r=0;r<nVertex;r++)
{         printf("\n v%-2d          %-9d %-9d",r+1,inDeg[r],outDeg[r]);
if(inDeg[r]==0 && outDeg[r]!=0)
printf(":          %s","SOURE");
if(inDeg[r]!=0 && outDeg[r]==0)
printf(":          %s","SKIN");
if(inDeg[r]==1 && outDeg[r]==0)
printf(":          %s","PENDANT");
if(inDeg[r]==0 && outDeg[r]==0)
printf(":          %s","ISOLATED");
}
}
void display_menu()
```

```
{
printf("\n\n basic operation in an adjacency matrix..........");
printf("\n\t 0. Display Menu");
printf("\n\t 1.Creation of Graph");
printf("\n\t 2. Insert a Vertex");
printf("\n\t 3. Insert an Edge");
printf("\n\t 4. Delete aVertex");
printf("\n\t 5. Delete an Edge ");
printf("\n\t 6. Veiw the Graph");
printf("\n\t 7. Exit");
}
```

OUTPUT:

```
Basic operation in an adjacency matrix..........
0. Display Menu
1.Creation of Graph
2. Insert a Vertex
3. Insert an Edge
4. Delete aVertex
5. Delete an Edge
6. Veiw the Graph
7. Exit
?1

enter the no. of vertices : 5
enter weight of the Edge from v1 to v2 :1
enter weight of the Edge from v1 to v3 :0
enter weight of the Edge from v1 to v4 :1
enter weight of the Edge from v1 to v5 :1
enter weight of the Edge from v2 to v1 :1
enter weight of the Edge from v2 to v3 :1
enter weight of the Edge from v2 to v4 :1
enter weight of the Edge from v2 to v5 :0
enter weight of the Edge from v3 to v1 :0
enter weight of the Edge from v3 to v2 :1
enter weight of the Edge from v3 to v4 :1
enter weight of the Edge from v3 to v5 :1
enter weight of the Edge from v4 to v1 :1
enter weight of the Edge from v4 to v2 :1
enter weight of the Edge from v4 to v3 :1
enter weight of the Edge from v4 to v5 :1
enter weight of the Edge from v5 to v1 :1
enter weight of the Edge from v5 to v2 :0
enter weight of the Edge from v5 to v3 :1
enter weight of the Edge from v5 to v4 :1

?6
v1 v2 v3 v4 v5
v1    0    1    0    1    1
v2    1    0    1    1    0
v3    0    1    0    1    1
v4    1    1    1    0    1
v5    1    0    1    1    0



n UNDIRECTED Graphn UNWeighted G
raphtotal no. of vertices :5
total no.of Edges : 8Vertex Indegree Outdegree
v1   3      3
v2   3      3
v3   3      3
v4   4      4
v5   3      3
? 7
```

Program 6

/**

To implement Depth First Search for a graph nonrecursively

**/

AIM: To implement Depth First Search for a graph non recursively

Source Code:

```
#include<stdio.h>
#include<conio.h>
int bfs[20],rear=-1,front=-1,vt[20];
void store(int);
int remove();
void store(int n)
{
bfs[++rear]=n;
}
int removeele()
{
int n;
n=bfs[++front];
return n;
}
void bfsearch(int a[][10],int n,int id)
{
int i,j;
for(i=0;i<n;i++)
{
vt[i]=0;
}
id=id-1;
store(id);
vt[id]=1;
printf("Visited nodes are:");
while(front!=rear)
{
id=removeele();
printf("%5d",id+1);
for(i=0;i<n;i++)
{
if(a[id][i]==1)
{
if(vt[i]==0)
{
store(i);
vt[i]=1;
}
}
}
}
}

int main()
{
int a[10][10],n,i,j,id=1;
clrscr();
printf("Enter no.of nodes:");
scanf("%d",&n);
printf("Enter adjacency matrix:");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("Enter starting node:");
scanf("%d",&id);
bfsearch(a,n,id);
getch();
}
```

OUTPUT:

```
Enter no.of nodes:4

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

Enter starting node:1

Visited nodes are:    1    2    3    4
```

Program 7

```
/***********************************************************

To implement Breadth First Search for a graph nonrecursively

***********************************************************/
```

AIM: To implement Breadth First Search for a graph nonrecursively

Source Code:

