"Techno – Social Excellence"

Marathwada Mitra Mandal's

INSTITUTE OF TECHNOLOGY (MMIT)

Lohgaon, Pune-411047

"Towards Ubiquitous Computing Technology"

DEPARTMENT OF COMPUTER ENGINEERING

LABORATORY MANUAL

310258: Laboratory Practice-II

TE Computer Engineering (2019 Course)

Semester - II

Prepared By

Mr. Vikas V. Chavan

310258: Laboratory Practice II

A. Course Outcome

Course Outcome	Statement
	At the end of the course, student will be able to
1	Design a system using different informed search / uninformed search or heuristic approaches
2	Apply basic principles of AI in solutions that require problem solving, inference, perception, knowledge representation, and learning
3	Design and develop an interactive AI application

B. CO-PO Mapping (Levels :1-Low , 2-Medium, 3-High)

Course Outcome	Program outcomes											
	1	2	3	4	5	6	7	8	9	10	11	12
310258.1	2	-	2		3			2	2	2	1	2
310258.2	1	-	2	2	3	2		2	2	2	1	2
310258.3	1	-	2	2	3	2		2	2	2	2	2

CO-PSO mapping

Course Outcome	Program Specific Outcomes		
	1	2	
310258.1	2	1	
310258.2	1	2	
310258.3	2	1	

Semester II Academic Year 2022-23

INDEX

Sr. No.	Title of Assignment	CO	PO
1	Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.	1	PO1,3,5,8,9,10, 11,12 PSO:1,2
2	Implement A star Algorithm for any game search problem.	1	PO1,3,5,8,9,10, 11,12 PSO:1,2
3	Implement Greedy search algorithm for any of the following application: I. Selection Sort II. Minimum Spanning Tree III. Single-Source Shortest Path Problem IV. Job Scheduling Problem V. Prim's Minimal Spanning Tree Algorithm VI. Kruskal's Minimal Spanning Tree Algorithm VII. Dijkstra's Minimal Spanning Tree Algorithm	1	PO1,3,5,8,9,10, 11,12 PSO:1,2
4	Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.	2	PO1,3,4,5,6,8,9 ,10,11,12 PSO:1,2
5	Develop an elementary catboat for any suitable customer interaction application.	3	PO1,3,4,5,6,8,9 ,10,11,12 PSO:1,2

310258: Laboratory Practice II

6	Implement any one of the following Expert System I. Information management II. Hospitals and medical facilities III. Help desks management IV. Employee performance evaluation V. Stock market trading VI. Airline scheduling and cargo schedules	3	PO1,3,4,5,6,8,9,10,11,12 PSO:1,2

310258: Laboratory Practice II

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

2. Problem Analysis: Identify, formulates, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design / development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research – based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

1. A graduate of the Computer Engineering Program will be able to Attain proficiency in analyzing and applying Computer Engineering Fundamentals viz. algorithms, system software's and networking for contemporary system development through advanced technologies like - Cyber Security, High Performance Computing and Data Analytics (Artificial Intelligence, Machine Learning) etc.
2. Identify, formulate and solve real world problems in societal and environmental contexts through lifelong learning, entrepreneurship and leadership skills.

LABORATORY MANUAL

310258: Laboratory Practice-II

Part 1: Artificial Intelligence

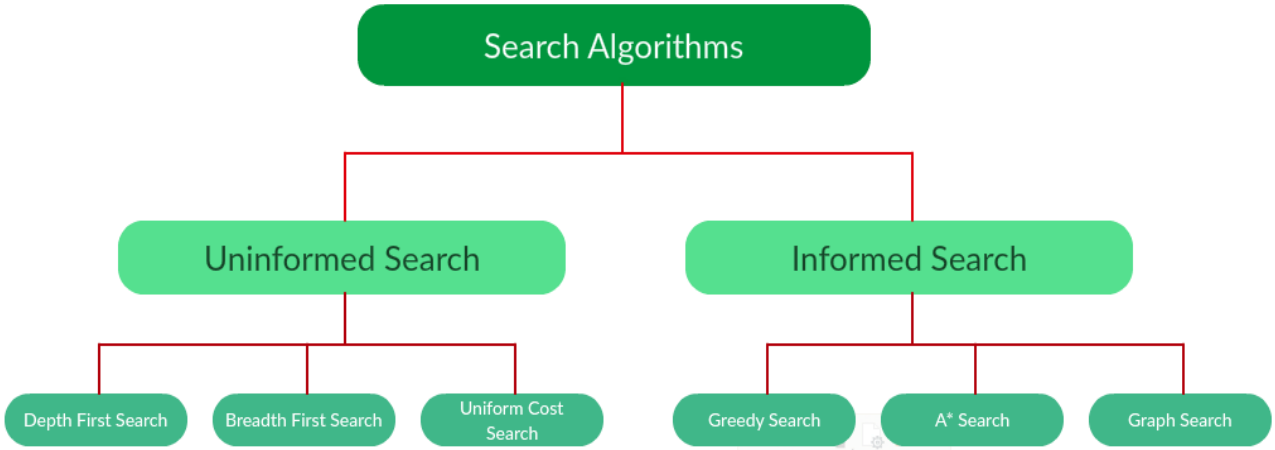
Assignment No. 1

TITLE: Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

OBJECTIVE: To understand the use of searching in artificial intelligence

THEORY:

- **Searching in AI:**
  - **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  - **Search Space:** Search space represents a set of possible solutions, which a system may have.
  - **Start State:** It is a state from where agent begins **the search**.
  - **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.
- Properties of Search Algorithms:
- Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:
- **Completeness:** A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.
- **Optimality:** If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.
- **Time Complexity:** Time complexity is a measure of time for an algorithm to complete its task.
- **Space Complexity:** It is the maximum storage space required at any point during the search, as the complexity of the problem. Types of search algorithms
- **Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.**



- **Uninformed Search:**
  The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search.It examines each node of the tree until it achieves the goal node

- **Informed Search:**
- Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.
  A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

- **Heuristic Search**
  It includes Blind Search, Uninformed Search, and Blind control strategy. These search techniques are not always possible as they require much memory and time. These techniques search the complete space for a solution and use the arbitrary ordering of operations.
  The examples of Direct Heuristic search techniques include Breadth-First Search (BFS) and Depth First Search (DFS).
  Weak Heuristic Search techniques in AI
  It includes Informed Search, Heuristic Search, and Heuristic control strategy. These techniques are helpful when they are applied properly to the right types of tasks. They usually require domain-specific information.
  The examples of Weak Heuristic search techniques include Best First Search (BFS) and A*.
  Before describing certain heuristic techniques, let's see some of the techniques listed below:

