**Practical no :- 2**

**Program :-**

|  |
| --- |
|  |
|  | #include<stdio.h> |
|  | 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. Not all nodes are reachable"); |
|  | break; |
|  | } |
|  | } |
|  | } |

**Output:-**

1. Enter the number of vertices:4

Enter graph data in matrix form:

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

Enter the starting vertex:1

The node which are reachable are:

1

Bfs is not possible. Not all nodes are reachable

1. Enter the number of vertices:4

Enter graph data in matrix form:

1 0 1 0

0 0 1 1

1 1 1 1

1 1 1 1

Enter the starting vertex:1

The node which are reachable are:

1 2 3 4

**Practical no :- 3**

**Program :-**

#include<stdio.h>

int ary[10][10],completed[10],n,cost=0;

void takeInput()

{

int i,j;

printf("Enter the number of villages: ");

scanf("%d",&n);

printf("\nEnter the Cost Matrix\n");

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

{

printf("\nEnter Elements of Row: %d\n",i+1);

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

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

completed[i]=0;

}

printf("\n\nThe cost list is:");

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

{

printf("\n");

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

printf("\t%d",ary[i][j]);

}

}

void mincost(int city)

{

int i,ncity;

completed[city]=1;

printf("%d--->",city+1);

ncity=least(city);

if(ncity==999)

{

ncity=0;

printf("%d",ncity+1);

cost+=ary[city][ncity];

return;

}

mincost(ncity);

}

int least(int c)

{

int i,nc=999;

int min=999,kmin;

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

{

if((ary[c][i]!=0)&&(completed[i]==0))

if(ary[c][i]+ary[i][c] < min)

{

min=ary[i][0]+ary[c][i];

kmin=ary[c][i];

nc=i;

}

}

if(min!=999)

cost+=kmin;

return nc;

}

int main()

{

takeInput();

printf("\n\nThe Path is:\n");

mincost(0); //passing 0 because starting vertex

printf("\n\nMinimum cost is %d\n ",cost);

return 0;

}

**Output:-**

Enter the number of villages: 4

Enter the Cost Matrix

Enter Elements of Row: 1

2 4 5 6

Enter Elements of Row: 2

3 5 6 1

Enter Elements of Row: 3

2 4 5 6

Enter Elements of Row: 4

5 6 2 4

The cost list is:

2 4 5 6

3 5 6 1

2 4 5 6

5 6 2 4

The Path is:

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

**Practical no :- 4**

Program :- #include<stdio.h>

#include<string.h>

#include<unistd.h>

#include<sys/types.h>

#include <sys/stat.h>

#include <stdlib.h>

#include<time.h>

#define r 3

#define c 3

char matrix[r][c];

char new[r][c];

int count;

char final[r][c] = {{'1','2','3'},{'4','5','6'},{'7','8',' '}};

int i,j;

char z ;

int p,q,x,y;

int t =0;

int result = 0;

void load()

{for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{if(new[i][j] == '0')

{

matrix[i][j]= ' ';

continue;

}

matrix[i][j]= new[i][j];

}}}

void blank()

{for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{new[i][j]= ' ';}}}

int main()

{time\_t T= time(NULL);

struct tm tm = \*localtime(&T);

char f[4];

int rsl ;

int random,t;

int randvalues[9];

main:

count = 0;

blank();

T= time(NULL);

tm = \*localtime(&T);

srand(tm.tm\_sec);

while(count!=9)

{

rsl=rand()%9;

sprintf(f,"%d",rsl);

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

{

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

{

if((new[i][j]) == f[0])

{

i = 4; j = 4;

continue;

}else if((new[i][j]) == ' ')

{

new[i][j] = f[0];

i = 4; j = 4;

count++;

}

}

}

}

load();

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

{

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

{

printf("|%c|",matrix[i][j]);

}

printf("\n");

}

while(1)

{

printf("enter value to change its position to blank space\n");

scanf(" %c",&z);

if(z=='q')

{

printf("\n\*\*\*\*\*You Quit\*\*\*\*\*\n");

goto main;

// break;

}

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

{

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

{

if((matrix[i][j])== z)

{

p = i;

q = j;

}else if((matrix[i][j])== ' ')

