8. Linked List

#include<iostream.h>

#include<conio.h>

#include<alloc.h>

#include<process.h>

struct node

{

int data;

struct node \*link;

};

struct node \*head=NULL;

struct node \*temp=NULL;

void insertion();

void deletion();

void infirst();

void inlast();

void inpos();

void delfirst();

void dellast();

void delpos();

void display();

void display1();

void search();

void editposition();

void sorting();

void assort();

void dessort();

void creation();

int value();

int count=0;

inline void insertsuc(){

cout<<"\nInsertion Successful.";

count++;

getch();

}

inline void deletesuc(){

cout<<"\nDeletion Successful.";

count--;

getch();

}

void main()

{

int ch=0;

clrscr();

do

{

clrscr();

if(count==0)

{

cout<<"\nList is Empty";

cout<<"\n\t\tSingly Linked List Operations";

cout<<"\n1.Create a node";

cout<<"\n2.Exit";

cout<<"\nEnter your Choice:";

cin>>ch;

if(ch==1)

creation();

else if(ch==2)

exit(0);

else

{

cout<<"\nInvalid Input";

getch();

}

}

else{

cout<<"\n\tSingly Linked List Operations\n";

cout<<"\n1.Insertion";

cout<<"\n2.Deletion";

cout<<"\n3.Display";

cout<<"\n4.Search";

cout<<"\n5.Position Editing";

cout<<"\n6.Sorting";

cout<<"\n7.Exit";

cout<<"\nEnter your Choice : ";

cin>>ch;

switch(ch)

{

case 1:

insertion();

break;

case 2:

deletion();

break;

case 3:

display();

break;

case 4:

search();

break;

case 5:

editposition();

break;

case 6:

sorting();

break;

case 7:

break;

default:

cout<<"\nInvalid choice";

}

}

}while(ch!=7);

}

void creation()

{

head=(struct node\*)malloc(sizeof(struct node));

head->data=value();

head->link=NULL;

count=1;

}

void insertion()

{

int ch;

do

{

clrscr();

cout<<"Insert Operations";

cout<<"\n1.Insert at first";

cout<<"\n2.Insert at last";

cout<<"\n3.Insert at specified position";

cout<<"\n4.Exit";

cout<<"\n\nEnter your Choice : ";

cin>>ch;

switch(ch)

{

case 1:

infirst();

break;

case 2:

inlast();

break;

case 3:

inpos();

break;

case 4:

break;

default:

cout<<"\nInvalid choice";

}

}while(ch!=4);

}

int value()

{

int y;

cout<<"\nEnter the value : ";

cin>>y;

return(y);

}

void infirst()

{

temp=(struct node\*)malloc(sizeof(struct node));

temp->data=value();

temp->link=head;

head=temp;

insertsuc();

}

void inlast()

{

struct node \*p=NULL;

temp=(struct node\*)malloc(sizeof(struct node));

temp->data=value();

p=head;

while(p->link!=NULL)

p=p->link;

p->link=temp;

temp->link=NULL;

insertsuc();

}

void inpos()

{

struct node \*p=NULL;

temp=(struct node\*)malloc(sizeof(struct node));

int pos=0; //Position

cout<<"\nEnter the position : ";

cin>>pos;

if(pos<=count&&pos>0)

{

if(pos==1)

infirst();

else

{

temp->data=value();

p=head;

for(int i=1;i<pos-1;i++)

p=p->link;

temp->link=p->link;

p->link=temp;

insertsuc();

}

}

else if(pos==count+1)

inlast();

else{

cout<<"\nPosition you entered is wrong";

getch();

}

}

void deletion()

{

int ch;

do

{

clrscr();

cout<<"Delete Operations";

cout<<"\n1.Delete at first";

cout<<"\n2.Delete at Last ";

cout<<"\n3.Delete at Specify Position";

cout<<"\n4.Exit";

cout<<"\n\nEnter your Choice : ";

cin>>ch;

switch(ch)

{

case 1:

delfirst();

break;

case 2:

dellast();

break;

case 3:

delpos();

break;

case 4:

break;

default:cout<<"\nInvalid choice";

}

if(count==0)

ch=4;

}while(ch!=4);

}

void delfirst()