- Bidirectional Search
- A* search
- Simulated Annealing
- Hill Climbing
- Best First search

- Beam search

Algorithm:

```
//Graph ADT
classgraph
{
        structnode      //create structure of node
        {
                intdata;
                node *next;
        };
public:
        node *head[20];
        stack<node*>stack;
        queue<node*>Q;
        intvisited[20], no;
        voiddisplay();
        voidedge_v1_v2(int, int);
        voidedge_v2_v1(int, int);
        voidDFS();
        voidBFS();
};

//Depth First Search Algorithm
DFS()
{
  node *v, *f;
  while(!empty())
  {
     f = pop();
     if(visited[f->data] == 0)
     {
                visited[f->data] = 1;
                v = head[f->data];
                cout<<" "<<v->data;
                v = v->next;
                while(v)
                {
                        push(v);
                        v = v->next;
                }
        }
  }
}
```

```
//Breadth First Search
BFS()
{
  node *v, *f;
  while(!Qempty())
  {
     f = dequeue();
     if(visited[f->data] == 0)
     {
                visited[f->data] = 1;
                v = head[f->data];
                cout<<" "<<v->data;
                v = v->next;
                while(v)
                {
                        queue(v);
                        v = v->next;
                }
        }

  }
}

//Insert an Edge
edge_v1_v2(int v1, int v2)
{
        node *n, *f, *h;

        //creating link from v1 to v2
        h = newnode;
        h->data = v1;
        h->next = NULL;

        n = newnode;  //allocate memory for new node
        n->data = v2;  //assign vertex
        n->next = NULL;       //next link will be NULL

        f = head[v1];   //head vertex is v1

        if(f == NULL)
        {
                h->next = n;
                head[v1] = h;
```

```
        }
        else
        {
                while(f->next)
                    f = f->next;
                f->next = n;
        }

}

//Display Adjacency List
display()
{
        int i;
        node *f;
        for(i = 0 ; i <no; i++)
        {
                cout<<endl;
                f = head[i];
                while(f)
                {
                        cout<<"\t"<<f->data;
                        f = f->next;
                }
        }
}
```

FAQS:

1. State different search strategy used in AI
2. What is mean by uninformed search? List uninformed search algorithms.
3. What is mean by informed search?

Conclusion:

In this assignment we have successfully implemented the DFS and BFS.

Assignment No.2

TITLE: Implement A-star Algorithm for any game search problem

OBJECTIVE: To use A-star Algorithm for optimal searching

THEORY:

A* Algorithm:

What is an A* Algorithm?

It is a searching algorithm that is used to find the shortest path between an initial and a final point. It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal. It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem. Another aspect that makes A* so powerful is the use of weighted graphs in its implementation.

A weighted graph uses numbers to represent the cost of taking each path or course of action. This means that the algorithms can take the path with the least cost, and find the best route in terms of distance and time.

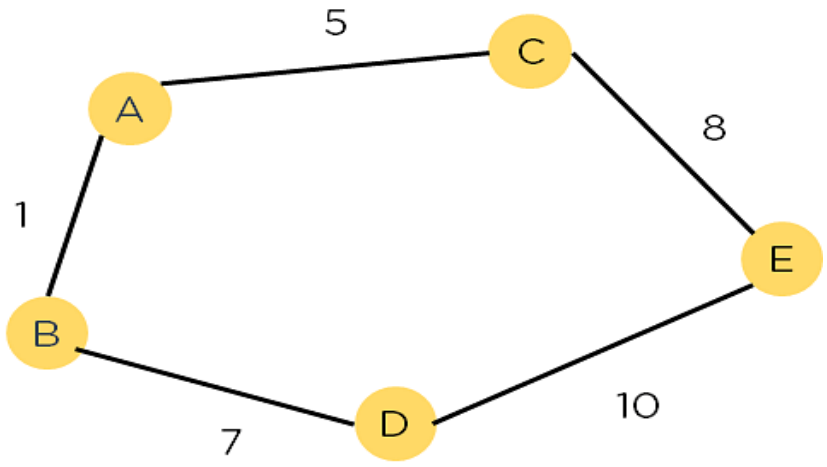


Figure 1: Weighted Graph

A major drawback of the algorithm is its space and time complexity. It takes a large amount of space to store all possible paths and a lot of time to find them.

The Basic Concept of A* Algorithm

A heuristic algorithm sacrifices optimality, with precision and accuracy for speed, to solve problems faster and more efficiently. All graphs have different nodes or points which the algorithm has to take, to reach the final node. The paths between these nodes all have a

numerical value, which is considered as the weight of the path. The total of all paths transverse gives you the cost of that route.

Initially, the Algorithm calculates the cost to all its immediate neighboring nodes, n, and chooses the one incurring the least cost. This process repeats until no new nodes can be chosen and all paths have been traversed. Then, you should consider the best path among them. If f(n) represents the final cost, then it can be denoted as :

f(n) = g(n) + h(n), where :

g(n) = cost of traversing from one node to another. This will vary from node to node

h(n) = heuristic approximation of the node's value. This is not a real value but an approximation cost

How Does the A* Algorithm Work?

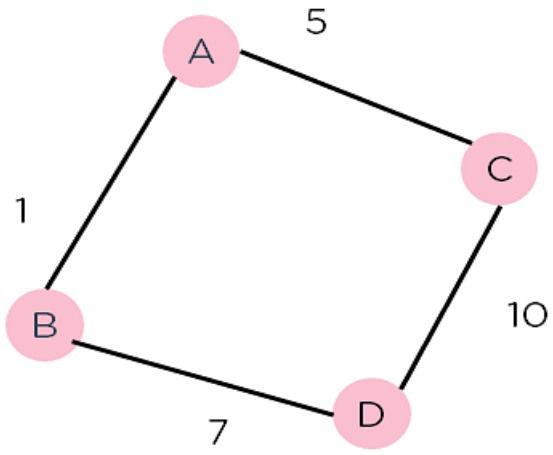


Figure 2: Weighted Graph 2

Consider the weighted graph depicted above, which contains nodes and the distance between them. Let's say you start from A and have to go to D. Now, since the start is at the source A, which will have some initial heuristic value. Hence, the results are

f(A) = g(A) + h(A)

f(A) = 0 + 6 = 6

Next, take the path to other neighboring vertices :

f(A-B) = 1 + 4

f(A-C) = 5 + 2

Now take the path to the destination from these nodes, and calculate the weights :

f(A-B-D) = (1+ 7) + 0

f(A-C-D) = (5 + 10) + 0

It is clear that node B gives you the best path, so that is the node you need to take to reach the destination.

