SRTCT’S

**SUMAN RAMESH TULSIANI TECHNICAL CAMPUS – FACULTY OF ENGINEERING,**

 **KHAMSHET**

**An ISO 9001:2015 Certified Institute**

**DEPARTMENT OF COMPUTER ENGINEERING**

### DEPARTMENT OF COMPUTER ENGINEERING

**Academic Year:**

**2024-25**

**Semesters -II**

LABORATORY MANUAL

## Class: TE

**Subject: LP -II (2019 Course) [ 310258 ]**

Subject In charge: Prof.Dr. Amruta Surana

Subject In charge: Prof. Suresh Reddy

SRTCT’S

**SUMAN RAMESH TULSIANI TECHNICAL CAMPUS – FACULTY OF ENGINEERING,**

 **KHAMSHET**

**An ISO 9001:2015 Certified Institute**

**DEPARTMENT OF COMPUTER ENGINEERING**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S.R.**  **No.** | **Name of the Experiment** | **Date of**  **Conduction** | **Date of**  **Checking** | **Page No.** | **Sign** | **Remarks** |  |
| 1 | **Artificial Intelligence**  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. |  |  |  |  |  |
| 2 | Implement A star Algorithm for any game search problem. |  |  |  |  |  |
| 3 | Implement Greedy search algorithm for any of the following application: Kruskal’s Minimum Spanning Tree Algorithm |  |  |  |  |  |
| 4 | Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem. |  |  |  |  |  |
| 5 | Develop an elementary chatbot for any suitable customer interaction application |  |  |  |  |  |
| 6 | Implement any one of the following Expert System  I. Information management |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 10. | Mini project |  |  |  |  |  |

SRTCT’S

**SUMAN RAMESH TULSIANI TECHNICAL CAMPUS – FACULTY OF ENGINEERING,**

 **KHAMSHET**

**An ISO 9001:2015 Certified Institute**

**DEPARTMENT OF COMPUTER ENGINEERING**

**DEPARTMENT OF COMPUTER ENGINEERING**

**Academic Year: 2024-25**

***CERTIFICATE***

This is to certify that Mr. /Miss. Of

Class T.E. Computer Roll No. has

satisfactory completed practical of the subject “Laboratory Practice II Lab” for 2nd semester

of Academic Year 2024 – 2025.

**Date:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Prof.S.V.Reddy** | **Prof.Dr.A.V.Surana** | **Prof.Dr.A.V.Surana** | **Prof. (Dr.) J.B.Sankpal** |
| **Subject Teacher** | **Subject Teacher** | **Head of Department** | **Principal** |

SRTCT’S

**SUMAN RAMESH TULSIANI TECHNICAL CAMPUS – FACULTY OF ENGINEERING,**

 **KHAMSHET**

**An ISO 9001:2015 Certified Institute**

**DEPARTMENT OF COMPUTER ENGINEERING**

## Practical 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.

**Depth First Search or DFS for a Graph**

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a Boolean visited array.

