**PROGRAM CODE:**

#include<stdio.h>

#include<stdlib.h>

#include "heapadt.h"

void main()

{

priorityqueue q=NULL;

position pos;

int choice,element,n;

printf("Enter the maximum no of elements: ");

scanf("%d",&n);

q=create(n);

do{

printf("\n\*\*\*PRIORITY QUEUE MENU\*\*\*\n\n1.Insert element\n2.Delete element\n3.Display the elements\n4.Exit\n");

printf("\nEnter choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

if(!isfull(q))

{ printf("Enter element to be inserted: ");

scanf("%d",&element);

}

insert(element,q);

break;

case 2:

if(isempty(q))

printf("\nPriority Queue is empty\n");

else

{ element=deletemin(q);

printf("\n%d deleted from priority queue\n",element); }

break;

case 3:

display(q);

printf("\n");

break;

case 4:

break;

default:

printf("Invalid choice\n");

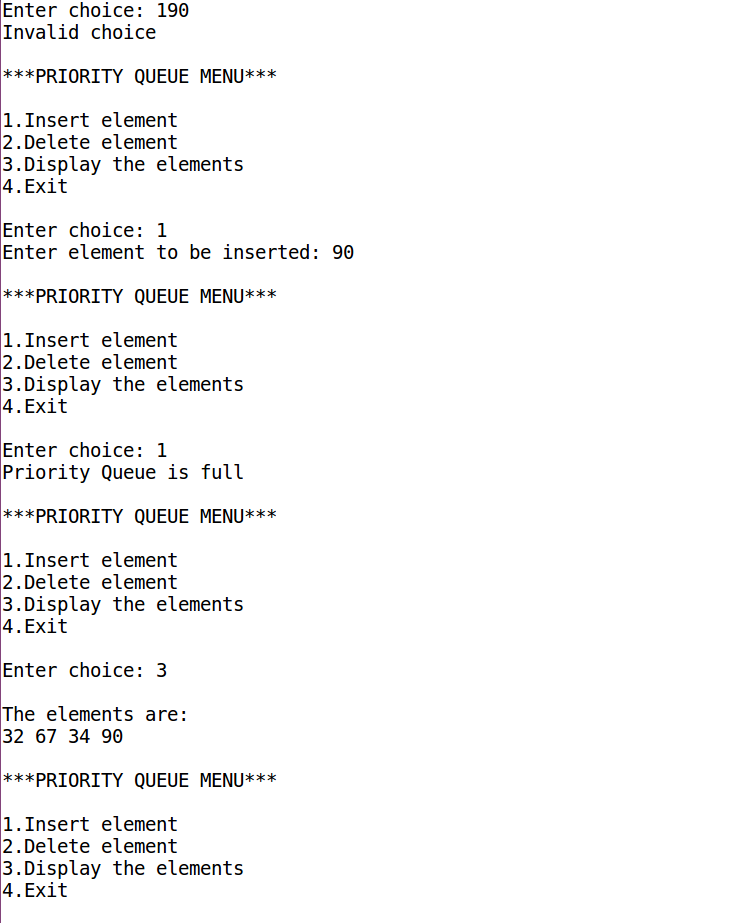
}

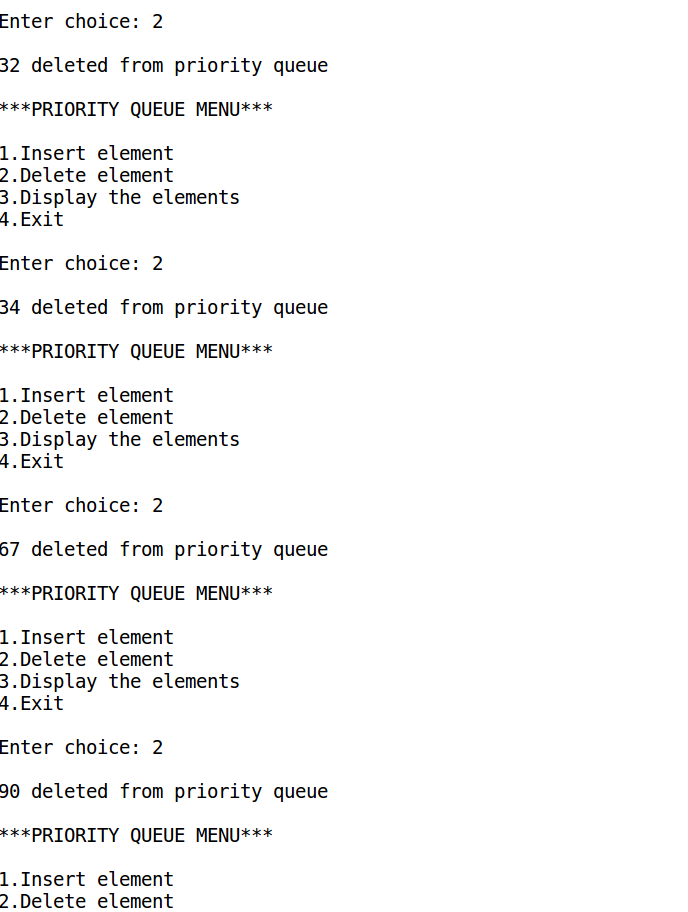
}while(choice!=4);