{

struct node \*p=NULL;

p=head;

head=head->link;

cout<<"\n"<<p->data<<" Is Deleted";

free(p);

deletesuc();

}

void dellast()

{

struct node \*p=NULL;

temp=NULL;

if(count==1)

delfirst();

else

{

p=head;

while((p->link)->link!=NULL)

p=p->link;

temp=p->link;

p->link=NULL;

cout<<"\n"<<temp->data<<" Is Deleted";

free(temp);

deletesuc();

}

}

void delpos()

{

int pos;

struct node \*p=NULL;

cout<<"\nEnter the position : ";

cin>>pos;

if(pos==1)

delfirst();

else if(pos==count)

dellast();

else if(pos<count&&pos>1){

p=head;

for(int i=1;i<pos-1;i++)

p=p->link;

temp=p->link;

p->link=temp->link;

cout<<temp->data<<" Is Deleted";

free(temp);

deletesuc();

}

else{

cout<<"\nPosition you entered wrong";

getch();

}

}

void display()

{

struct node \*p=NULL;

p=head;

cout<<"\nList: ";

while(p->link!=NULL)

{

cout<<"\t"<<p->data;

p=p->link;

}

cout<<"\t"<<p->data;

cout<<"\nNo of nodes: "<<count;

getch();

}

void display1()

{

struct node \*p=NULL;

p=head;

cout<<"\nList:";

while(p->link!=NULL)

{

cout<<"\t"<<p->data;

p=p->link;

}

cout<<"\t"<<p->data;

getch();

}

void search()

{

int x,no=0,pos=0;

temp=head;

cout<<"\nEnter searching element: ";

cin>>x;

while(temp->link!=NULL||(pos+1)==count){

if(x==temp->data)

{

cout<<"\nSearching element position: "<<pos+1;

no++;

}

temp=temp->link;

pos++;

}

if(no!=0)

cout<<"\nNumber of time searching element occured : "<<no;

else

cout<<"\nIt not occured anywhere in list";

getch();

}

void editposition()

{

int pos=0,x=0;

temp=NULL;

cout<<"\nEnter the position : ";

cin>>pos;

if(pos>0&&pos<=count){

cout<<"\nEnter the new value: ";

cin>>x;

temp=head;

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

temp=temp->link;

temp->data=x;

cout<<"\nSuccesfully edited";

}

else

cout<<"\nEntered position does not exit";

getch();

}

void sorting()

{

int ch;

do

{

clrscr();

cout<<"\n1.Ascending sorting";

cout<<"\n2.Descending sorting";

cout<<"\n3.Exit";

cout<<"\nEnter your choice:";

cin>>ch;

switch(ch)

{

case 1:

assort();

break;

case 2:

dessort();

break;

case 3:

break;

default:

cout<<"\nInvalid choice";

}

}

while(ch!=3);

}

void assort()

{

temp=NULL;

struct node \*p=NULL,\*q=NULL;

int min=0,pos=0;

temp=head;

q=head;

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

{

min=temp->data;

pos=i;

p=temp->link;

for(int j=i+1;j<=count;j++)

{

if(min>(p->data))

{

min=p->data;

pos=j;

}

p=p->link;

}

q=head;

for(int k=1;k<pos;k++)

q=q->link;

q->data=temp->data;

temp->data=min;

temp=temp->link;

}

cout<<"\nAscending sorted list";

display1();

}

void dessort()

{

temp=NULL;

struct node \*p=NULL,\*q=NULL;

int min=0,pos=0;

temp=head;

q=head;

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

{

min=temp->data;

pos=i;

p=temp->link;

for(int j=i+1;j<=count;j++)

{

if(min<(p->data))

{

min=p->data;

pos=j;

}

p=p->link;

}

q=head;

for(int k=1;k<pos;k++)

q=q->link;

q->data=temp->data;

temp->data=min;

temp=temp->link;

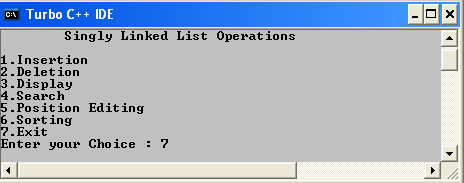
}

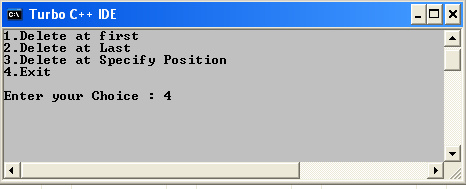
cout<<"\n Descending order sorted list:";