```
#include<stdio.h>
#include<conio.h>
int vt[10],dfs[10],top=-1;
void push(int);
int pop();
void push(int n)
{
dfs[++top]=n;
}
int pop()
{
int n;
n=dfs[top--];
return n;
}
void dfsearch(int a[][10],int n,int id)
{
int i,j;
for(i=0;i<n;i++)
{
vt[i]=0;
}
id=id-1;
push(id);
vt[id]=-1;
printf("Visited nodes are:");
while(top!=-1)
{
id=pop();
printf("%5d",id+1);
for(i=0;i<n;i++)
{
if(a[id][i]==1)
{
if(vt[i]==0)
{
push(i);
vt[i]=1;
}
}
}
}

}
Int  main()
{

int a[10][10],m,i,j,id=0,n;
clrscr();
printf("Enter no.of elements:");
scanf("%d",&n);
printf("Enter the adjacency matrix:");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("Enter searching node:");
scanf("%d",&id);
dfsearch(a,n,id);
getch();
}
```

OUTPUT:

```
Enter no.of elements:4

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

Enter searching node:1


Visited nodes are:    1    3    4    2
```

Program 8

```
/***********************************************************

To implement Prim’s algorithm to generate a min-cost spanning tree

***********************************************************/
```

AIM: To implement Prim’s algorithm to generate a min-cost spanning tree.

Source Code:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 10000000;
int main()
{
   int g[20][20],vi[20][20],te[20][20],n,i,j,t,s,min_key,output[10],min,k,b;
   printf("enter the number of vertices u wanna insert");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
               for(j=1;j<=n;j++)
               {
                           printf("\nif the edge present between %d and %d enter weight else enter zero",i,j);
                           scanf("%d",&b);
                           g[i][j]=b;
                           g[j][i]=b;
               }//for
   }//for
```

```
for(i=1;i<=n;i++)
{
            for(j=1;j<=n;j++)
            {
                        if(i==j)
                        {
                             te[i][j]=0;
                        }//if
                        else if(g[i][j]==0)
                        {
                           te[i][j]=INFINITY;
                        }
                        else if(g[i][j]!=0)
                        {
                          te[i][j]=g[i][j];
                        }//else
           }//for
}//for

 for(i=1;i<=n;i++)
{
            for(j=1;j<=n;j++)
            {
                        vi[i][j]=0;
            }//for
```

```
}//for
 //printf("where from the visit shud start%d",INFINITY);
//scanf("%d",&i);
i=1;
for(j=1;j<=n;j++)
{
        vi[j][i]=1;
}
min=INFINITY;

t=1;
s=1;
output[t]=1;

while(t<n)
{
    for(i=1;i<=t;i++)
    {
            for(j=1;j<=n;j++)
            {
                    if(vi[i][j]!=1&&te[i][j]!=0)
                    {
                                if(te[i][j]<min)
                                {
```

```
                                                        min=te[i][j];
                                                        min_key=j;
                                                }
                                }//if
                        }//for
                }//for
                 // printf("%5d",min);
                  //printf("\n");
                  //printf("%5d",min_key);
                t++;
                output[t]= min_key;
               for(k=1;k<=n;k++)
               {
                        vi[k][min_key]=1;
               }
               min=INFINITY;


}//while


printf("\nthe required min spanning tree is \n");
for(i=1;i<=n;i++)
{
         printf("%5d",output[i]);
}
```

```
getch();
return 0;
}//main
```

OUTPUT:

```
Enter the number of vertices you wanna insert3

If the edge  present between 1 and 1enter the weight else enter zero 1

If the edge  present between 1 and 2enter the weight else enter zero 0

If the edge  present between 1 and 3 enter the weight else enter zero 5

If the edge  present between 2 and 1 enter the weight else enter zero 9

If the edge  present between 2 and 2 enter the weight else enter zero 6

If the edge  present between 2 and 3 enter the weight else enter zero 3

If the edge  present between 3 and 1 enter the weight else enter zero 7

If the edge  present between 3 and 2 enter the weight else enter zero 4

If the edge  present between 3 and 3 enter the weight else enter zero 5

The required min spanning tree is

1       3       2
```

Program 9

```
/************************************************************

To implement Krushkal's algorithm to generate a min-cost spanning tree.

************************************************************/
```

AIM: To implement Krushkal's algorithm to generate a min-cost spanning tree.