### Example:

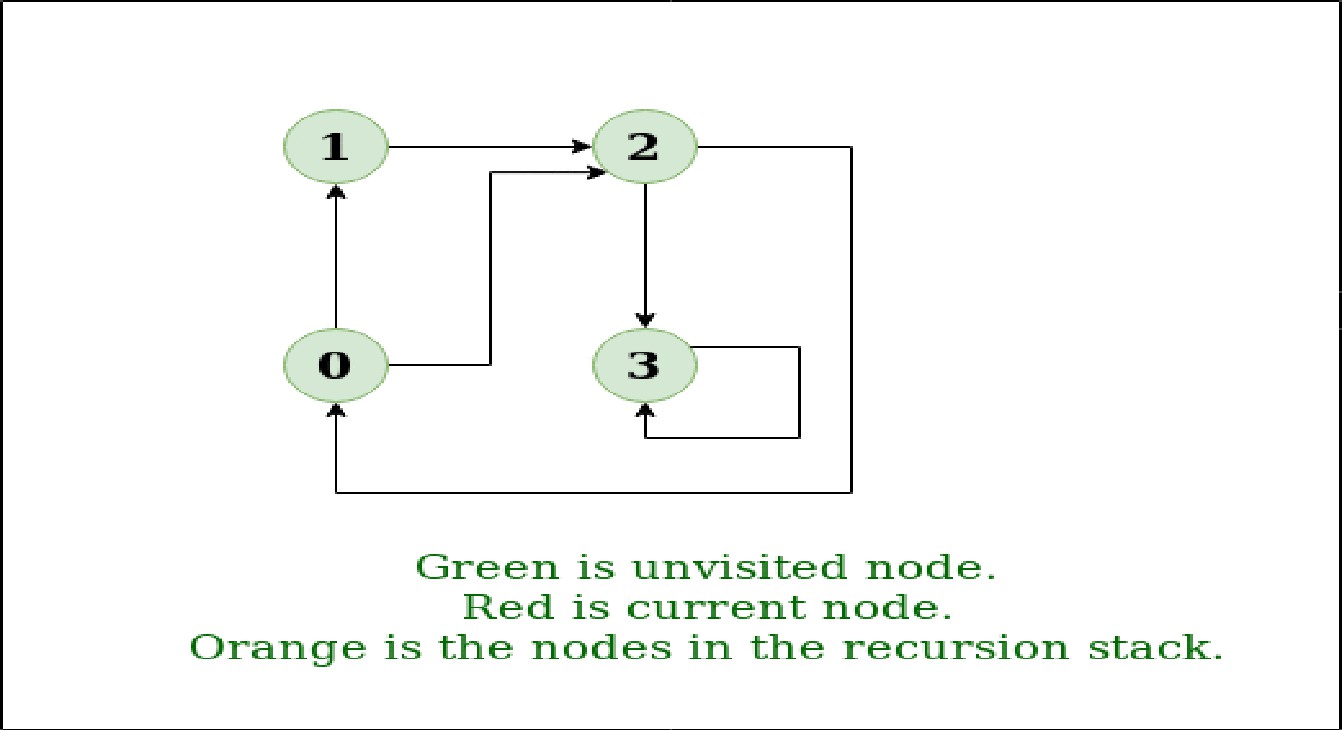
***Input:*** *n = 4, e = 6*

*0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3*

***Output:*** *DFS from vertex 1: 1 2 0 3*

#### Explanation:

*DFS Diagram:*

**

***Input:*** *n = 4, e = 6*

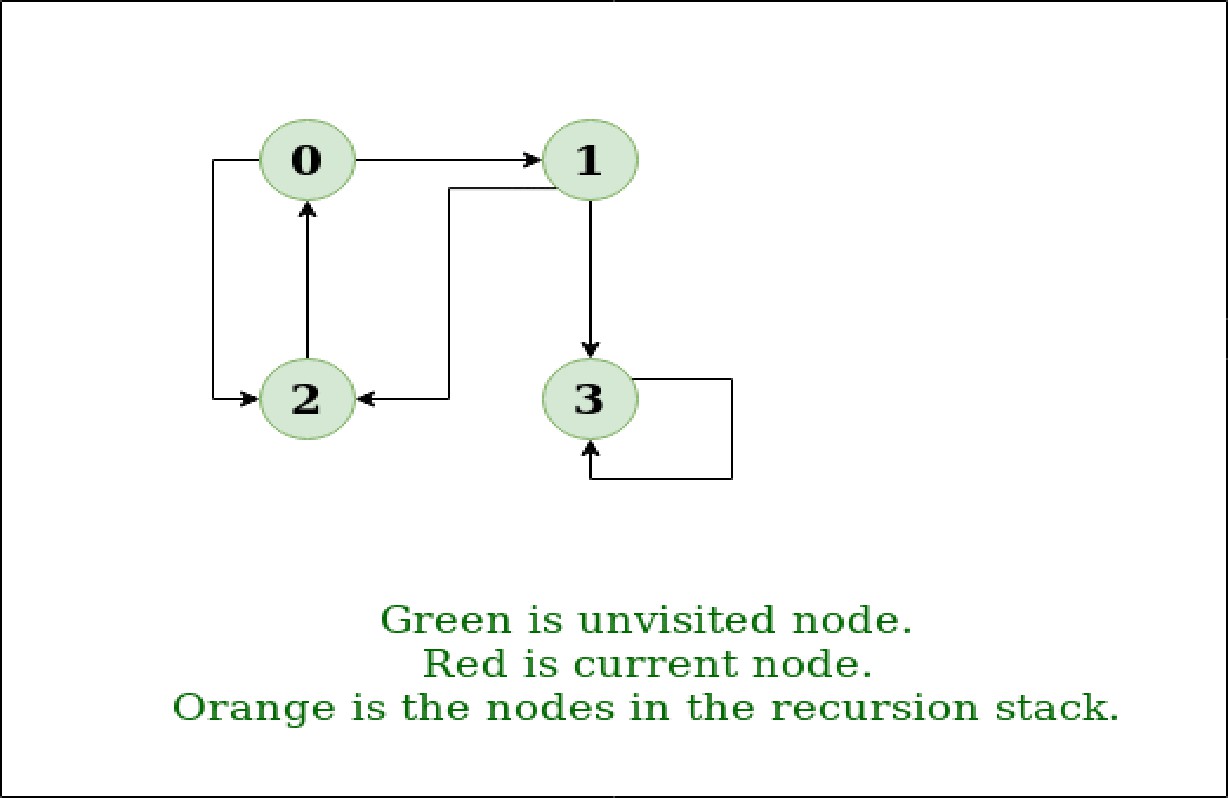
*2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3*

***Output:*** *DFS from vertex 2 : 2 0 1 3*

#### 

#### Explanation:

*DFS Diagram:*

**

### Approach:

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.

### Algorithm:

Create a recursive function that takes the index of the node and a visited array.

1. Mark the current node as visited and print the node.
2. Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

### Implementation:

Below are implementations of simple Depth First Traversal. The C++ implementation uses an

adjacency list representation of graphs. STL’s list container is used to store lists of adjacent nodes.

### // C++ program to print DFS traversal from

**// a given vertex in a given graph**

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

// Graph class represents a directed graph

// using adjacency list representation class Graph {

public:

map<int, bool> visited; map<int, list<int> > adj;

// function to add an edge to graph void addEdge(int v, int w);

// DFS traversal of the vertices

// reachable from v void DFS(int v);

};

void Graph::addEdge(int v, int w)

{

adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::DFS(int v)

{

// Mark the current node as visited and

// print it visited[v] = true; cout << v << " ";

// Recur for all the vertices adjacent

// to this vertex list<int>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (!visited[\*i]) DFS(\*i);

}

// Driver code int main()

{

// Create a graph given in the above diagram Graph g;