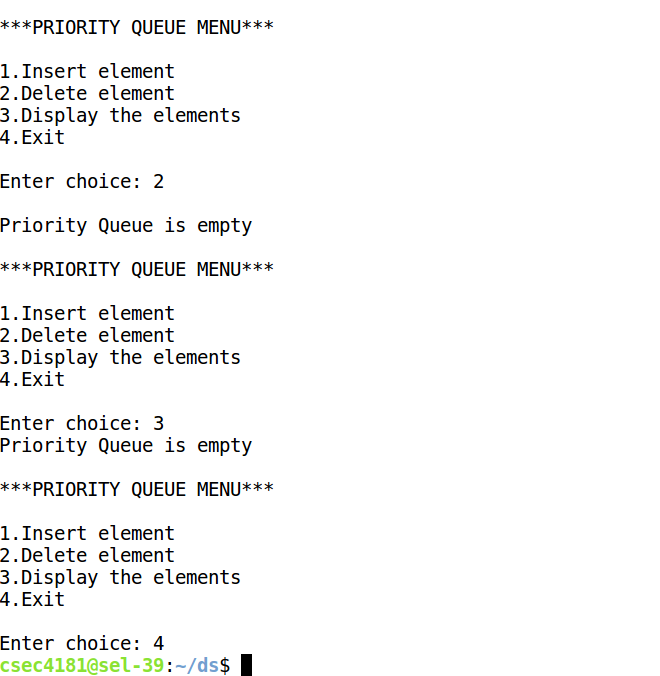
}

**OUTPUT:**









**PROGRAM CODE:**

#include<stdio.h>

#include<stdlib.h>

#include "intque.h"

#include "intstack.h"

#include "graphADT.h"

void visit(elementtype x)

{

printf("%d ",x);

}

int unvisitedNeigh(int n,int \*\* adj\_mat,int z,int visited[])

{

int i,j,y;

for(y=1;y<n+1;y++)

if(adj\_mat[z][y]==1 && visited[y]==0)

return y;

return -1;

}

void dfs(graph g)

{

int i,x=1,y,z,p=0; //p is the no of visited notes

int n=g->vertices;

int visited[n+1]; //array to check if a node has been visited

for(i=0;i<n+1;i++)

visited[i]=0;

int\_stack s=create\_int\_stack(n);

while(p!=n)

{

visit(x);

p++;

visited[x]=1;

intpush(x,s);

x++;

while(!int\_isempty(s))

{ z=int\_return\_top(s);

y=unvisitedNeigh(n,g->adj\_mat,z,visited);

if(y!=-1)

{

visit(y);

p++;

visited[y]=1;

intpush(y,s);

}

else

intpop(s);

}

}

}

void bfs(graph g)

{

int i,x=1,y,z,p=0; //p is the no of visited notes

int n=g->vertices;

int visited[n+1]; //array to check if a node has been visited

for(i=0;i<n+1;i++)

visited[i]=0;

queue q =createqueue(n);

while(p!=n)

{

visit(x);

p++;

visited[x]=1;

enqueue(q,x);

x++;

while(!isempty(q))

{ z=dequeue(q);

for(y=1;y<n+1;y++)

{ if(g->adj\_mat[z][y]==1 && visited[y]==0)

{

visit(y);

p++;

visited[y]=1;

enqueue(q,y);

}

}

}

}

}

void main()

{

int n;

printf("Enter number of vertices: ");

scanf("%d",&n);

graph g=create\_graph(n);

printf("\nThe adjacency matrix is\n");

for(int i=1;i<n+1;i++)

{ for(int j=1;j<n+1;j++)

printf("%d ",g->adj\_mat[i][j]);

printf("\n");

}

printf("\nThe BFS is: ");

bfs(g);

printf("\nThe DFS is: ");

dfs(g);

printf("\n");

}

**graphADT header file**

typedef struct graphnode\* graph;

struct graphnode

{

int vertices;

int \*\*adj\_mat;

};

graph create\_graph(int v)

{

int i,n,j,k;

graph g=(graph)malloc(sizeof(struct graphnode));

if(g==NULL)

{ printf("Out of space\n");

exit(1);

}

g->vertices=v;

int\*\*a=(int\*\*)malloc(sizeof(int\*)\*v+1);

for(int i=1;i<v+1;i++)

a[i]=(int\*)malloc(sizeof(int)\*v+1);

for(int i=1;i<v+1;i++)

for(int j=1;j<v+1;j++)

a[i][j]=0;

printf("Enter the egdes as ViVj (Press -1 to stop):\n ");

scanf("%d",&n);

do

{

k=n%10;

j=n/10;

a[j][k]=1;

scanf("%d",&n);

}while(n!=-1);

g->adj\_mat=a;

return g;

}

**intque header file**

typedef struct queuerecord\* queue;

typedef int elementtype;

struct queuerecord

{

int capacity;

int front;

int rear;

int size;

elementtype \*array;