Admissibility of A* Algorithm

- A search algorithm is admissible if, for any graph, it always terminates in an optimal path (if it exists), from initial state to goal state.
- Thus, A search algorithm is said to be *admissible*, if it is guaranteed to return an optimal solution.
- A **heuristic** "h(n)" is admissible, if for every node n,

  h(n) ≤ h*(n),

  where h*(n) = True cost to reach the goal state from n.
- When A* terminates its search, it has found a path, from start to goal, whose actual cost is lower than the estimated cost of any path, through any open node.
- An admissible heuristic never overestimates the cost to reach the goal, i.e. it is optimistic.

- **Algorithms:**
  //A-Star Algorithm to find the shortest path of graph

```
classgraph
{
        structnode      //create structure of node
        {
                intdata;
                intweight;
                node *next;
        };
public:
        node *head[20];
        intHeuristic[20], no, fn, gn;
        voiddisplay();
        voidedge_v1_v2(int, int, int);
        voidAstar(int , int);
};
voidgraph :: edge_v1_v2(int v1, int v2, int w)
{
        node *n, *f, *h;

        //creating link from v1 to v2
        h = newnode;
        h->data = v1;
        h->next = NULL;

        n = newnode;  //allocate memory for new node
        n->data = v2;  //assign vertex
        n->weight = w;
        n->next = NULL;       //next link will be NULL

        f = head[v1];   //head vertex is v1
```

```
        if(f == NULL)
        {
                h->next = n;
                head[v1] = h;
        }
        else
        {
                while(f->next)
                    f = f->next;
                f->next = n;
        }

}

voidgraph :: display()
{
        int i;
        node *f;
        for(i = 0 ; i <no; i++)
        {
                cout<<endl;
                f = head[i];
                while(f)
                {
                        cout<<"\t"<<f->data;
                        f = f->next;
                }
        }
}
voidgraph :: Astar(int src, int goal){
        node *f;
        gn = 0;
        f = head[src];
        cout<<"\n Shortest Path is :";
        cout<<src<<"\t";
        while(src != goal){
                f = head[src];
                f = f->next;
                fn = Max;
                while(f){
                        if(fn> (gn + f->weight + Heuristic[f->data])){
                                gn = gn + f->weight;
                                fn = gn + Heuristic[f->data];
                                src = f->data;
```

```
                    }
                    f = f->next;
                }
                cout<<src<<"\t";
            }
        }
```

FAQ's :

1. What is heuristic search?
2. How to find heuristic function?
3. How the A* algorithm works?

CONCLUSION:

In this assignment we have successfully implemented the A-star Algorithm

Assignment No. 3

TITLE: Implement Greedy search algorithm for Prim's Minimal Spanning Tree Algorithm

OBJECTIVE: To use Greedy Algorithm for searching a solution of a given problem

THEORY:

Greedy Algorithm

The greedy method is one of the strategies like Divide and conquer used to solve the problems. This method is used for solving optimization problems. An optimization problem is a problem that demands either maximum or minimum results. Let's understand through some terms. The Greedy method is the simplest and straightforward approach. It is not an algorithm, but it is a technique. The main function of this approach is that the decision is taken on the basis of the currently available information. This technique is basically used to determine the feasible solution that may or may not be optimal. The feasible solution is a subset that satisfies the given criteria. The optimal solution is the solution which is the best and the most favorable solution in the subset. In the case of feasible, if more than one solution satisfies the given criteria then those solutions will be considered as the feasible, whereas the optimal solution is the best solution among all the solutions.

Pseudo code of Greedy Algorithm

```
Algorithm Greedy (a, n)
{
  Solution : = 0;
 for i = 0 to n do
 {
    x: = select(a);
   if feasible(solution, x)
  {
     Solution: = union(solution , x)
  }
    return solution;
 }
}
```

The above is the greedy algorithm. Initially, the solution is assigned with zero value. We pass the array and number of elements in the greedy algorithm. Inside the for loop, we select the element one by one and checks whether the solution is feasible or not. If the solution is feasible, then we perform the union.

Minimal Spanning Tree:

A **minimum spanning tree** (**MST**) or **minimum weight spanning tree** is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.[2] More generally, any edge-weighted undirected graph (not necessarily connected) has a **minimum spanning forest**, which is a union of the minimum spanning trees for its connected components.

Prim's Algorithm:

It is a greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices:

- Contain vertices already included in MST.
- Contain vertices not yet included.

At every step, it considers all the edges and picks the minimum weight edge. After picking the edge, it moves the other endpoint of edge to set containing MST.

Spanning tree - A spanning tree is the subgraph of an undirected connected graph.

Minimum Spanning tree - Minimum spanning tree can be defined as the spanning tree in which the sum of the weights of the edge is minimum. The weight of the spanning tree is the sum of the weights given to the edges of the spanning tree.

Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

How does the prim's algorithm work?

Prim's algorithm is a greedy algorithm that starts from one vertex and continue to add the edges with the smallest weight until the goal is reached. The steps to implement the prim's algorithm are given as follows -

- First, we have to initialize an MST with the randomly chosen vertex.
- Now, we have to find all the edges that connect the tree in the above step with the new vertices.

From the edges found, select the minimum edge and add it to the tree.

- Repeat step 2 until the minimum spanning tree is formed.

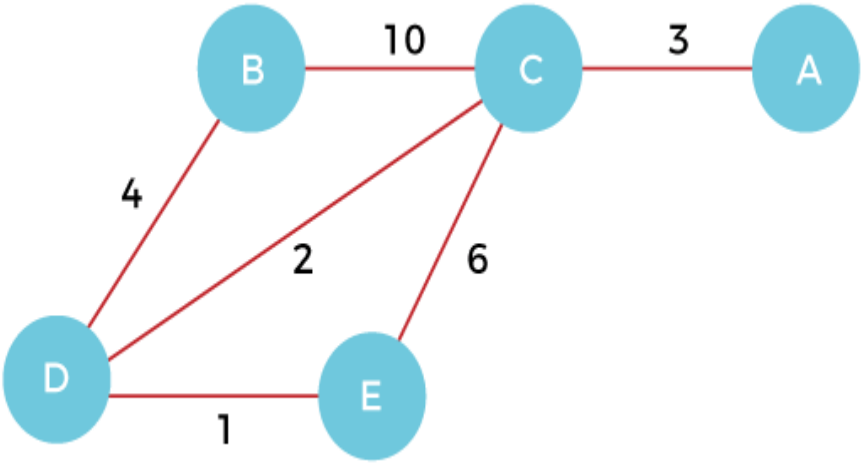
The applications of prim's algorithm are -

- Prim's algorithm can be used in network designing.
- It can be used to make network cycles.
- It can also be used to lay down electrical wiring cables.