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

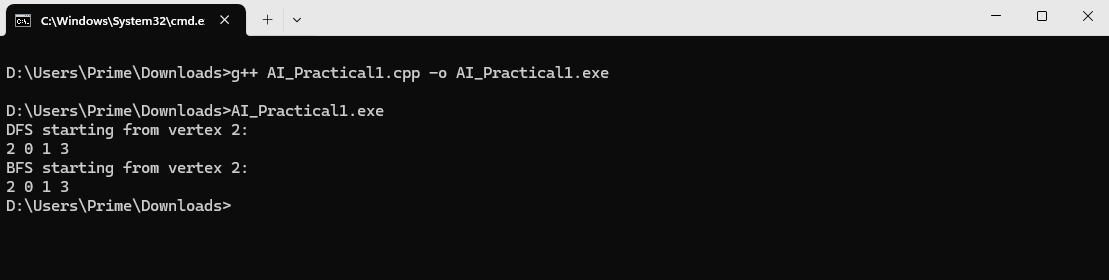
cout << "Following is Depth First Traversal" " (starting from vertex 2) \n";

g.DFS(2);

return 0;

}

### Output:



**Conclusion :**

SRTCT’S

**SUMAN RAMESH TULSIANI TECHNICAL CAMPUS – FACULTY OF ENGINEERING,**

 **KHAMSHET**

**An ISO 9001:2015 Certified Institute**

**DEPARTMENT OF COMPUTER ENGINEERING**

**Assignment No. 2**

# Title : Implement A star Algorithm for any game search problem.

# Theory :-

**What is A\* search algorithm?**

Intelligence is the strength of the human species; we have used it to improve our lives. Then, we created the concept of artificial intelligence to amplify human intelligence and to develop and flourish civilizations like never before. A\* Search Algorithm is one such algorithm that has been developed to help us. In this blog, we will learn more about what the A\* algorithm in artificial intelligence means, the steps involved in the A\* search algorithm in artificial intelligence, its implementation in Python, and more. A\* search algorithm is an algorithm that separates it from other traversal techniques. This makes A\* smart and pushes it much ahead of conventional algorithms. Let’s try to understand Basic AI Concepts and comprehend how does A\* algorithm work. Imagine a huge maze that is too big that it takes hours to reach the endpoint manually. Once you complete it on foot, you need to go for another one. This implies that you would end up investing a lot of time and effort to find the possible paths in this maze. Now, you want to make it less time-consuming. To make it easier, we will consider this maze as a search problem and will try to apply it to other possible mazes we might encounter in due course, provided they follow the same structure and rules. As the first step to converting this maze into a search problem, we need to define these six things.

1. A set of prospective states we might be in

2. A beginning and end state

3. A way to decide if we’ve reached the endpoint

4. A set of actions in case of possible direction/path changes

5. A function that advises us about the result of an action

6. A set of costs incurring in different states/paths of movement

To solve the problem, we need to map the intersections to the nodes (denoted by the red dots) and all the possible ways we can make movements towards the edges (denoted by the blue lines).A denotes the starting point, and B denotes the endpoint. We define the starting and endpoints at nodes A and B, respectively. If we use an uninformed search algorithm, it would be like finding a path that is blind, while an informed algorithm for a search problem would take the path that brings you closer to your destination. For instance, consider Rubik’s cube; it has many prospective states that you can be in, making the solution very difficult. This calls for the use of a guided search algorithm to find a solution. This explains the importance of A\*. Unlike other algorithms, A\* decides to take up a step only if it is convincingly sensible and reasonable as per its functions. This means it never considers any no optimal steps. This is why A\* is a popular choice for AI systems that replicate the real world – like video games and machine learning.

# A\* search algorithm steps:

### 

### Step 1: Add the beginning node to the open list

### Step 2: Repeat the following step

In the open list, find the square with the lowest F cost, which denotes the current square. Now we move to the closed square. Consider 8 squares adjacent to the current square and ignore it if it is on the closed list or if it is not workable. Do the following if it is workable. Check if it is on the open list; if not, add it. You need to make the current square as this square’s a parent. You will now record the different costs of the square, like the F, G, and H costs.If it is on the open list, use G cost to measure the better path. The lower the G cost, the better the path.If this path is better, make the current square as the parent square. Now you need to recalculate the other scores – the G and F scores of this square.

### – You’ll stop:

If you find the path, you need to check the closed list and add the target

square to it. There is no path if the open list is empty and you cannot find

the target square.

**Step 3.** Now you can save the path and work backward, starting from the target square, going to the

parent square from each square you go, till it takes you to the starting square. You’ve found your path

now.

# A\* Search Algorithm and Its Basic Concepts

A\* algorithm works based on heuristic methods, and this helps achieve optimality. A\* is a different form of the best-first algorithm. Optimality empowers an algorithm to find the best possible solution to a problem. Such algorithms also offer completeness; if there is any solution possible to an existing problem, the algorithm will definitely find it.

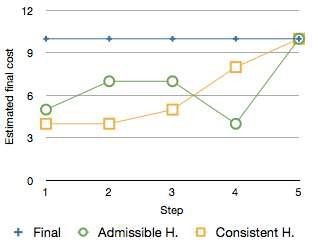
When A\* enters into a problem, firstly, it calculates the cost to travel to the neighboring nodes and chooses the node with the lowest cost. If The f(n) denotes the cost, A\* chooses the node with the lowest f(n) value. Here ‘n’ denotes the neighboring nodes. The calculation of the value can be done as shown below: f(n)=g(n)+h(n)f(n)=g(n)+h(n)

g(n) = shows the shortest path’s value from the starting node to node n h(n) = The heuristic approximation of the value of the node

The heuristic value has an important role in the efficiency of the A\* algorithm. To find the best solution, you might have to use different heuristic functions according to the type of the problem. However, the creation of these functions is a difficult task, and this is the basic problem we face in AI.

# What is a Heuristic Function?

A heuristic is simply called a heuristic function that helps rank the alternatives given in a search algorithm at each of its steps. It can either produce a result on its own or work in conjugation with a given algorithm to create a result. Essentially, a heuristic function helps algorithms to make the best decision faster and more efficiently. This ranking is based on the best available information and helps the algorithm decide the best possible branch to follow. Admissibility and consistency are the two fundamental properties of a heuristic function.