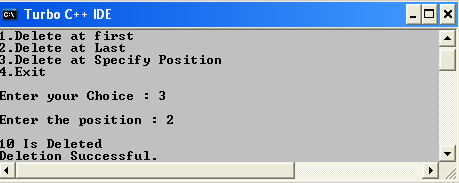
display1();

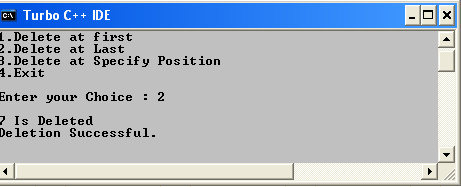
}

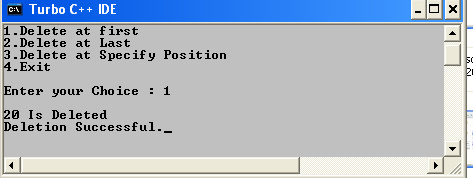
**Output:**

****

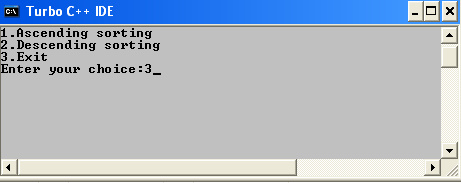
****

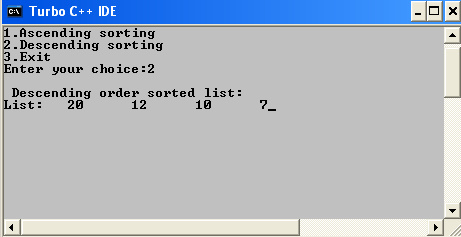
****

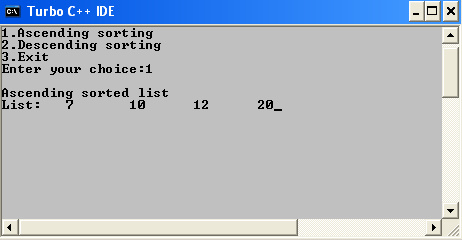
****

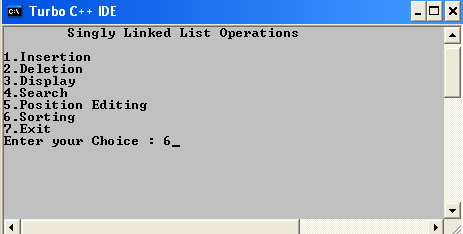
****

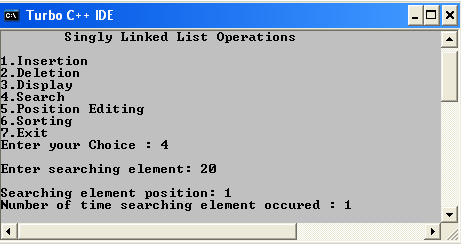
****

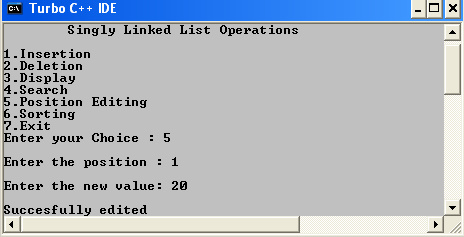
****

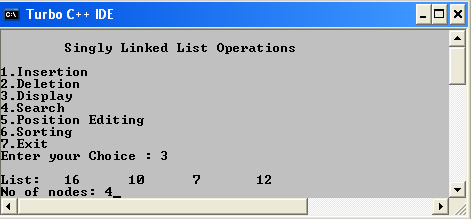
****

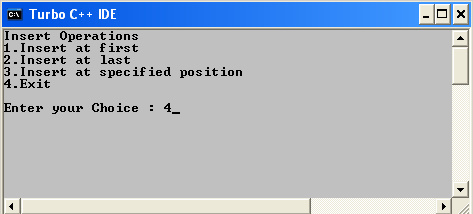
****

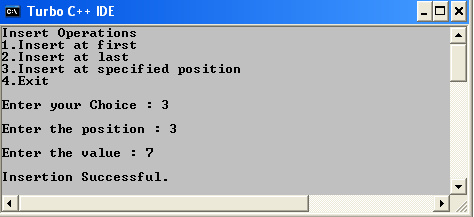
****

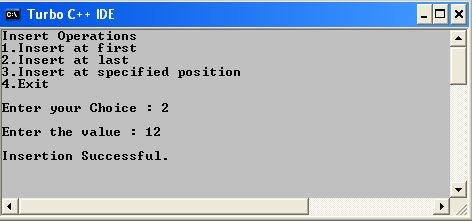
****

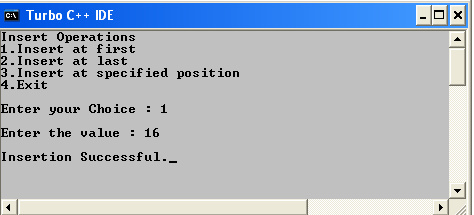
****

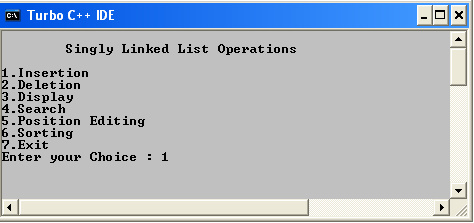
****

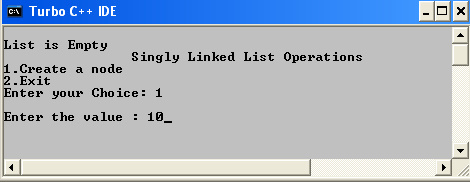
****

****

****

****

****

****

**9. KNAPSACK PROBLEM USING DYNAMIC PROGRAMMING**

#include <stdio.h>

#include<conio.h>

#define max(a,b) (a > b ? a : b)

int matrix[100][100] = {0};

int picks[100][100] = {0};

int knapsack(int nItems, int size, int weights[], int values[]){

int i,j;

for (i=1;i<=nItems;i++){

for (j=0;j<=size;j++){