Example of prim's algorithm

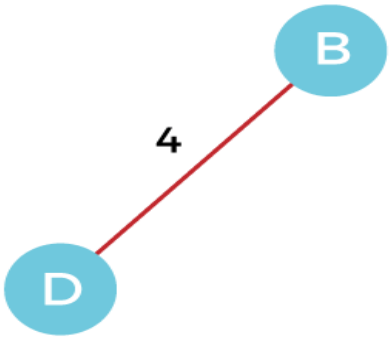
Now, let's see the working of prim's algorithm using an example. It will be easier to understand the prim's algorithm using an example.

Suppose, a weighted graph is -

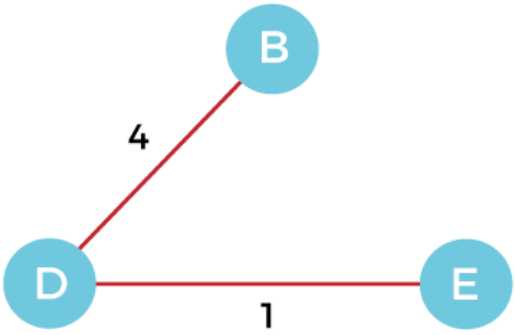


Step 1 - First, we have to choose a vertex from the above graph. Let's choose B.

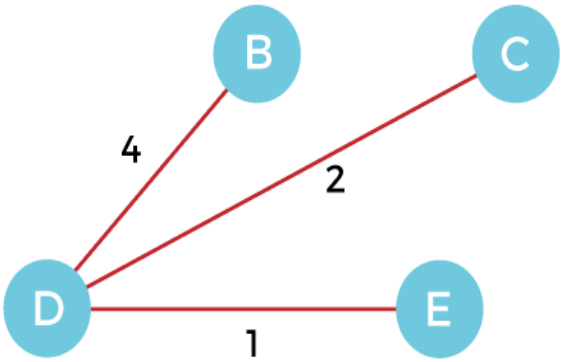
Step 2 - Now, we have to choose and add the shortest edge from vertex B. There are two edges from vertex B that are B to C with weight 10 and edge B to D with weight 4. Among the edges, the edge BD has the minimum weight. So, add it to the MST.



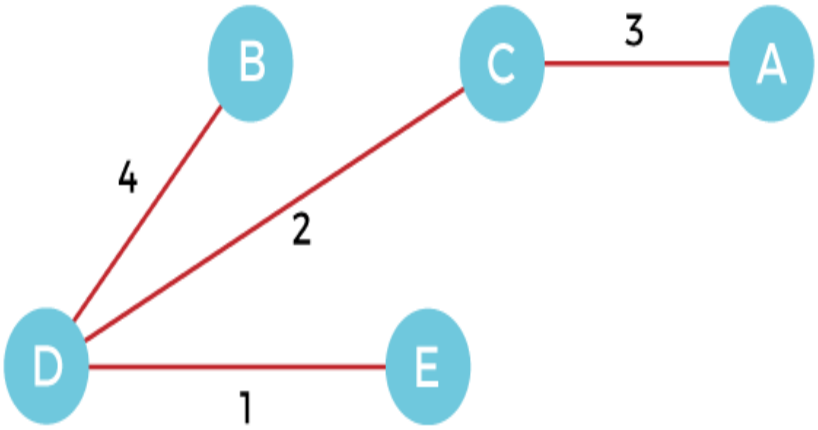
Step 3 - Now, again, choose the edge with the minimum weight among all the other edges. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C, i.e., E and A. So, select the edge DE and add it to the MST.



Step 4 - Now, select the edge CD, and add it to the MST.



Step 5 - Now, choose the edge CA. Here, we cannot select the edge CE as it would create a cycle to the graph. So, choose the edge CA and add it to the MST.



So, the graph produced in step 5 is the minimum spanning tree of the given graph. The cost of the MST is given below -
Cost of MST = 4 + 2 + 1 + 3 = 10 units.

Algorithms:

//A-Star Algorithm to find the shortest path of graph

```
classgraph
{
        structnode      //create structure of node
        {
                intdata;
                intweight;
                node *next;
        };
public:
        node *head[20];
        intno;
        voiddisplay();
        voidedge_v1_v2(int, int, int);
        voidPrims();
};

voidgraph :: edge_v1_v2(int v1, int v2, int w)
{
        node *n, *f, *h;

        //creating link from v1 to v2
        h = newnode;
        h->data = v1;
        h->next = NULL;

        n = newnode;  //allocate memory for new node
        n->data = v2;  //assign vertex
```

```
        n->weight = w;
        n->next = NULL;       //next link will be NULL

        f = head[v1];   //head vertex is v1

        if(f == NULL)
        {
                h->next = n;
                head[v1] = h;
        }
        else
        {
                while(f->next)
                    f = f->next;
                f->next = n;
        }

}

voidgraph :: display()
{
        int i;
        node *f;
        for(i = 0 ; i <no; i++)
        {
                cout<<endl;
                f = head[i];
                while(f)
                {
                        cout<<"\t"<<f->data;
                        f = f->next;
                }
        }
}

Prims()
{
        node *p;
        int m, min_wt=0;
        int v, v1, v2, wt;
        int i,j;
   int visited[10], vertex[10];

        //Initialization
```

```
        for(i=0; i<no; i++)
        {
                visited[i]=0;   //identify whether vertex visited
                vertex[i]=-1;   //visited vertex inserted
        }
        vertex[0] = 0;
        visited[0] = 1;
        cout<<"\nMinimum Spanning Tree with minimum cost is\n";
        cout<<head[0]->data;
        for(i = 0; i<no-1; i++)  //No. of vertices to visit
        {
           m = 9999;       //set max value to find minimum cost
                for(j=0; j<=i;j++)      //check all edges of vistited vertex
                {
                        v = vertex[j];          //get element from set of vertex
                        p = head[v];            //get header node
                p = p->next;
                        while(p)                //if p != NULL
                        {
                              if(visited[p->data] == 0)//identify whether vertex visited
                                 {
                                        wt = p->weight;
                                        if(m > wt)
                                        {
                                                m = wt;
                                                v1 = v;
                                                v2 = p->data;
                                        }
                                }
                                p = p->next;
                        }
                }
                cout<<" "<<head[v2]->data;
                min_wt += m;
                vertex[j] = v2;
                visited[v2] = 1;
        }
        cout<<"\n Minimum cost is "<<min_wt<<endl;
        cout<<endl;
}
```

FAQ's:

1. What is spanning tree?
2. What is minimum spanning tree?
3. Explain Prim's Algorithm
4. Explain Greedy Algorithm
5. What is the complexity of Prim's and Krushkal's Algorithm?