# Admissibility of the Heuristic Function:

A heuristic function is admissible if it can effectively estimate the real distance between a node ‘n’ and the end node. It never overestimates; if it ever does, it will be denoted by ‘d’, which also denotes the accuracy of the solution.

# Consistency of the Heuristic Function:

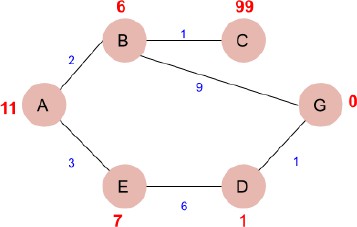
A heuristic function is consistent if the estimate of a given heuristic function turns out to be equal to or less than the distance between the goal (n) and a neighbor and the cost calculated to reach that neighbor.

A\* is indeed a very powerful algorithm used to increase the performance of artificial intelligence. It is one of the most popular search algorithms in AI. The sky is the limit when it comes to the potential of this algorithm.

However, the efficiency of an A\* algorithm highly depends on the quality of its heuristic function. Wonder why this algorithm is preferred and used in many software systems? There is no single facet of AI where the A\*algorithm has not found its application. From search optimization to games, robotics, and machine learning, the A\* algorithm is an inevitable part of a smart program

# Implementation with Python:

In this section, we are going to find out how the A\* search algorithm can be used to find the most cost-effective path in a graph. Consider the following graph below.



# Algorithm:

**Step 1:** Place the starting node into OPEN and find its f (n) value.

**Step 2:** Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return SUCCESS.

**Step 3:** Else remove the node from OPEN, find all its successors.

**Step 4:** Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.

**Step 5:** Go to Step-2.

**Step 6:** Exit.

# Advantages:

1. It is complete and optimal.
2. It is the best one from other techniques. It is used to solve very complex problems.
3. It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A\*.

# Disadvantages:

1. This algorithm is complete if the branching factor is finite and every action has fixed cost.
2. The speed execution of A\* search is highly dependant on the accuracy of the heuristic algorithm that is used to compute h (n).
3. It has complexity problems.

**Code:**

from queue import PriorityQueue

def a\_star(graph, start, goal, h):

open\_set = PriorityQueue()

open\_set.put((0, start))

came\_from = {}

g\_score = {node: float('inf') for node in graph}

g\_score[start] = 0

while not open\_set.empty():

\_, current = open\_set.get()

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

path.reverse()

return path

for neighbor, cost in graph[current]:

tentative\_g = g\_score[current] + cost

if tentative\_g < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g

f\_score = tentative\_g + h[neighbor]

open\_set.put((f\_score, neighbor))

return None

# example graph

graph = {

'A': [('B', 1), ('C', 3)],

'B': [('D', 3), ('E', 1)],

'C': [('F', 5)],

'D': [('G', 2)],

'E': [('G', 1)],

'F': [('G', 2)],

'G': []

}

# heuristic values

h = {

'A': 7, 'B': 6, 'C': 5, 'D': 4, 'E': 2,

'F': 3, 'G': 0

}

path = a\_star(graph, 'A', 'G', h)

print("Path found:", path)

**Output**:



**Conclusion :**

SRTCT’S

**SUMAN RAMESH TULSIANI TECHNICAL CAMPUS – FACULTY OF ENGINEERING,**

 **KHAMSHET**

**An ISO 9001:2015 Certified Institute**

**DEPARTMENT OF COMPUTER ENGINEERING**

## Assignment No. 3

**Implement Greedy search algorithm for the following application:**

**Kruskal’s Minimum Spanning Tree Algorithm**

#### What is Minimum Spanning Tree?

Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

#### How many edges does a minimum spanning tree has?

A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

### Below are the steps for finding MST using Kruskal’s algorithm

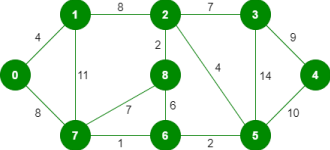
1. *Sort all the edges in non-decreasing order of their weight.*
2. *Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.*
3. *Repeat step#2 until there are (V-1) edges in the spanning tree.*

Step #2 uses the Union-Find algorithm to detect cycles. So we recommend reading the following post as a prerequisite.

Union-Find Algorithm | Set 1 (Detect Cycle in a Graph)

Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)

The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far. Let us understand it with an example: Consider the below input graph. weight edge that does not cause a cycle in the MST constructed so far. Let us understand it with an example: Consider the below input graph.



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

After sorting:

Weight Src Dest

1 7 6

2 8 2

2 6 5

4 0 1

4 2 5

6 8 6

7 2 3

7 7 8

8 0 7

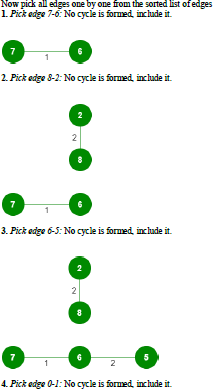
8 1 2

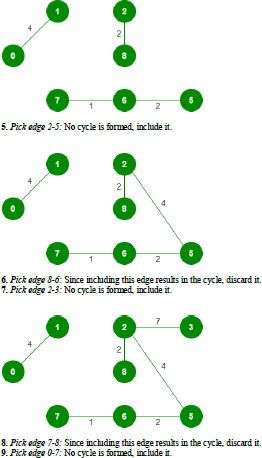
9 3 4

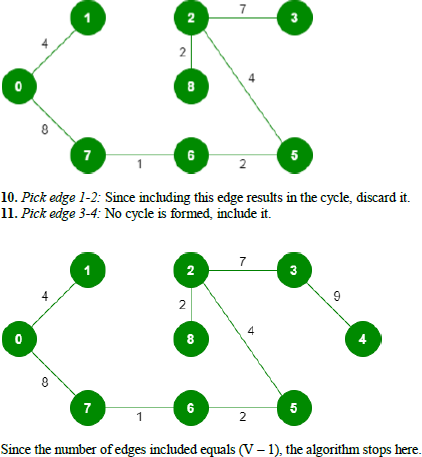
10 5 4

11 1 7

14 3 5







### Source Code

// Java program for Kruskal's algorithm to // find Minimum Spanning Tree of a given