if (weights[i-1]<=j){

matrix[i][j] = max(matrix[i-1][j],values[i-1]+matrix[i-1][j-weights[i-1]]);

if (values[i-1]+matrix[i-1][j-weights[i-1]]>matrix[i-1][j])

picks[i][j]=1;

else

picks[i][j]=-1;

}

else{

picks[i][j] = -1;

matrix[i][j] = matrix[i-1][j];

}

}

}

return matrix[nItems][size];

}

void printPicks(int item, int size, int weights[]){

while (item>0){

if (picks[item][size]==1){

printf("%d ",item-1);

item--;

size -= weights[item];

}

else{

item--;

}

}

printf("\n");

return;

}

int main(){

int i;

int nItems;

int knapsackSize;

int weights[10];

int values[10];

clrscr();

printf("\n KNAPSACK PROBLEM USING DYNAMIC PROGRAMMING ");

printf("\nEnter No of Items: ");

scanf("%d",&nItems);

printf("\nEnter Knapsacksize: ");

scanf("%d",&knapsackSize);

for(i=1;i<=nItems;i++)

{

printf("\nEnter Weight of Item %d: ",i);

scanf("%d",&weights[i]);

}

for(i=1;i<=nItems;i++)

{

printf("\nEnter value of Item %d: ",i);

scanf("%d",&values[i]);

}

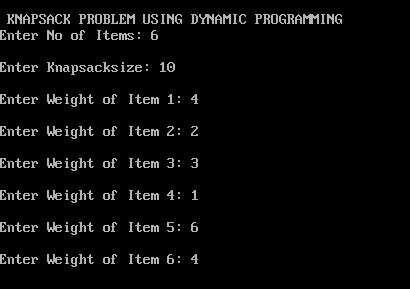
printf("\nMax value = %d",knapsack(nItems,knapsackSize,weights,values));