Conclusion:

In this assignment we have successfully implemented the Minimum Spanning Tree with Prim's Algorithm

Assignment No. 4

TITLE: Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem

OBJECTIVE: To implement a solution for a constraint satisfaction problem

THEORY:

Constraint Satisfaction Problem:

Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem. Such type of technique leads to a deeper understanding of the problem structure as well as its complexity. Constraint satisfaction depends on three components, namely:

- X: It is a set of variables.
- D: It is a set of domains where the variables reside. There is a specific domain for each variable.
- C: It is a set of constraints which are followed by the set of variables.

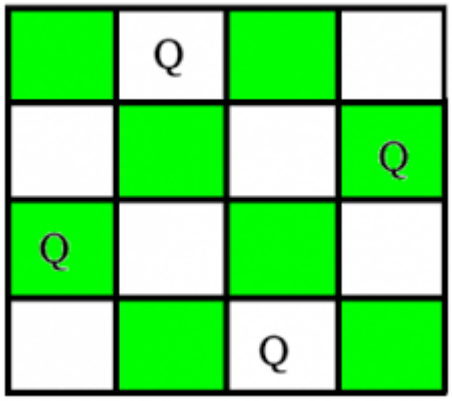
Types of Domains in CSP

There are following two types of domains which are used by the variables :

- Discrete Domain: It is an infinite domain which can have one state for multiple variables. For example, a start state can be allocated infinite times for each variable.
- Finite Domain: It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

N—Queens Problem:

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



The expected output is a binary matrix which has 1s for the blocks where queens are placed. For example, following is the output matrix for above 4 queen solution.

{ 0, 1, 0, 0}
{ 0, 0, 0, 1}
{ 1, 0, 0, 0}
{ 0, 0, 1, 0}

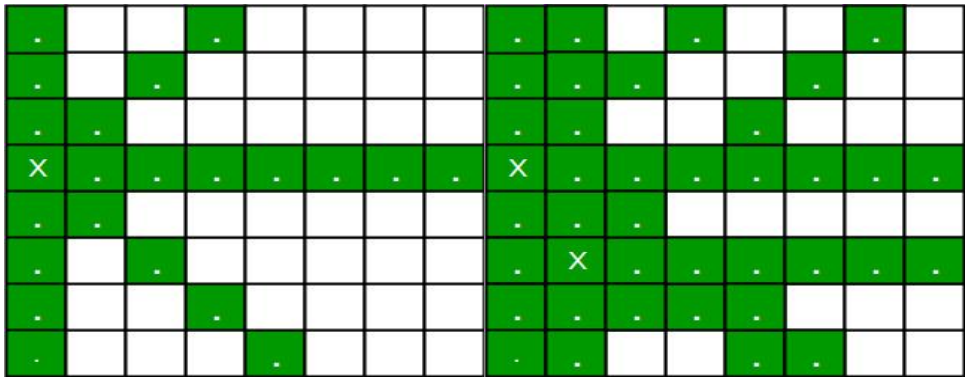
Backtracking Algorithm: The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes

with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

Branch and bound is an algorithm:

Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. The N queens puzzle is the problem of placing N chess queens on an N×N chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. In Branch and Bound solution, after building a partial solution, we figure out that there is no point going any deeper as we are going to hit a dead end.

Let's begin by describing backtracking solution. "The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false."



For the 1st Queen, there are total 8 possibilities as we can place 1st Queen in any row of first column. Let's place Queen 1 on row 3. After placing 1st Queen, there are 7 possibilities left for the 2nd Queen. But wait, we don't really have 7 possibilities. We cannot place Queen 2 on rows 2, 3 or 4 as those cells are under attack from Queen 1. So, Queen 2 has only 8 – 3 = 5 valid positions left. After picking a position for Queen 2, Queen 3 has even fewer options as most of the cells in its column are under attack from the first 2 Queens. We need to figure out an efficient way of keeping track of which cells are under attack. In previous solution we kept an 8--by--8 Boolean matrix and update it each time we placed a queen, but that required linear time to update as we need to check for safe cells.

Basically, we have to ensure 4 things:

1. No two queens share a column.
2. No two queens share a row.
3. No two queens share a top-right to left-bottom diagonal.
4. No two queens share a top-left to bottom-right diagonal.

Number 1 is automatic because of the way we store the solution. For number 2, 3 and 4, we can perform updates in O(1) time. The idea is to keep three Boolean arrays that tell us which rows and which diagonals are occupied.

Algorithms:

```
//Placing an Queen in k row and i column

int place(int k, int i)
{
        int j;
        for(j = 1; j<=k-1; j++)
        {
                if(x[j] == i || abs(x[j]-i) == abs(j - k))
                        return 1;
        }
        return 0;
}

//Backtracking Algorithm for N-Queens
void NQueens(intk, int n)
{
        int i, j;
        for(i = 1; i<=n; i++)
        {
                if(place(k, i) == 0)
                {
                        x[k] = i;
                        if(k == n)
                        {
                                for(j = 1; j<=n; j++)
                                        cout<<" "<<x[j];
                                cout<<"\n";
                        }
                        else
                                NQueens(k+1, n);
                }
        }
}
```

FAQ's:

1. Explain N-Queens problem
2. What is constraint satisfaction problem?
3. Explain CSP using N-Queens problem.
4. Explain Backtracking algorithmic strategy
5. How the branch and bound algorithm works?

CONCLUSION : In this assignment we have successfully implemented the N-Queens problem using backtracking and branch and bound

Assignment No. 5

TITLE: Develop elementary chatboat for any suitable customer interaction application
OBJECTIVE: To implement a solution chatboat for a customer interaction application

THEORY:

What is a Chatbot?

A chatbot is an AI-based software designed to interact with humans in their natural languages. These chatbots are usually converse via auditory or textual methods, and they can effortlessly mimic human languages to communicate with human beings in a human-like manner. A chatbot is arguably one of the best applications of natural language processing.The Rule-based approach trains a chatbot to answer questions based on a set of pre-determined rules on which it was initially trained. These set rules can either be very simple or very complex. While rule-based chatbots can handle simple queries quite well, they usually fail to process more complicated queries/requests. As the name suggests, self-learning bots are chatbots that can learn on their own. These leverage advanced technologies like Artificial Intelligence and Machine Learning to train themselves from instances and behaviors. Naturally, these chatbots are much smarter than rule-based bots. Self-learning bots can be further divided into two categories – Retrieval Based or Generative.