//connected, undirected and weighted graph import java.util.\*;

import java.lang.\*; import java.io.\*;

class Graph {

// A class to represent a graph edge

class Edge implements Comparable<Edge>

{

int src, dest, weight;

// Comparator function used for

// sorting edgesbased on their weight public int compareTo(Edge compareEdge) {

return this.weight - compareEdge.weight; }

};

// A class to represent a subset for

// union-find class subset

{

int parent, rank;

};

int V, E; // V-> no. of vertices & E->no.of edges Edge edge[]; // collection of all edges

// Creates a graph with V vertices and E edges Graph(int v, int e)

{

V = v;

E = e;

edge = new Edge[E]; for (int i = 0; i < e; ++i) edge[i] = new Edge();

}

// A utility function to find set of an

// element i (uses path compression technique) int find(subset subsets[], int i)

{

// find root and make root as parent of i // (path compression)

if (subsets[i].parent != i) subsets[i].parent

= find(subsets, subsets[i].parent);

return subsets[i].parent;

}

// A function that does union of two sets

// of x and y (uses union by rank)

void Union(subset subsets[], int x, int y)

{

int xroot = find(subsets, x); int yroot = find(subsets, y);

// Attach smaller rank tree under root

// of high rank tree (Union by Rank) if (subsets[xroot].rank

< subsets[yroot].rank) subsets[xroot].parent = yroot; else if (subsets[xroot].rank

> subsets[yroot].rank) subsets[yroot].parent = xroot;

// If ranks are same, then make one as

// root and increment its rank by one else {

subsets[yroot].parent = xroot; subsets[xroot].rank++;

}

}

// The main function to construct MST using Kruskal's

// algorithm

void KruskalMST()

{

// Tnis will store the resultant MST Edge result[] = new Edge[V];

// An index variable, used for result[]

int e = 0;

// An index variable, used for sorted edges int i = 0;

for (i = 0; i < V; ++i) result[i] = new Edge();

// Step 1: Sort all the edges in non-decreasing // order of their weight. If we are not allowed to // change the given graph, we can create a copy of // array of edges Arrays.sort(edge);

// Allocate memory for creating V subsets subset subsets[] = new subset[V];

for (i = 0; i < V; ++i) subsets[i] = new subset();

// Create V subsets with single elements for (int v = 0; v < V; ++v)

{

subsets[v].parent = v; subsets[v].rank = 0;

}

i = 0; // Index used to pick next edge

// Number of edges to be taken is equal to V-1 while (e < V - 1)

{

// Step 2: Pick the smallest edge. And increment // the index for next iteration

Edge next\_edge = edge[i++];

int x = find(subsets, next\_edge.src); int y = find(subsets, next\_edge.dest);

// If including this edge does't cause cycle, // include it in result and increment the index // of result for next edge

if (x != y) {

result[e++] = next\_edge; Union(subsets, x, y);

}

// Else discard the next\_edge

}

// print the contents of result[] to display

// the built MST

System.out.println("Following are the edges in " + "the constructed MST"); int minimumCost = 0;

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

{

System.out.println(result[i].src + " -- " + result[i].dest

+ " == " + result[i].weight); minimumCost

+= result[i].weight;

}

System.out.println("Minimum Cost Spanning Tree " + minimumCost);

}

// Driver Code

public static void main(String[] args)

{

/\* Let us create following weighted graph 10

0 1

| \ |

6| 5\ |15

| \ |

2 3

4 \*/

int V = 4; // Number of vertices in graph int E = 5; // Number of edges in graph

Graph graph = new Graph(V, E);

// add edge 0-1 graph.edge[0].src = 0;

graph.edge[0].dest = 1;

graph.edge[0].weight = 10;

// add edge 0-2 graph.edge[1].src = 0;

graph.edge[1].dest = 2;

graph.edge[1].weight = 6;

// add edge 0-3 graph.edge[2].src = 0;

graph.edge[2].dest = 3;

graph.edge[2].weight = 5;

// add edge 1-3 graph.edge[3].src = 1;

graph.edge[3].dest = 3;

graph.edge[3].weight = 15;

// add edge 2-3 graph.edge[4].src = 2;

graph.edge[4].dest = 3;

graph.edge[4].weight = 4;

// Function call graph.KruskalMST();

}

}

// This code is contributed by Aakash Hasija

### Output:

**Conclusion** :

SRTCT’S

**SUMAN RAMESH TULSIANI TECHNICAL CAMPUS – FACULTY OF ENGINEERING,**

 **KHAMSHET**

**An ISO 9001:2015 Certified Institute**

**DEPARTMENT OF COMPUTER ENGINEERING**

## Assignment No. 4

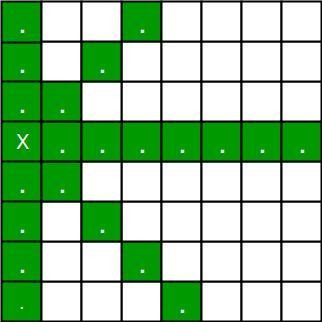
### Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

**N Queen Problem using Branch and Bound**

The **N queen’s 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.

