```cpp
#include<bits/stdc++.h>
using namespace std;

int g = 0;

void Print(int puzzle[]){
    for(int i= 0; i < 9; i++){
        if((i % 3) == 0) cout<<"\n";

        if(puzzle[i] == -1){
            cout<<"_ ";
        }
        else{
            cout<<puzzle[i]<<" ";
        }
    }
    cout<<"\n\n";
}

void moveLeft(int start[], int position){
    swap(start[position], start[position - 1]);
}

void moveRight(int start[], int position){
    swap(start[position], start[position + 1]);
}

void moveUp(int start[], int position){
    swap(start[position], start[position - 3]);
}

void moveDown(int start[], int position){
    swap(start[position], start[position + 3]);
}

void copy(int temp[], int real[]){
    for(int i = 0; i < 9; i++){
        temp[i] = real[i];
    }
}

int heuristic(int start[], int goal[]){
    int h = 0;

    for(int i = 0; i < 9; i++){
        for(int j = 0; j < 9; j++){
            if(start[i] == goal[j] && start[i] != -1){
                h += abs((j-i)/3) + abs((j-i)%3);
            }
        }
    }
    return h + g;
}

void moveTile(int start[], int goal[]){
    int emptyAt = 0;
    for(int i =0; i<9; i++){
```

```
        if(start[i] == -1){
            emptyAt = i;
        }
    }

    int t1[9], t2[9], t3[9], t4[9], f1 = INT_MAX, f2 = INT_MAX, f3 = INT_MAX, f4 = INT_MAX;
    copy(t1, start);
    copy(t2, start);
    copy(t3, start);
    copy(t4, start);

    int row = emptyAt / 3;
    int col = emptyAt % 3;

    if(col - 1>=0){
        moveLeft(t1, emptyAt);
        f1 = heuristic(t1, goal);
    }

    if(col + 1 < 3){
        moveRight(t2, emptyAt);
        f2 = heuristic(t2, goal);
    }

    if(row + 1 < 3){
        moveDown(t3, emptyAt);
        f3 = heuristic(t3, goal);
    }

    if(row - 1 >= 0){
        moveUp(t4, emptyAt);
        f4 = heuristic(t4, goal);
    }

    if(f1 <= f2 && f1 <= f3 && f1 <= f4){
        moveLeft(start, emptyAt);
    }
    else if(f2 <= f1 && f2 <= f3 && f2 <= f4){
        moveRight(start, emptyAt);
    }
    else if(f3 <= f1 && f3 <= f2 && f3 <= f4){
        moveDown(start, emptyAt);
    }
    else if(f4 <= f1 && f4 <= f2 && f4 <= f3){
        moveUp(start, emptyAt);
    }

}

void solveEight(int start[], int goal[]){
    g++;

    moveTile(start, goal);
    Print(start);

    int f = heuristic(start, goal);
```

```
    if(f == g){
        cout<<"Solved in "<<f<<" moves!!"<<endl;
        return;
    }


    solveEight(start, goal);

}




int main(){

    int start[9], goal[9];
    cout<<"Input: ";
    for(int i = 0; i < 9; i++){
        cin>>start[i];
    }
    cout<<"Goal: ";
    for(int i = 0; i < 9; i++){
        cin>>goal[i];
    }

    solveEight(start, goal);

    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;

class Graph {
    public:

    map<int, list<int>> adjlist;
    map<int, bool> visited, visited1;
    queue<int> q;

    void addEdge(int s, int d){
        adjlist[s].push_back(d);
        adjlist[d].push_back(s);
    }

    void dfs(int node){

        visited1[node] = true;

        cout<<node<<" ";

        for(int i : adjlist[node]){
            if(!visited1[i]){
                dfs(i);
            }
```

```
        }
    }

    void bfs(){
        if(q.empty()){
            return ;
        }

        int node = q.front();
        q.pop();

        cout<<node<<" ";

        for(int i : adjlist[node]){
            if(!visited[i]){
                visited[i] = true;
                q.push(i);
            }
        }