1. Retrieval-based Chatbots:

A retrieval-based chatbot is one that functions on predefined input patterns and set responses. Once the question/pattern is entered, the chatbot uses a heuristic approach to deliver the appropriate response. The retrieval-based model is extensively used to design goal-oriented chatbots with customized features like the flow and tone of the bot to enhance the customer experience.

2. Generative Chatbots :

Unlike retrieval-based chatbots, generative chatbots are not based on predefined responses – they leverage seq2seq neural networks. This is based on the concept of machine translation where the source code is translated from one language to another language. In seq2seq approach, the input is transformed into an output. The first chatbot dates back to 1966 when Joseph Weizenbaum created ELIZA that could imitate the language of a psychotherapist in only 200 lines of code. However, thanks to the rapid advancement of technology, we've come a long way from scripted chatbots to chatbots in python today.

Chatbot in Today's Generation:

Today, we have smart AI-powered Chatbots that use natural language processing (NLP) to understand human commands (text and voice) and learn from experience. Chatbots have become a staple customer interaction tool for companies and brands that have an active online presence (website and social network platforms). Chatbots using python are a nifty tool since they facilitate instant messaging between the brand and the customer. Think about Apple's Siri, Amazon's Alexa, and Microsoft's Cortana. Aren't these just wonderful? Aren't you already curious to learn how to make a chatbot in Python? ChatterBot is a Python library that is designed to deliver automated responses to user inputs. It makes use of a combination of ML algorithms to generate many different types of responses. This feature allows developers to build chatbots using python that can converse with humans and deliver appropriate and relevant responses. Not just that, the ML algorithms help the bot to improve its performance with experience. Another excellent feature of ChatterBot is its language independence. The library is designed in a way that makes it possible to train your bot in multiple programming languages.

How does ChatterBot function?

When a user enters a specific input in the chatbot (developed on ChatterBot), the bot saves the input along with the response, for future use. This data (of collected experiences) allows the chatbot to generate automated responses each time a new input is fed into it.
The program chooses the most-fitting response from the closest statement that matches the input, and then delivers a response from the already known selection of statements and responses. Over time, as the chatbot engages in more interactions, the accuracy of response improves.

Algorithm:

#Import Classes

```
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
```

 # Create and Train the Chatbot

```
my_bot = ChatBot
(name='PyBot',read_onsly=True,logic_adapter=['chatterbot.logic.MathematicalEvaluation',
'chatterbot.logic.BestMatch'])

small_talk=['hi there',
'hi', 'how do you do',
'how are you',
'I am cool',
'I\'m fine',
'glad to hear that']
math_talk1= ['pythagorean theorem',
'a square plus b square equal c square']
math_talk2 = ['law of cosine', 'c**2=a**2+b**2-2*a*b*cos(gamma)']
list_trainer = ListTrainer(my_bot)
for item in(small_talk, math_talk1, math_talk2):
  list_trainer.train(item)
```

#Communicate with the Python Chatbot

```
print(my_bot.get_response("hi"))
print(my_bot.get_response("how are you"))
print(my_bot.get_response("pythagorean theorem"))
```

FAQS:

1) What is a Chatbot? why is important ?
2) How does ChatterBot function?
3) How to Make a Chatbot in Python

CONCLUSION:

In this assignment we have successfully implemented the chatboat for customer interaction application.

Assignment No. 6

TITLE: Implement the Expert System for Hospital and Medical Facilities.

OBJECTIVE: To understand the use of Expert system for Hospital and Medical Facilities

THEORY:

What is an Expert System?

In artificial intelligence (AI), an expert system is a computer-based decision-making system. It is designed to solve complex problems. To do so, it applies knowledge and logical reasoning and adheres to certain rules. An expert system is one of the first successful forms of artificial intelligence

Component Of Expert System:

- **User interface** – It is the most important part of the expert system software. The user interface transfers the queries of the user and into the inference engine. Then it shows the results to the user. It acts as a two-way communicator between the expert system and the user.

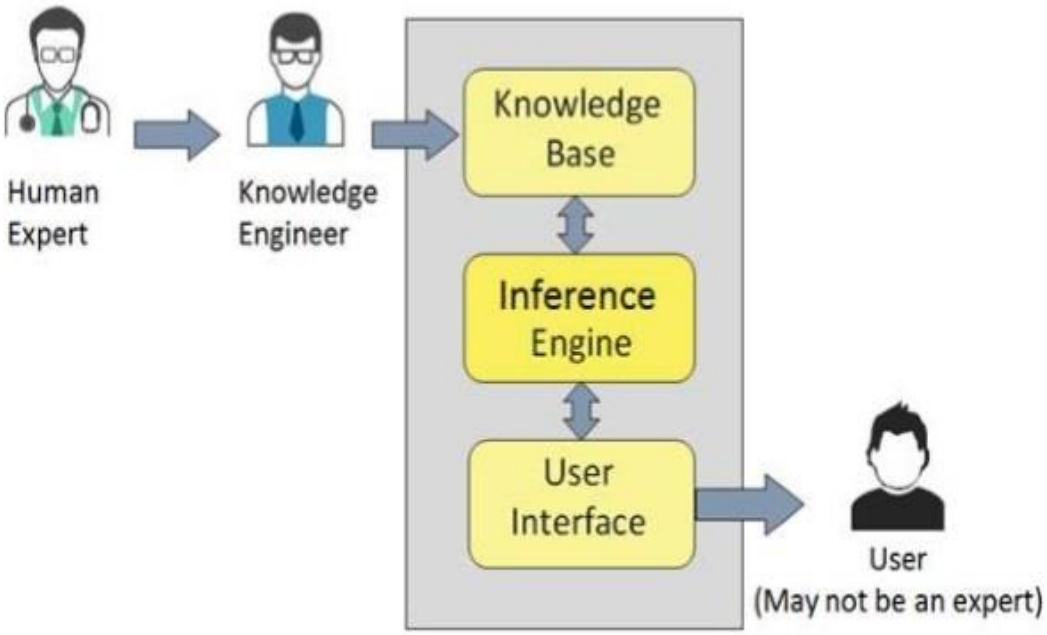


Fig: Component Of Expert System

- **Inference engine** – The inference engine is the central processing unit of the expert system. An inference engine works on rules and regulations to solve complex problems. It uses information from the knowledge base. It smartly selects factual data and rules, and processes and applies them to answer the user's query. It also gives proper reasoning about the data in the knowledge base. This helps detect and deduce complex problems and prevents recurrence. And the last, the inference engine formulates conclusions.