Backtracking Algorithm for N-Queen is already discussed here. In backtracking solution we backtrack when we hit a dead end. ***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.”



1. 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.

1. 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.

1. 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:

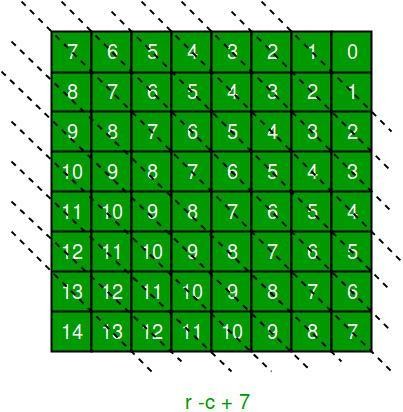
1. No two queens share a column.
2. No two queens share a row.
3. No two queens share a top-right to left-bottom diagonal.
4. No two queens share a top-left to bottom-right diagonal.

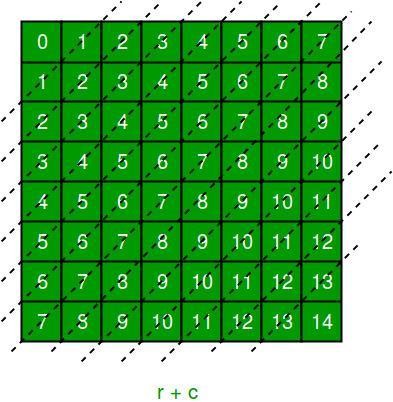
Number 1 is automatic because of the way we store the solution. For number 2, 3 and 4, we can perform updates in O(1) time. The idea is to keep **three Boolean arrays that tell us which rows and which diagonals are occupied**.

Lets do some pre-processing first. Let’s create two N x N matrix one for / diagonal and other one for \ diagonal. Let’s call them slashCode and backslashCode respectively. The trick is to fill them in such a way that two queens sharing a same /diagonal will have the same value in matrix slashCode, and if they share same \diagonal, they will have the same value in backslashCode matrix.

For an N x N matrix, fill slashCode and backslashCode matrix using below formula – slashCode[row][col] = row + col

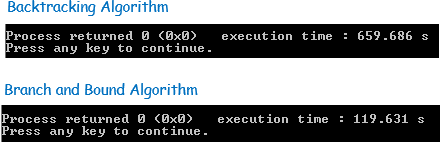
backslashCode[row][col] = row – col + (N-1) Using above formula will result in below matrices





The ‘N – 1’ in the backslash code is there to ensure that the codes are never negative because we will be using the codes as indices in an array.

Now before we place queen i on row j, we first check whether row j is used (use an array to store row info). Then we check whether slash code ( j + i ) or backslash code ( j – i + 7 ) are used (keep two arrays that will tell us which diagonals are occupied). If yes, then we have to try a different location for queen i. If not, then we mark the row and the two diagonals as used and recurse on queen i + 1. After the recursive call returns and before we try another position for queen i, we need to reset the row, slash code and backslash code as unused again, like in the code from the previous notes.



Below is the implementation of above idea –

""" Python3 program to solve N Queen Problem using Branch or Bound """

N = 8

""" A utility function to print solution """ def printSolution(board):

for i in range(N):

for j in range(N):

print(board[i][j], end = " ")

print()

""" A Optimized function to check if

a queen can be placed on board[row][col] """ def isSafe(row, col, slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup): if (slashCodeLookup[slashCode[row][col]] or

backslashCodeLookup[backslashCode[row][col]] or

rowLookup[row]):

return False return True

""" A recursive utility function to solve N Queen problem """

def solveNQueensUtil(board, col, slashCode, backslashCode,

rowLookup, slashCodeLookup, backslashCodeLookup):

""" base case: If all queens are placed then return True """ if(col >= N):

return True for i in range(N):

if(isSafe(i, col, slashCode, backslashCode,

rowLookup, slashCodeLookup, backslashCodeLookup)):

""" Place this queen in board[i][col] """ board[i][col] = 1

rowLookup[i] = True slashCodeLookup[slashCode[i][col]] = True backslashCodeLookup[backslashCode[i][col]] = True

""" recur to place rest of the queens """

if(solveNQueensUtil(board, col + 1,

return True

slashCode, backslashCode, rowLookup, slashCodeLookup, backslashCodeLookup)):

""" If placing queen in board[i][col]

doesn't lead to a solution,then backtrack """

""" Remove queen from board[i][col] """ board[i][col] = 0

rowLookup[i] = False slashCodeLookup[slashCode[i][col]] = False backslashCodeLookup[backslashCode[i][col]] = False

""" If queen can not be place in any row in this column col then return False """ return False

""" This function solves the N Queen problem using Branch or Bound. It mainly uses solveNQueensUtil()to solve the problem. It returns False if queens

cannot be placed,otherwise return True or prints placement of queens in the form of 1s. Please note that there may be more than one solutions,this function prints one of the feasible solutions."""

def solveNQueens():

board = [[0 for i in range(N)]

for j in range(N)]

# helper matrices

slashCode = [[0 for i in range(N)]

for j in range(N)] backslashCode = [[0 for i in range(N)]

for j in range(N)]

# arrays to tell us which rows are occupied rowLookup = [False] \* N

# keep two arrays to tell us

# which diagonals are occupied x = 2 \* N - 1

slashCodeLookup = [False] \* x backslashCodeLookup = [False] \* x

# initialize helper matrices for rr in range(N):

for cc in range(N):

slashCode[rr][cc] = rr + cc backslashCode[rr][cc] = rr - cc + 7

if(solveNQueensUtil(board, 0, slashCode, backslashCode,

rowLookup, slashCodeLookup, backslashCodeLookup) == False):

print("Solution does not exist") return False

# solution found printSolution(board) return True

# Driver Cde solveNQueens()

### Output :

**Performance:**

When run on local machines for N = 32, the backtracking solution took 659.68 seconds while above branch and bound solution took only 119.63 seconds. The difference will be even huge for larger values of N.