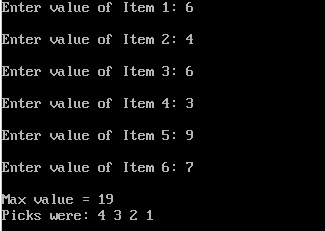
printf("\nPicks were: ");

printPicks(nItems,knapsackSize, weights);

getch();}

**OUTPUT:**

****



**10. TRAVELING SALESMAN PROBLEM (DYNAMMIC PROGRMMING)**

#include <iostream.h>

#include <conio.h>

class dynamic

{

private:

int c[5][5],n,d[24],p[24][6],list[5],r;

public:

dynamic();

void getdata();

void display();

int fact(int num);

int min(int list[]);

void perm(int list[], int k, int m);

void sol();

};

dynamic::dynamic()

{

r=0;

}

void dynamic::getdata()

{

cout<<"Enter no. of cities:";

cin>>n;

cout<<endl;

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

for (int j=0;j<n;j++)

c[i][j]=0;

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

{

for (j=0;j<n;j++)

{

if (i!=j)

{

if (c[i][j]==0)

{

cout<<"Enter cost from "<<i<<" to "<<j<<" :";

cin>>c[i][j];

c[j][i]=c[i][j];

}

}

}

}

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

list[i]=i+1;

}

int dynamic::fact(int num)

{

int f=1;

if (num!=0)

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

f=f\*i;

return f;

}

void dynamic::perm(int list[], int k, int m)

{

int i,temp;

if (k==m)

{

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

{

p[r][i+1]=list[i];

}

r++;

}

else

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

{

temp=list[k]; list[k]=list[i]; list[i]=temp;

perm(list,k+1,m);

temp=list[k]; list[k]=list[i]; list[i]=temp;

}

}

void dynamic::sol()

{

perm(list,0,n-2);

for (int i=0;i<fact(n-1);i++)

{

p[i][0]=0;

p[i][n]=0;

}

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

{

d[i]=0;

for (int j=0;j<n;j++)

{

d[i]=d[i]+c[p[i][j]][p[i][j+1]];

}

}

}

int dynamic::min(int list[])

{

int minimum=list[0];

for (int i=0;i<fact(n-1);i++)

{

if (list[i]<minimum)

minimum=list[i];

}

return minimum;

}

void dynamic::display()

{

int i,j;

cout<<endl<<"The cost Matrix:"<<endl;

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

{

for (j=0;j<n;j++)

cout<<c[i][j]<<"\t";

cout<<endl;

}

cout<<endl<<"The Possible paths and their corresponding cost:"<<endl;

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

{

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

cout<<p[i][j]<<"\t";

cout<<"--> "<<d[i]<<endl;

}

cout<<endl<<"The shortest path :"<<endl;

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

{

if (d[i]==min(d))

break;

}

for (j=0;j<=n;j++)

{

cout<<p[i][j]<<" ";

}

cout<<endl<<"\nThe cost of this path is "<<d[i]<<endl;

}

void main()

{

clrscr();

dynamic ts;

ts.getdata();

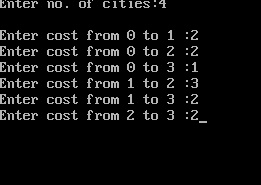
ts.sol();

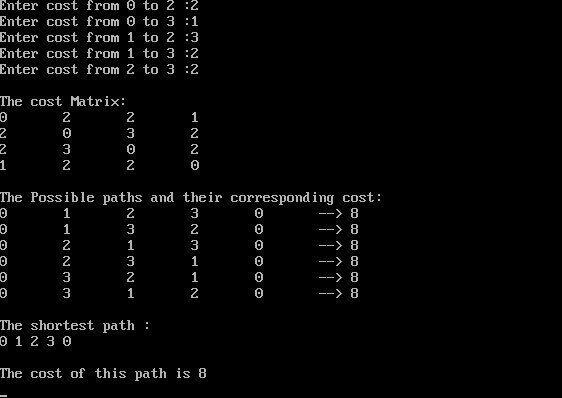
ts.display();

getch();

}

OUTPUT :





**11. PRIM’S ALGORITHM**

#include<stdio.h>

#define INF 1000

int vertex[10];

int wght[10][10];

int new\_wght[10][10];

int closed[10];

int n;

int inclose(int i,int n1)