The inference engine has the following strategies:

Forward chaining – Answers the question, "What can happen in the future?"

1. Backward chaining – Answers the question, "Why did this happen?"

- **Knowledge base** – The knowledge base is the information center. It contains all the information about problem domains. It is like a large repository of information collected from various experts.

Knowledge Base Components

Factual and heuristic knowledge is stored in the knowledge base.

- Factual Knowledge − Information pertaining to knowledge engineers.
- Heuristic Knowledge − Ability to evaluate and guess.

Capabilities Of Expert System:

Other Key Terms used in Expert System:

Apart from the expert system components listed above, the following terms are also extensively used when discussing expert systems.

- **Facts and rules** – A fact is a small piece of important knowledge. Facts have limited use. An expert system selects the rules to solve a problem.
- **Knowledge acquisition** – Knowledge acquisition refers to the method used to extract domain-specific information by an expert system. The process begins by acquiring knowledge from a human expert, converting human knowledge into facts and rules, and finally feeding those rules into the knowledge base.

Algorithms:

```
from pyknow import *
diseases_list = []
diseases_symptoms = []
symptom_map = {}
d_desc_map = {}
d_treatment_map = {}

def preprocess():
    global diseases_list,diseases_symptoms,symptom_map,d_desc_map,d_treatment_map
    diseases = open("diseases.txt")
    diseases_t = diseases.read()
    diseases_list = diseases_t.split("\n")
    diseases.close()
    for disease in diseases_list:
        disease_s_file = open("Disease symptoms/" + disease + ".txt")
        disease_s_data = disease_s_file.read()
        s_list = disease_s_data.split("\n")
        diseases_symptoms.append(s_list)
        symptom_map[str(s_list)] = disease
        disease_s_file.close()
        disease_s_file = open("Disease descriptions/" + disease + ".txt")
        disease_s_data = disease_s_file.read()
        d_desc_map[disease] = disease_s_data
        disease_s_file.close()
        disease_s_file = open("Disease treatments/" + disease + ".txt")
        disease_s_data = disease_s_file.read()
```

```
                d_treatment_map[disease] = disease_s_data
                disease_s_file.close()


def identify_disease(*arguments):
        symptom_list = []
        for symptom in arguments:
                symptom_list.append(symptom)
        # Handle key error
        return symptom_map[str(symptom_list)]

def get_details(disease):
        return d_desc_map[disease]

def get_treatments(disease):
        return d_treatment_map[disease]

def if_not_matched(disease):
                print("")
                id_disease = disease
                disease_details = get_details(id_disease)
                treatments = get_treatments(id_disease)
                print("")
                print("The most probable disease that you have is %s\n" %(id_disease))
                print("A short description of the disease is given below :\n")
                print(disease_details+"\n")
                print("The common medications and procedures suggested by other real doctors
are: \n")
                print(treatments+"\n")

# @my_decorator is just a way of saying just_some_function =
my_decorator(just_some_function)
#def identify_disease(headache, back_pain, chest_pain, cough, fainting, sore_throat, fatigue,
restlessness,low_body_temp ,fever,sunken_eyes):
class Greetings(KnowledgeEngine):
        @DefFacts()
        def _initial_action(self):
                print("")
                print("Hi! I am Dr.Yar, I am here to help you make your health better.")
                print("For that you'll have to answer a few questions about your conditions")
                print("Do you feel any of the following symptoms:")
                print("")
                yield Fact(action="find_disease")
```

```
@Rule(Fact(action='find_disease'), NOT(Fact(headache=W())),salience = 1)
def symptom_0(self):
        self.declare(Fact(headache=input("headache: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(back_pain=W())),salience = 1)
def symptom_1(self):
        self.declare(Fact(back_pain=input("back pain: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(chest_pain=W())),salience = 1)
def symptom_2(self):
        self.declare(Fact(chest_pain=input("chest pain: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(cough=W())),salience = 1)
def symptom_3(self):
        self.declare(Fact(cough=input("cough: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(fainting=W())),salience = 1)
def symptom_4(self):
        self.declare(Fact(fainting=input("fainting: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(fatigue=W())),salience = 1)
def symptom_5(self):
        self.declare(Fact(fatigue=input("fatigue: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(sunken_eyes=W())),salience = 1)
def symptom_6(self):
        self.declare(Fact(sunken_eyes=input("sunken eyes: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(low_body_temp=W())),salience = 1)
def symptom_7(self):
        self.declare(Fact(low_body_temp=input("low body temperature: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(restlessness=W())),salience = 1)
def symptom_8(self):
        self.declare(Fact(restlessness=input("restlessness: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(sore_throat=W())),salience = 1)
def symptom_9(self):
        self.declare(Fact(sore_throat=input("sore throat: ")))

@Rule(Fact(action='find_disease'), NOT(Fact(fever=W())),salience = 1)
def symptom_10(self):
        self.declare(Fact(fever=input("fever: ")))
```

```
    @Rule(Fact(action='find_disease'), NOT(Fact(nausea=W())),salience = 1)
    def symptom_11(self):
        self.declare(Fact(nausea=input("Nausea: ")))

    @Rule(Fact(action='find_disease'), NOT(Fact(blurred_vision=W())),salience = 1)
    def symptom_12(self):
        self.declare(Fact(blurred_vision=input("blurred_vision: ")))

    @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="no"),Fact(che
st_pain="no"),Fact(cough="no"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="yes"
),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="yes"),Fact(sunken_eyes="no
"),Fact(nausea="yes"),Fact(blurred_vision="no"))
    def disease_0(self):
        self.declare(Fact(disease="Jaundice"))

    @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="no"),Fact(che
st_pain="no"),Fact(cough="no"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="no")
,Fact(restlessness="yes"),Fact(low_body_temp="no"),Fact(fever="no"),Fact(sunken_eyes="no"
),Fact(nausea="no"),Fact(blurred_vision="no"))
    def disease_1(self):
        self.declare(Fact(disease="Alzheimers"))

    @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="yes"),Fact(ch
est_pain="no"),Fact(cough="no"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="yes
"),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="no"),Fact(sunken_eyes="no