### Conclusion :

SRTCT’S

**SUMAN RAMESH TULSIANI TECHNICAL CAMPUS – FACULTY OF ENGINEERING,**

 **KHAMSHET**

**An ISO 9001:2015 Certified Institute**

**DEPARTMENT OF COMPUTER ENGINEERING**

## Assignment No. 5

### Develop an elementary chatbot for any suitable customer interaction application.

**Introduction**

Chatbot is a python based project. A chatbot is a computer program that interacts with human conversation through voice or text. It is built using python programming.

The chatbot project is using python version 3. It is an interesting project. Python is a general purpose high-level programming. It is useful for developing desktop GUI, websites, and web apps. This is a command-line based project. This project chatbot is a beginner or school project. It is coded in a simple way that every learner will understand.

When you execute the project you will see a set of questions in a command line. The chatbot will ask a set of questions and you have to answer theme. You must start the conversation answering the question asked by the boy. It is an entertainment-based project. In this project you must import time and random. The bot will mostly use random input. Time is useful for adding and waiting for the user’s response.

DOWNLOAD PROJECT:

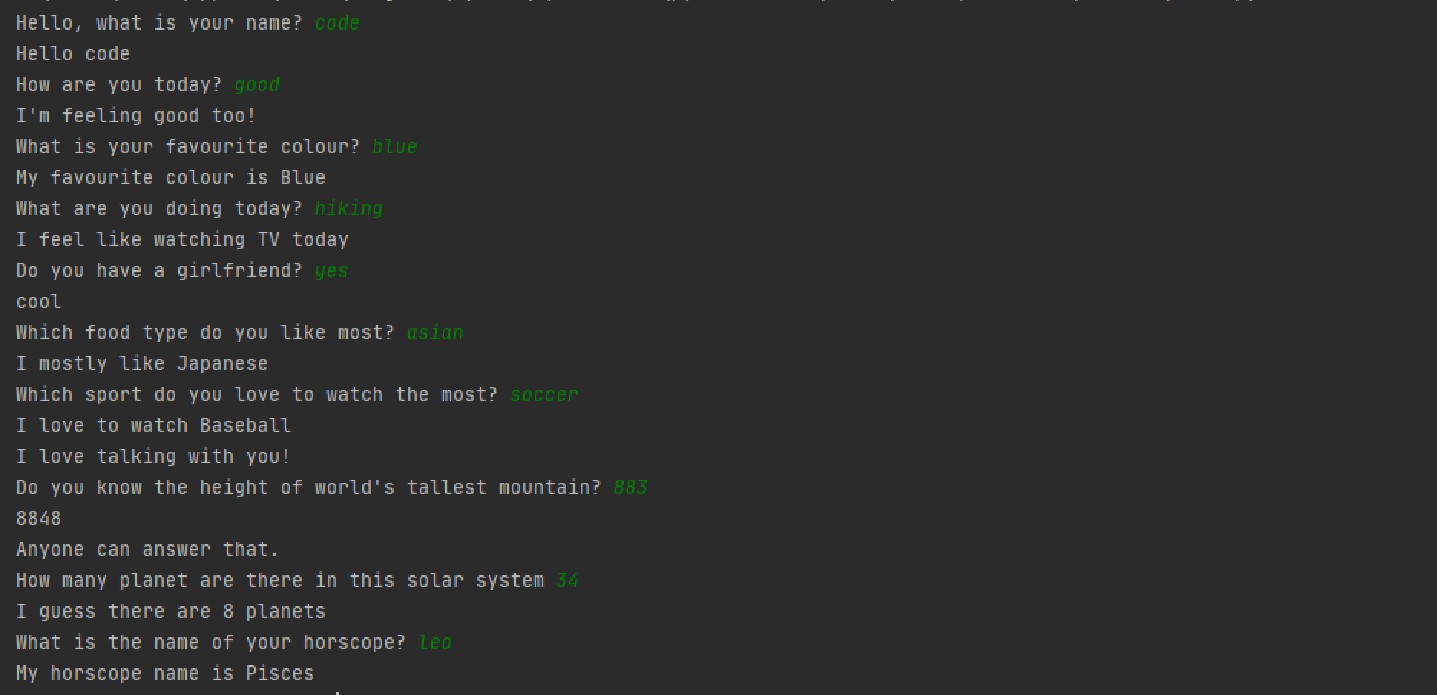
https://code-mentor.org/chatbot-in-python-with-source-code/

### PROCEDURE:

**How to use this project?**

* Install python version 3.
* Download the project.
* Extract the zip file and get the code.
* Set up an editor or IDE. (vs code, pycharm, anaconda)
* Open the python file in an editor.
* Execute the program.
* Start conversation with bot.
* Enjoy and Share!

Output



The image above is the screenshot of the output. This is how you must chat with bot.It is a simple beginner project. There is no necessary to import all the complex packages. Such as nltk, sklearn, tkinter and many others. This project will be fruitful for beginners to enhance their skills. You will learn new skills in this code. The project will be productive for you. Download the project and use the code by yourself.