{

int j;

for(j=0;j<=n1;j++)

if(closed[j]==i)

return 1;

return 0;

}

void buildtree()

{

int i=0,j,count=0;

int min,k,v1=0,v2=0;

closed[0]=0;

while(count<n-1)

{

min=INF;

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

for(j=0;j<n;j++)

if(wght[closed[i]][j]<min && !inclose(j,count))

{

min=wght[closed[i]][j];

v1=closed[i];

v2=j;

}

new\_wght[v1][v2]=new\_wght[v2][v1]=min;

count++;

closed[count]=v2;

getch();

}

}

void main()

{

int i,j,ed,sum=0;

clrscr();

printf("PRIM'S ALGORITHM\n");

printf("Enter the No of Nodes : ");

scanf("%d",&n);

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

{

vertex[i]=i+1;

for(j=0;j<n;j++)

{

wght[i][j]=INF;

new\_wght[i][j]=INF;

}

}

printf("\nEnter Weight\n");

printf("Enter 0 if path doesn't exist b/w two vertices(v1,v2) Otherwise enter Weight\n");

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

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

{

printf("%d --- %d : ",vertex[i],vertex[j]);

scanf("%d",&ed);

if(ed>=1)

wght[i][j]=wght[j][i]=ed;

}

getch();

clrscr();

buildtree();

printf("\nMINIMUM SPANNING TREE\n");

printf("LIST OF EDGES\n");

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

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

if(new\_wght[i][j]!=INF)

{

printf("%d --- %d = %d ",vertex[i],vertex[j],new\_wght[i][j]);

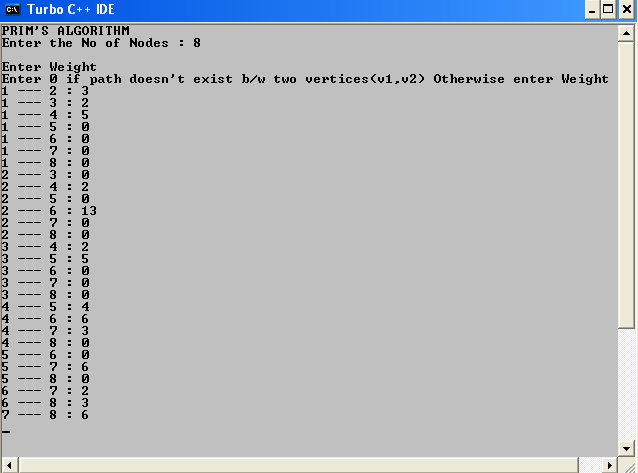
sum+=new\_wght[i][j];

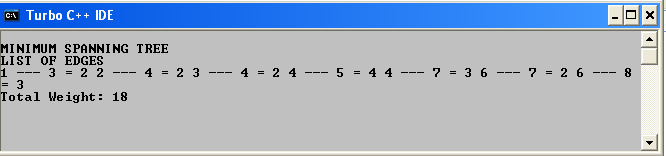
}

printf("\nTotal Weight: %d ",sum);

getch();

}OUTPUT :



****

**12. KRUSKAL ALGORITHM**

#include<stdio.h>

#define INF 1000

char vertex[10];

int wght[10][10];

int span\_wght[10][10];

int source;

struct Sort

{

int v1,v2;

int weight;

}que[20];

int n,ed,f,r;

int cycle(int s,int d)

{

int j,k;

if(source==d)

return 1;

for(j=0;j<n;j++)

if(span\_wght[d][j]!=INF && s!=j)

{

if(cycle(d,j))

return 1;

}

return 0;

}

void build\_tree()

{

int i,j,w,k,count=0;

for(count=0;count<n;f++)

{

i=que[f].v1;

j=que[f].v2;

w=que[f].weight;

span\_wght[i][j]=span\_wght[j][i]=w;

source=i;

k=cycle(i,j);

if(k)

span\_wght[i][j]=span\_wght[j][i]=INF;

else

count++;

}

}

void swap(int \*i,int \*j)

{

int t;

t=\*i;

\*i=\*j;

\*j=t;

}

void main()

{

int i,j,k=0,temp;

int sum=0;

clrscr();

printf("\n\n\tKRUSKAL'S ALGORITHM\n\n");

printf("\n\tEnter the No. of Nodes : ");

scanf("%d",&n);