"),Fact(nausea="no"),Fact(blurred_vision="no"))
    def disease_2(self):
        self.declare(Fact(disease="Arthritis"))

    @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="no"),Fact(che
st_pain="yes"),Fact(cough="yes"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="no
"),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="yes"),Fact(sunken_eyes="n
o"),Fact(nausea="no"),Fact(blurred_vision="no"))
    def disease_3(self):
        self.declare(Fact(disease="Tuberculosis"))

    @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="no"),Fact(che
st_pain="yes"),Fact(cough="yes"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="no
"),Fact(restlessness="yes"),Fact(low_body_temp="no"),Fact(fever="no"),Fact(sunken_eyes="n
o"),Fact(nausea="no"),Fact(blurred_vision="no"))
    def disease_4(self):
        self.declare(Fact(disease="Asthma"))
```

```
    @Rule(Fact(action='find_disease'),Fact(headache="yes"),Fact(back_pain="no"),Fact(ch
est_pain="no"),Fact(cough="yes"),Fact(fainting="no"),Fact(sore_throat="yes"),Fact(fatigue="n
o"),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="yes"),Fact(sunken_eyes="
no"),Fact(nausea="no"),Fact(blurred_vision="no"))
    def disease_5(self):
        self.declare(Fact(disease="Sinusitis"))

    @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="no"),Fact(che
st_pain="no"),Fact(cough="no"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="yes"
),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="no"),Fact(sunken_eyes="no"
),Fact(nausea="no"),Fact(blurred_vision="no"))
    def disease_6(self):
        self.declare(Fact(disease="Epilepsy"))

    @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="no"),Fact(che
st_pain="yes"),Fact(cough="no"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="no"
),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="no"),Fact(sunken_eyes="no"
),Fact(nausea="yes"),Fact(blurred_vision="no"))
    def disease_7(self):
        self.declare(Fact(disease="Heart Disease"))

    @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="no"),Fact(che
st_pain="no"),Fact(cough="no"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="yes"
),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="no"),Fact(sunken_eyes="no"
),Fact(nausea="yes"),Fact(blurred_vision="yes"))
    def disease_8(self):
        self.declare(Fact(disease="Diabetes"))

    @Rule(Fact(action='find_disease'),Fact(headache="yes"),Fact(back_pain="no"),Fact(ch
est_pain="no"),Fact(cough="no"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="no"
),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="no"),Fact(sunken_eyes="no"
),Fact(nausea="yes"),Fact(blurred_vision="yes"))
    def disease_9(self):
        self.declare(Fact(disease="Glaucoma"))

    @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="no"),Fact(che
st_pain="no"),Fact(cough="no"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="yes"
),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="no"),Fact(sunken_eyes="no"
),Fact(nausea="yes"),Fact(blurred_vision="no"))
    def disease_10(self):
        self.declare(Fact(disease="Hyperthyroidism"))

    @Rule(Fact(action='find_disease'),Fact(headache="yes"),Fact(back_pain="no"),Fact(ch
est_pain="no"),Fact(cough="no"),Fact(fainting="no"),Fact(sore_throat="no"),Fact(fatigue="no"
```

```
),Fact(restlessness="no"),Fact(low_body_temp="no"),Fact(fever="yes"),Fact(sunken_eyes="no
"),Fact(nausea="yes"),Fact(blurred_vision="no"))
        def disease_11(self):
                self.declare(Fact(disease="Heat Stroke"))

        @Rule(Fact(action='find_disease'),Fact(headache="no"),Fact(back_pain="no"),Fact(che
st_pain="no"),Fact(cough="no"),Fact(fainting="yes"),Fact(sore_throat="no"),Fact(fatigue="no"
),Fact(restlessness="no"),Fact(low_body_temp="yes"),Fact(fever="no"),Fact(sunken_eyes="no
"),Fact(nausea="no"),Fact(blurred_vision="no"))
        def disease_12(self):
                self.declare(Fact(disease="Hypothermia"))

        @Rule(Fact(action='find_disease'),Fact(disease=MATCH.disease),salience = -998)
        def disease(self, disease):
                print("")
                id_disease = disease
                disease_details = get_details(id_disease)
                treatments = get_treatments(id_disease)
                print("")
                print("The most probable disease that you have is %s\n" %(id_disease))
                print("A short description of the disease is given below :\n")
                print(disease_details+"\n")
                print("The common medications and procedures suggested by other real doctors
are: \n")
                print(treatments+"\n")

        @Rule(Fact(action='find_disease'),
                  Fact(headache=MATCH.headache),
                  Fact(back_pain=MATCH.back_pain),
                  Fact(chest_pain=MATCH.chest_pain),
                  Fact(cough=MATCH.cough),
                  Fact(fainting=MATCH.fainting),
                  Fact(sore_throat=MATCH.sore_throat),
                  Fact(fatigue=MATCH.fatigue),
                  Fact(low_body_temp=MATCH.low_body_temp),
                  Fact(restlessness=MATCH.restlessness),
                  Fact(fever=MATCH.fever),
                  Fact(sunken_eyes=MATCH.sunken_eyes),
                  Fact(nausea=MATCH.nausea),

          Fact(blurred_vision=MATCH.blurred_vision),NOT(Fact(disease=MATCH.disease)),sa
lience = -999)
```

```
        def not_matched(self,headache, back_pain, chest_pain, cough, fainting, sore_throat,
fatigue, restlessness,low_body_temp ,fever ,sunken_eyes ,nausea ,blurred_vision):
                print("\nDid not find any disease that matches your exact symptoms")
                lis = [headache, back_pain, chest_pain, cough, fainting, sore_throat, fatigue,
restlessness,low_body_temp ,fever ,sunken_eyes ,nausea ,blurred_vision]
                max_count = 0
                max_disease = ""
                for key,val in symptom_map.items():
                        count = 0
                        temp_list = eval(key)
                        for j in range(0,len(lis)):
                                if(temp_list[j] == lis[j] and lis[j] == "yes"):
                                        count = count + 1
                        if count > max_count:
                                max_count = count
                                max_disease = val
                if_not_matched(max_disease)


if __name__ == "__main__":
        preprocess()
        engine = Greetings()
        while(1):
                engine.reset()  # Prepare the engine for the execution.
                engine.run()  # Run it!
                print("Would you like to diagnose some other symptoms?")
                if input() == "no":
                        exit()
                #print(engine.facts)
```

FAQS:

1. What is Expert System in AI?
2. What is Examples Of Expert System?
3. What are Components Of Expert System?
4. What is Advantages of Expert System ?
5. What is Applications of Expert System?

CONCLUSION:

In this assignment we have successfully implemented the Expert System of Hospitals and Medical facilities.