{

x = i;

y = j;

}

}

}

t =0;

int m , n ;

m = p - 1;

n = q ;

if(m>=0)

{

if((matrix[m][n])== ' ')t=1;

}

m = p + 1;

if(m<=2)

{

if((matrix[m][n])== ' ')t=1;

}

m = p;

n = q - 1 ;

if(n>=0)

{

if((matrix[m][n])== ' ')t=1;

}

n = q + 1 ;

if(n<=2)

{

if((matrix[m][n])== ' ')t=1;

}

if(t==1)

{

matrix[x][y] = z;

matrix[p][q] = ' ';

}

t = 0;

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

{

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

{if((matrix[i][j])== final[i][j])

{

t++;

}}}

system("clear");

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

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

{

printf("|%c|",matrix[i][j]);}

printf("\n");}

if(t==9)

{printf("\n\*\*\*\*you Win\*\*\*\*\n");

break;

}

}return 1;

}

**Output:-**

|2|| ||8|

|5||4||3|

|6||1||7|

enter value to change its position to blank space

4

[H[2J[3J|2||4||8|

|5|| ||3|

|6||1||7|

enter value to change its position to blank space

3

[H[2J[3J|2||4||8|

|5||3|| |

|6||1||7|

enter value to change its position to blank space

7

[H[2J[3J|2||4||8|

|5||3||7|

|6||1|| |

enter value to change its position to blank space

1

[H[2J[3J|2||4||8|

|5||3||7|

|6|| ||1|

**Practical no :- 5**

**Program :-**

**#include <stdio.h>**

**#include<iostream>**

**#include<fstream>**

**#include<string.h>**

**using namespace std;**

**bool check(char \*\*board,int i,int j,int st,int N){**

**if(st == 1){**

**if(board[i][j] == 'K'){**

**return false;**

**}**

**for(int k = 0;k<N;k++){**

**if(board[i][k] == 'Q'){**

**return false;**

**}**

**}**

**for(int k = 0;k<N;k++){**

**if(board[k][j] == 'Q'){**

**return false;**

**}**

**}**

**for(int k = i-1,s = j-1;k >= 0 && s >= 0;k--,s--){**

**if(board[k][s] == 'Q'){**

**return false;**

**}**

**}**

**for(int k=i+1,s=j+1;k<N && s<N;k++,s++){**

**if(board[k][s] == 'Q'){**

**return false;**

**}**

**}**

**for(int k=i-1,s=j+1;k>=0 && s<N;k--,s++){**

**if(board[k][s] == 'Q'){**

**return false;**

**}**

**}**

**for(int k = i+1,s = j-1;k<N && s >= 0;k++,s--){**

**if(board[k][s] == 'Q'){**

**return false;**

**}**

**}**

**int k = i;**

**int s = j;**

**if( k-2 >= 0){**

**if(s-1 >= 0){**

**if(board[k-2][s-1] == 'K'){**

**return false;**

**}**

**}**

**if(s+1 < N){**

**if(board[k-2][s+1] == 'K'){**

**return false;**

**}**

**}**

**}**

**if( k+2 < N){**

**if(s-1 >= 0){**

**if(board[k+2][s-1] == 'K'){**

**return false;**

**}**

**}**

**if(s+1 < N){**

**if(board[k+2][s+1] == 'K'){**

**return false;**

**}**

**}**

**}**

**if( s-2 >= 0){**

**if(k-1 >= 0){**

**if(board[k-1][s-2] == 'K'){**

**return false;**

**}**

**}**

**if(k+1 < N){**

**if(board[k+1][s-2] == 'K'){**

**return false;**

**}**

**}**

**}**

**if( s+2 < N){**

**if(k-1 >= 0){**

**if(board[k-1][s+2] == 'K'){**

**return false;**

**}**

**}**

**if(k+1 < N){**

**if(board[k+1][s+2] == 'K'){**

**return false;**

**}**

**}**

**}**

**}**

**if(st == 2){**

**if(board[i][j] == 'Q'){**

**return false;**

**}**

**int k = i;**

**int s = j;**

**if( k-2 >= 0){**

**if(s-1 >= 0){**

**if(board[k-2][s-1] == 'K'){**

**return false;**

**}**

**}**

**if(s+1 < N){**

**if(board[k-2][s+1] == 'K'){**

**return false;**

**}**

**}**

**}**

**if( k+2 < N){**

**if(s-1 >= 0){**

**if(board[k+2][s-1] == 'K'){**

**return false;**

**}**

**}**

**if(s+1 < N){**

**if(board[k+2][s+1] == 'K'){**

**return false;**

**}**

**}**

**}**

**if( s-2 >= 0){**

**if(k-1 >= 0){**

**if(board[k-1][s-2] == 'K'){**

**return false;**

**}**

**}**

**if(k+1 < N){**

**if(board[k+1][s-2] == 'K'){**

**return false;**

**}**

**}**

**}**

**if( s+2 < N){**

**if(k-1 >= 0){**

**if(board[k-1][s+2] == 'K'){**

**return false;**

**}**

**}**

**if(k+1 < N){**

**if(board[k+1][s+2] == 'K'){**

**return false;**

**}**

**}**

**}**

**}**

**return true;**

**}**

**void initializeBoard(char \*\*board,int Q,int K,int N){**

**for(int i = 0;i<N;i++){**

**for(int j = 0;j<N;j++){**

**board[i][j] = ' ';**

**}**

**}**

**for(int i = 0;i<N;i++){**

**for(int j = 0;j<N;j++){**

**if(Q > 0){**

**if(check(board,i,j,1,N)){**

**board[i][j] = 'Q';**

**Q--;**

**}**

**}**

**if(K > 0){**

**if(check(board,i,j,2,N)){**

**board[i][j] = 'K';**

**K--;**

**}**

**}**

**if(Q == 0 && K == 0){**

**break;**

**}**

**}**

**}**

**}**

**void printBoard(char \*\*Board,int N){**

**cout<<endl;**

**for(int i =0;i<N;i++){**

**for(int j = 0;j<N;j++){**

**cout<<"|"<<Board[i][j];**

**}**

**cout<<"|"<<endl;**

**}**

**}**

**int main(){**

**int N;**

**int Q;**

**int K;**

**int tmax;**

**std::cout << "Value of N: "; cin >> N; cout << endl;**

**if (cin.fail())**

**{**

**std::cout << "Please enter integer only & try again. Thanks!" << endl;**

**exit(0);**

**}**

**else if (N < 4)**

**{**

**std::cout << " No possible solution. Please try again for n > 4.";**

**exit(0);**

**}**

**std::cout << "Value of Q: "; cin >> Q; cout << endl;**

**if (cin.fail())**

**if (cin.fail())**

**{**

**std::cout << "Please enter integer only & try again. Thanks!" << endl;**

**exit(0);**

**}**

**else if (Q < 1)**

**{**

**std::cout << " No possible solution. Please try again for Q > 1.";**

**exit(0);**

**}**

**std::cout << "Value of K: "; cin >> K; cout << endl;**

**if (cin.fail())**

**if (cin.fail())**

**{**

**std::cout << "Please enter integer only & try again. Thanks!" << endl;**

**exit(0);**

**}**

**else if (K < 1)**

**{**

**std::cout << " No possible solution. Please try again for K > 1.";**

**exit(0);**

**}**

**char \*\*board = new char\*[N];**

**for(int i = 0;i<N;i++){**

**board[i] = new char[N];**

**}**

**initializeBoard(board,Q,K,N);**

**printBoard(board,N);**

**return 0;**

**}**

**Output:-**

Value of N: 4

Value of Q: 4

Value of K: 2

|Q|K|K| |

| | |Q| |

| | | | |

| |Q| | |

**Practical no :- 6**

**Program :-**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int x, y;

struct node \*next;

}\*root, \*left, \*right;

struct node\* genNewState(struct node\*, int, int, int, int);

int isNodePresent(struct node \*next, int jug1, int jug2, int f1, int f2)

{

struct node \*temp;

if((next->x == f1) && (next->y == f2)){

return(0);

}

if((next->x == jug1) && (next->y == jug2)){

return(1);

}

if((next->x == 0) && (next->y == 0)){

return(1);

}

temp = left;

while(1)