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

{

printf("\n\tEnter %d value : ",i+1);

fflush(stdin);

scanf("%c",&vertex[i]);

for(j=0;j<n;j++)

{

wght[i][j]=INF;

span\_wght[i][j]=INF;

}

}

printf("\n\nEnter Weight\n");

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

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

{

printf("\nEnter 0 if path Doesn't exist b\w %c to %c : ",vertex[i],vertex[j]);

scanf("%d",&ed);

if(ed>=1)

{

wght[i][j]=wght[j][i]=ed;

que[r].v1=i;

que[r].v2=j;

que[r].weight=wght[i][j];

if(r)

{

for(k=0;k<r;k++)

if(que[k].weight>que[r].weight)

{

swap(&que[k].weight,&que[r].weight);

swap(&que[k].v1,&que[r].v1);

swap(&que[k].v2,&que[r].v2);

}

}

r++;

}

}

clrscr();

build\_tree();

printf("\n\n\t\tMINIMUM SPANNING TREE\n\n");

printf("\n\t\tLIST OF EDGES\n\n");

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

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

if(span\_wght[i][j]!=INF)

{

printf("\n\t\t%c ------ %c = %d ",vertex[i],vertex[j],span\_wght[i][j]);

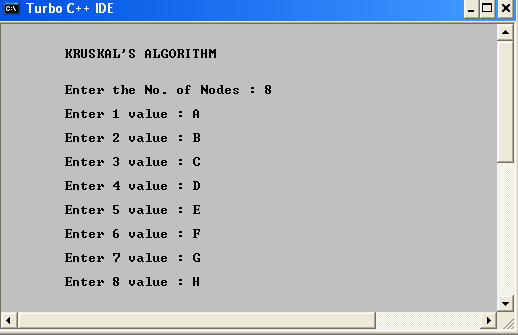
sum+=span\_wght[i][j];

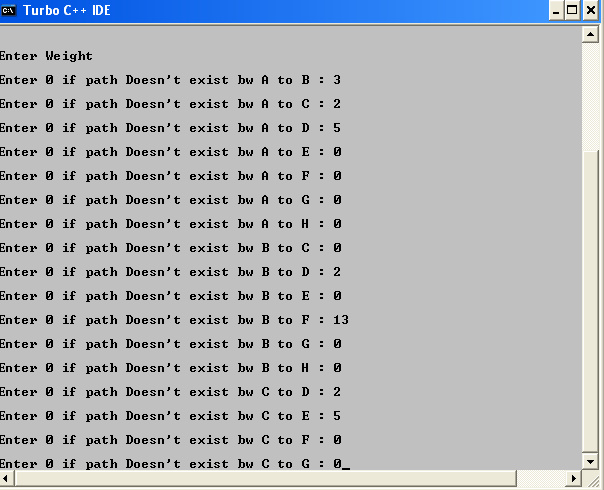
}

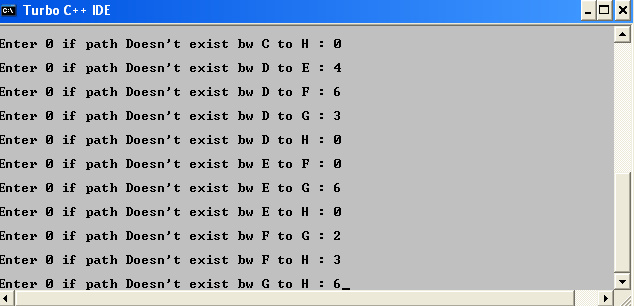
printf("\n\n\t\tTotal Weight: %d ",sum);

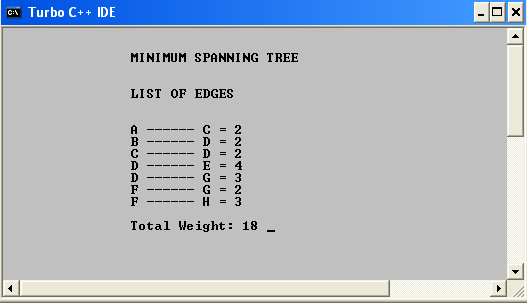
getch();

}

**OUTPUT :**

****

****

****