        bfs();
    }
};


int main(){
    Graph g;

    int n;
    cout<<"Enter total number of edges: ";
    cin>>n;

    for(int i = 0; i < n; i++){
        int a, b;
        cout<<"\nEnter nodes joining Edge: ";
        cin>>a>>b;
        g.addEdge(a, b);

    }


    int ch = 0;


    while(ch != 3){

    cout << "\nEnter \n 1 to perform DFS \n 2 to perform BFS \n 3 to exit";
    cin >> ch;

    switch(ch){
        case 1:
            cout << "\nDFS on the given graph is :";
            g.dfs(1);
            break;

        case 2:
            cout << "\nBFS on the given graph is: ";
            g.q.push(1);
```

```
            g.visited[1] = true;
            g.bfs();
            break;

        case 3:
            ch = 3;
    }

    }
    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;

struct node{
    int parent;
    int rank;
};

struct edge{
    int src;
    int dest;
    int wt;
};

vector<node> dsuf;
vector<edge> mst;

bool cmp(edge a, edge b){
    return a.wt < b.wt;
}


int find(int v){
    if(dsuf[v].parent == -1){
        return v;
    }

    return dsuf[v].parent = find(dsuf[v].parent);

}

void union_op(int fromP, int toP){
    if(dsuf[fromP].rank < dsuf[toP].rank){
        dsuf[fromP].parent = toP;
        dsuf[toP].rank++;
    }
    else if(dsuf[fromP].rank > dsuf[toP].rank){
        dsuf[toP].parent = fromP;
        dsuf[fromP].rank++;
    }
    else{
```

```
        dsuf[fromP].parent = toP;
        dsuf[toP].rank++;
    }
}

void krushkal(vector<edge>& edgeList, int V, int E){
    sort(edgeList.begin(), edgeList.end(), cmp);

    int i = 0, j = 0;
    cout<<V<<" "<<E<<endl;
    while(i < (V-1) && (j < E)){
        int fromP = find(edgeList[j].src);
        int toP = find(edgeList[j].dest);

        if(fromP == toP){
            j++;
            continue;
        }

        union_op(fromP, toP);
        mst.push_back(edgeList[j]);

        i++;
    }
}

void printMst(vector<edge> mst){
    int min_cost = 0;
    for(auto edge: mst){
        cout<<"\n"
            <<"src "<<edge.src<<" "
            <<"dest "<<edge.dest<<" "
            <<"wt "<<edge.wt<<"\n";

        min_cost += edge.wt;

    }
     cout <<"Minimum cost is "<< min_cost << endl;



}

int main(){

    int v, e;
    cout << "Enter the number of edges and vertices: ";
    cin >> e >> v;
    dsuf.resize(v);
    for(int i = 0; i < v; i++){
        dsuf[i].parent = -1;
        dsuf[i].rank = 0;
    }

    vector<edge> edgeList;
    edge temp;
```

```
    for(int i = 0; i < e; i++){
        int s, d, w;
        cin>>s>>d>>w;
        temp.src = s;
        temp.dest = d;
        temp.wt = w;

        edgeList.push_back(temp);
    }

    krushkal(edgeList, v, e);
    printMst(mst);

}
```

```
#include<bits/stdc++.h>
using namespace std;

#define V 5

int minkey(int key[], bool mstSet[]){
    int min = INT_MAX, min_index = -1;

    for(int i = 0; i < V; i++){
        if(key[i] < min && mstSet[i] == false){
            min = key[i];
            min_index = i;
        }
    }
    return min_index;

}

void printMst(int graph[V][V], int parent[]){
    for(int i = 1; i < V; i++){
        cout<<parent[i]<<" - "<<i<<" "<<graph[i][parent[i]]<<"\n";
    }
}

void primMst(int graph[V][V]){
    int parent[V];

    int Keys[V];

    bool mstSet[V];

    for(int i = 0; i < V; i++){
        Keys[i] = INT_MAX;
        mstSet[i] = false;
    }

    Keys[0] = 0;
    parent[0] = -1;
```

```
    for(int count = 0; count < V-1; count++){