};

void makeempty(queue q)

{

q->size = 0;

q->front = -1;

q->rear = -1;

}

queue createqueue(int maxelements)

{

int i;

queue q=(queue)malloc(sizeof(struct queuerecord));

if(q==NULL)

printf("Out of space!");

q->array=(elementtype\*)malloc(sizeof(elementtype) \* maxelements);

if(q->array == NULL)

printf("Out of space!");

q->capacity = maxelements;

makeempty(q);

return q;

}

int isempty (queue q)

{

return q->size == 0;

}

int isfull (queue q)

{

return q->size == q->capacity;

}

void enqueue(queue q, elementtype item)

{

if(isfull(q))

printf("Queue is full");

else

{

q->rear++;

q->array[q->rear]=item;

if(q->front==-1)

q->front=0;

++q->size;

}

}

int dequeue(queue q)

{

int data;

if(isempty(q))

{

printf("Queue is empty");

return;

}

else

{

data = q->array[q->front];

if(q->front==q->rear)

{ makeempty(q);

return data; }

else if(q->front+1 == q->capacity)

q->front=0;

else

q->front++;

q->size--;

return data;

}

}

void display(queue q)

{

int i;

if(isempty(q))

printf("\nContainer is empty");

else

{ printf("\nThe products in the container are: ");

i=q->front;

while(i!=q->rear)

{ printf("\n%d",q->array[i]); i=(i+1)%q->capacity; } //end while

printf("\n%d\n",q->array[q->rear]);

} //end else

}

**intstack header file**

typedef struct int\_stackrecord\* int\_stack;

struct int\_stackrecord

{

int capacity;

int topofstack;

int \*array;

};

int\_stack create\_int\_stack(int maxelements)

{

int\_stack s=malloc(sizeof(struct int\_stackrecord));

if(s==NULL)

printf("Out of space!");

s->array=malloc(sizeof(int) \* maxelements);

if(s->array == NULL)

printf("Out of space!");

s->capacity = maxelements;

s->topofstack = -1;

return s;

}

int int\_isempty(int\_stack s)

{

return s->topofstack == -1;

}

int int\_isfull(int\_stack s)

{

return s->topofstack==s->capacity-1;

}

void intpush(int x,int\_stack s)

{

if(int\_isfull(s))

printf("Stack is full");

else

s->array[++s->topofstack] = x;

}

void intpop(int\_stack s)

{

if(int\_isempty(s))

printf("Stack is empty");

else

s->topofstack--;

}

void int\_display(int\_stack s)

{

int i;

for(i=0;i<s->topofstack;i++)

printf("%c",s->array[i]);

}

int int\_return\_top(int\_stack s)

{

if(!int\_isempty(s))

return s->array[s->topofstack];

else

return s->topofstack;

}

int int\_topandpop(int\_stack s)

{

if(!int\_isempty(s))

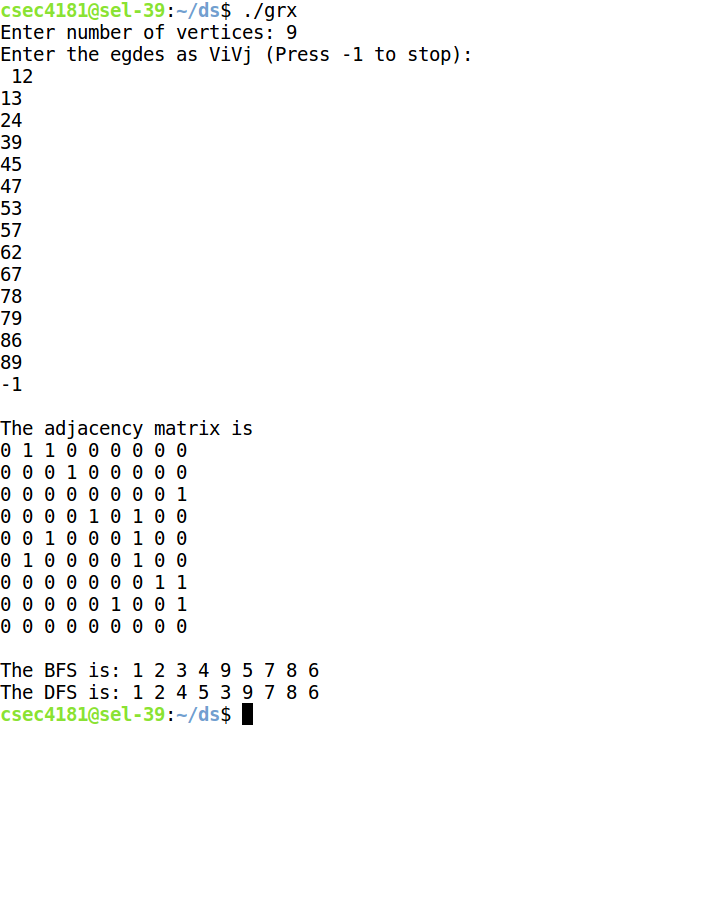
return s->array[s->topofstack--];

else

return s->topofstack;

}

**OUTPUT:**

****