Source Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[20][20],b[20][20],c[20][20],d[20][20],nod=0,n,val1=0,i,j,k,t,m=0,posx,posy,val;
   clrscr();
   printf("\nEnter the value of n ");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix ");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&a[i][j]);
         b[i][j]=(i==j?0:a[i][j]);
         m=m+(b[i][j]?1:0);
         c[i][j]=0;
         d[i][j]=0;
      }
   }
   for(m=m/2;m!=0&&(nod!=(n-1));m--)
   {
      val=32767;
      for(i=0;i<n;i++)
      {
         for(j=0;j<n;j++)
         {
            if(b[i][j]!=0&&b[i][j]<val)
            {
               posx=i;
               posy=j;
               val=b[i][j];
            }
         }
      }
      b[posx][posy]=0;
      b[posy][posx]=0;
      if(c[posx][posy]==0)
      {
         c[posx][posy]=1;
         c[posy][posx]=1;
         for(k=0;k<n;k++)
         {
            for(i=0;i<n;i++)
            {
               for(j=0;j<n;j++)
               {
                  c[i][j]=c[i][j]|(c[i][k]&c[k][j]);
               }
            }
         }
         val1=val1+a[posx][posy];
         nod=nod+1;
         d[posx][posy]=a[posx][posy];
         d[posy][posx]=a[posy][posx];
      }
   }
   if(nod==n-1)
   {
      for(i=0;i<n;i++)
      {
         printf("\n");
         for(j=0;j<n;j++)
         {
               printf("%d ",d[i][j]);
         }
      }
      printf("\nSpanning tree has a cost of %d",val1);
   }
   else
   {
      printf("\nSpanning tree does not exist!!");
   }
   getch();
}
```

OUTPUT:

```
Enter the value of n 3

Enter the adjacency matrix 4

5

6

1

2

3

7

8

9


0       5       0

1       0       3

0       8       0

Spanning tree has a cost of 4
```

Program 10

/***

To implement Dijkstra's algorithm to find shortest path in the graph
***/

AIM: To implement Dijkstra's algorithm to find shortest path in the graph.

Source Code:

Method -1

```
#include<stdio.h>
#include<conio.h>
struct prioq
{
   int dist,pr,s;
};
void display(struct prioq p[20],int source,int dest,int d)
{
   if(dest==source)
   {
      printf("%d",source);
      return;
   }
   display(p,source,p[dest].s,d);
   printf("->%d",dest);
}
void main()
{
   struct prioq p[20];
   int i,j,n,k,min,pos,source,a[20][20];
   clrscr();
   printf("\nEnter the value of n ");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix ");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&a[i][j]);
      }
      p[i].pr=0;
```

```
    }
    printf("\nEnter the source node ");
    scanf("%d",&source);
    for(i=0;i<n;i++)
    {
        p[i].s=source;
        p[i].dist=a[source][i];
    }
    pos=source;
    for(j=0;j<n;j++)
    {
        i=pos;
        p[i].pr=-1;
        min=9999;
        for(k=0;k<n;k++)
        {
            if(p[i].dist+a[i][k]<p[k].dist)
            {
                p[k].dist=p[i].dist+a[i][k];
                p[k].s=i;
            }
            if(p[k].pr!=-1&&p[k].dist<min)
            {
                pos=k;
                min=p[k].dist;
            }
        }
    }
    for(i=0;i<n;i++)
    {
        printf("\n");
        if(i==source)
        {
            printf("%d->%d",source,source);
        }
        else
        {
        display(p,source,i,i);
        }
        printf(" %d",p[i].dist);
    }
    getch();
}
```

OUTPUT:

```
Enter the value of n 3

Enter the adjacency matrix 2

5

8

4

6

9

1

2

4

Enter the source node 2

2 → 0  1

2→ 1   2

2→ 2   4
```

Method -2

```
#include<stdio.h>
#include<conio.h>
void display(int par[],int source,int dest1,int dest)
{
   if(source==dest1&&dest1==dest)
   {
      printf("%d->%d",source,source);
      return;
   }
   if(source==dest1)
   {
      printf("%d",source);
      return;
   }
   display(par,source,par[dest1],dest);
   printf("->%d",dest1);
}
int main()
{
   int par[20],dist[20],vist[20],a[20][20],n,i,j,k,min,pos,source;
  // clrscr();
   printf("\nEnter the value of n ");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix ");
```

```
for(i=0;i<n;i++)
{
   for(j=0;j<n;j++)
   {
      scanf("%d",&a[i][j]);
   }
}
printf("\nEnert the source node ");
scanf("%d",&source);
for(i=0;i<n;i++)
{
   vist[i]=0;
   dist[i]=a[source][i];
   par[i]=source;
}
i=source;
for(j=0;j<n;j++)
{
   vist[i]=1;
   min=9999;
   for(k=0;k<n;k++)
   {
   if(dist[i]+a[i][k]<dist[k])
      {
         dist[k]=dist[i]+a[i][k];
```

```
                par[k]=i;
            }
            if(vist[k]==0&&dist[k]<min)
            {
                min=dist[k];
                pos=k;
            }
        }
        i=pos;
    }
    for(i=0;i<n;i++)
    {
        printf("\n");
        display(par,source,i,i);
        printf(" %d",dist[i]);
    }
    getch();
}
```

OUTPUT:

```
Enter the value of n 3

Enter the adjacency matrix7

8

9

4

5

6

1

2

3

Enter the source node 6

6 → 2→0       1

6→2→1         2

6→ 2   0
```

Program 11

```
/**************************************************************

To implement pattern matching using Boyer-Moore algorithm.

**************************************************************/

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

int *build_table(char *p)

{

int last[128];

int i;

for(i=0;i<128;i++)

last[i]=-1;

for(i=0;i<strlen(p);i++)

{

last[p[i]]=i;
}
return last;
}
int min(int a,int b)
{
if(a>b)
return b;
else
return a;
}
int search(char *t,char *p)
{
int *last=build_table(p);
int i,j,m,n;
 n=strlen(t);
 m=strlen(p);
 i=m-1;
 j=m-1;
if(i>n-1)
return -1;
do
{
if(p[j]==t[i])
if(j==0)
return i;
else
{
i--;
j--;
else
{
i=i+m-min(j,1+last[t[i]]);
j=m-1;

}

}

while(i<=n-1);

return -1;

}

void main()

{

char t[50];

char p[50];

clrscr();

int index;

printf("enter T string : ");

gets(t);

printf("enter P string : ");

gets(p);
```

```
index=search(t,p);

if(index!=-1)

{

printf("match found at:index");

}

else

{

printf("match not found");

}

}
```

Program 12

```
/************************************************************
To implement Knuth-Morris-Pratt algorithm for pattern matching.
************************************************************/

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

int *create_prefix_table(char p[10])
```

```
{
int *prefix_table;
int m=strlen(p);
int k=0;
int q;
prefix_table=(int *)malloc(m*sizeof(int));
prefix_table[0]=0;
for(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[50])

{

int *prefix_table;

int i;

int j=0;

int n=strlen(t);

int m=strlen(p);

prefix_table=create_prefix_table(p);

printf("----------");

printf("\n prefix array is:\n");

for(i=0;i<m;i++)

printf("%d",prefix_table[i]);

printf("\n----------");

printf("\n");

for(i=0;i<n;i++)
```

```
{
printf("\n i=%d ,j=%d",i,j);
printf("\n comparing %c and %c",t[i],p[j]);
while(j>0 && p[j]!=t[i])
j=prefix_table[j-1];
if(p[j]==t[i])
{
printf("\n match found at:i=%d j=%d",i,j);
j++;
}
if(j==m)
{
printf("\n\t pattern is present in the text at: ");
printf(" position = %d",i-m+1);
printf("\n");
j=prefix_table[j-1];
}

}

}

int main()

{

char t[50];

char p[50];

clrscr();

printf("enter the T string : ");

gets(t);

printf("enter the P string : ");

gets(p);

kmp_match(t,p);

return 0;

}
```