        int u = minkey(Keys, mstSet);

        mstSet[u] = true;

        for(int v = 0; v < V; v++){

            if(graph[u][v] && mstSet[v] ==false&& graph[u][v] < Keys[v]){
                parent[v] = u;
                Keys[v] = graph[u][v];
            }
        }
    }
    printMst(graph, parent);

}


int main(){

    int graph[V][V] = { { 0, 0, 3, 0, 0},
                        { 0, 0, 10, 4, 0 },
                        { 3, 10, 0, 2, 6},
                        { 0, 4, 2, 0, 1},
                        { 0, 0, 6, 1, 0 } };

    // Print the solution
    primMst(graph);


    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;

bool isSafe(int **arr, int x, int y, int n){
    for(int row = 0; row < x; row++){
        if(arr[row][y] == 1){
            return false;
        }
    }

    int row = x;
    int col = y;
    cout<<"AB"<<endl;
    while(row >= 0 && col >=0){
        if(arr[row][col] == 1){
            return false;
        }
        row--;
        col--;
    }
```

```
    cout<<"CD"<<endl;
    row = x;
    col = y;

    while(row >= 0 && col < n){
        if(arr[row][col] == 1){
            return false;
        }
        row--;
        col++;
    }

    return true;
}

bool nQueens(int **arr, int x, int n){
    if(x >= n){
        return true;
    }
    cout<<"xyz"<<endl;
    for(int col = 0; col < n; col++){
        if(isSafe(arr, x, col, n)){
            arr[x][col] = 1;
            if(nQueens(arr, x+1, n)){
                return true;
            }
            arr[x][col] = 0;
        }
    }

    return false;
}

int main(){

    int n;
    cout<<"Enter number of Queens: ";
    cin>>n;

    int **arr = new int*[n];

    for(int i = 0;i <n;i++){
        arr[i] = new int[n];
        for(int j = 0; j < n; j++){
            arr[i][j] = 0;
        }
    }

    if(nQueens(arr, 0, n)){
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                cout<<arr[i][j]<<" ";
            }
            cout<<"\n";
        }
    }
    return 0;
```

```
}
```

```
def greet():
    return "Hi there! Welcome to PizzaBot."
def menu():
    return "Our menu includes: \n1. Margherita Pizza \n2. Pepperoni Pizza \n3. Vegetarian Pizza
\n4. Hawaiian Pizza"

def order_pizza(pizza_choice):
    return f"You have ordered {pizza_choice}. It will be delivered to your address shortly."

def delivery_time():
    return "Your pizza will arrive in 30 minutes."

def goodbye():
    return "Thank you for choosing PizzaBot. Have a great day!"

def respond(message):
    if "hi" in message.lower() or "hello" in message.lower():
        return greet()
    elif "menu" in message.lower():
        return menu()
    elif "order" in message.lower():
        if "order " in message.lower():
            pizza_choice = message.split("order ")[1]
            return order_pizza(pizza_choice)
        else:
            return "Please specify which pizza you'd like to order."
    elif "delivery" in message.lower() or "time" in message.lower():
        return delivery_time()
    elif "bye" in message.lower() or "thank you" in message.lower():
        return goodbye()
    else:
        return "I'm sorry, I didn't understand that."

def main():
    print("Welcome to PizzaBot! How can I assist you today?")
    while True:
        user_input = input("You: ")
        if user_input.lower() == "exit":
            print(goodbye())
            break
        else:
            print("PizzaBot:", respond(user_input))

if __name__ == "__main__":
    main()
```