{

if((temp->x == next->x) && (temp->y == next->y)){

return(1);

}

else if(temp->next == NULL){

break;

}

else{

temp = temp->next;

}

}

temp = right;

while(1)

{

if((temp->x == next->x) && (temp->y == next->y)){

return(1);

}

else if(temp->next == NULL){

break;

}

temp = temp->next;

}

return(0);

}

void bfstree(int jug1, int jug2, int f1, int f2)

{

int flag1, flag2;

struct node \*tempLeft, \*tempRight;

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

root->x = 0; root->y = 0; root->next = NULL;

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

left->x = 0; left->y = jug2; left->next = NULL;

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

right->x = jug1; right->y = 0; right->next = NULL;

tempLeft = left;

tempRight = right;

while(1)

{

flag1 = 0; flag2 = 0;

if((tempLeft->x != f1) || (tempLeft->y != f2)) {

tempLeft->next = genNewState(tempLeft, jug1, jug2, f1, f2);

tempLeft = tempLeft->next;

tempLeft->next = NULL;

flag1 = 1;

}

if((tempRight->x != f1) || (tempRight->y != f2)) {

tempRight->next = genNewState(tempRight, jug1, jug2, f1, f2);

tempRight = tempRight->next;

tempRight->next = NULL;

flag2 = 1;

}

if((flag1 == 0) && (flag2 == 0)){

break;

}

}

}

struct node\* genNewState(struct node \*current, int jug1, int jug2, int f1, int f2)

{

int d;

struct node \*next;

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

next->x = jug1;

next->y = current->y;

if(isNodePresent(next, jug1, jug2, f1, f2) != 1){

return(next);

}

next->x = current->x;

next->y = jug2;

if(isNodePresent(next, jug1, jug2, f1, f2) != 1){

return(next);

}

next->x = 0;

next->y = current->y;

if(isNodePresent(next, jug1, jug2, f1, f2) != 1){

return(next);

}

next->y = 0;

next->x = current->x;

if(isNodePresent(next, jug1, jug2, f1, f2) != 1) {

return(next);

}

if((current->y < jug2) && (current->x != 0)) {

d = jug2 - current->y;

if(d >= current->x) {

next->x = 0;

next->y = current->y + current->x;

} else {

next->x = current->x - d;

next->y = current->y + d;

}

if(isNodePresent(next, jug1, jug2, f1, f2) != 1) {

return(next);

}

}

if((current->x < jug1) && (current->y != 0)) {

d = jug1 - current->x;

if(d >= current->y) {

next->y = 0;

next->x = current->x + current->y;

}

else

{

next->y = current->y - d;

next->x = current->x + d;

}

if(isNodePresent(next, jug1, jug2, f1, f2) != 1) {

return(next);

}

}

return(NULL);

}

void BFS(int f1, int f2)

{

struct node \*temp1 = left, \*temp2 = right;

printf("\nSoultion : \n");

printf("(%d , %d)\n", root->x, root->y);

while(1)

{

printf("(%d , %d)\n", temp1->x, temp1->y);

if((temp1->x == f1)&&(temp1->y == f2)){

break;

}

temp1 = temp1->next;

printf("(%d , %d)\n", temp2->x, temp2->y);

if((temp2->x == f1)&&(temp2->y == f2)){

break;

}

temp2 = temp2->next;

}

}

int main()

{

int jug1, jug2, f1, f2;

printf("Enter the Capacity of jug1 : ");

scanf("%d", &jug1);

printf("Enter the Capacity of jug2 : ");

scanf("%d", &jug2);

printf("\nRequired Water in jug1 : ");

scanf("%d", &f1);

printf("Required Water in jug2 : ");

scanf("%d", &f2);

bfstree(jug1, jug2, f1, f2);

BFS(f1, f2);

return 0;

}

**Output:-**

Enter the Capacity of jug1 : 4

Enter the Capacity of jug2 : 3

Required Water in jug1 : 2

Required Water in jug2 : 0

Soultion :

(0 , 0)

(0 , 3)

(4 , 0)

(3 , 0)

(1 , 3)

(3 , 3)

(1 , 0)

(4 , 2)

(0 , 1)

(0 , 2)

(4 , 1)

(2 , 0)