Benefits of Bots –

1. Understandable information about the customer.
2. Can be called a selling partner by making and sending the products information.
3. Provides 24hrs services
4. Satisfy the need of clients as the customer will not go on waiting for your call. They need the action quickly or will turn to another brand.
5. Most of the customer prefers sending messages, text, SMS to the company for information. Marketing Bot can result or give your Business growth by making higher sales and satisfying the needs. Facebook Messenger is one of the widely used messengers in the U.S.
6. Recently chatbots were used by World Health Organization for providing information by ChatBot on Whatsapp.
7. Facebook Messenger, Slack, Whatsapp, and Telegram make use of ChatBot. 8. The modern need is there for Bot Building for growth of Business to make progress. 9. Another example of making use of ChatBo is Google Assistant and Siri. 10. Bots, for the most part, operate on a network. Bots that can communicate with one another will use internet-based services like IRC.

### Conclusion:

SRTCT’S

**SUMAN RAMESH TULSIANI TECHNICAL CAMPUS – FACULTY OF ENGINEERING,**

 **KHAMSHET**

**An ISO 9001:2015 Certified Institute**

**DEPARTMENT OF COMPUTER ENGINEERING**

## Assignment No. 6

**Implement any one of the following Expert System**

1. **Information management**

**What is Expert System in AI (Artificial Intelligence)? With Example What is Expert System?**

**Expert System** is an interactive and reliable computer-based decision-making system which uses both facts and heuristics to solve complex decision-making problems. It is considered at the highest level of human intelligence and expertise. The purpose of an expert system is to solve the most complex issues in a specific domain.

### Expert Systems in Artificial Intelligence

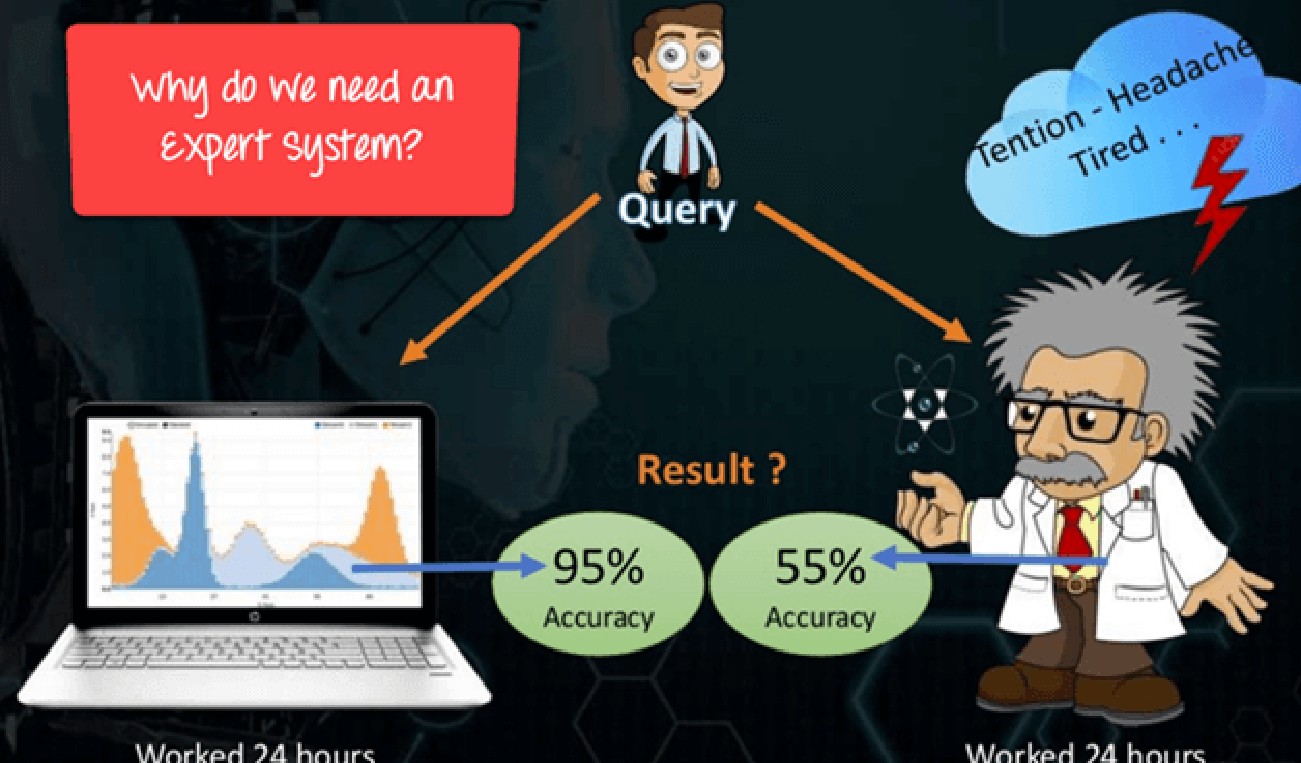
The Expert System in AI can resolve many issues which generally would require a human expert. It is based on knowledge acquired from an expert. Artificial Intelligence and Expert Systems are capable of expressing and reasoning about some domain of knowledge. Expert systems were the predecessor of the current day artificial intelligence, deep learning and machine learning systems. **Examples of Expert Systems**

Following are the Expert System Examples:

* + **MYCIN:** It was based on backward chaining and could identify various bacteria that could cause acute infections. It could also recommend drugs based on the patient’s weight. It is one of the best Expert System Example.
    - **DENDRAL:** Expert system used for chemical analysis to predict molecular structure. ∙

**PXDES:** An Example of Expert System used to predict the degree and type of lung cancer

* **CaDet:** One of the best Expert System Example that can identify cancer at early stages



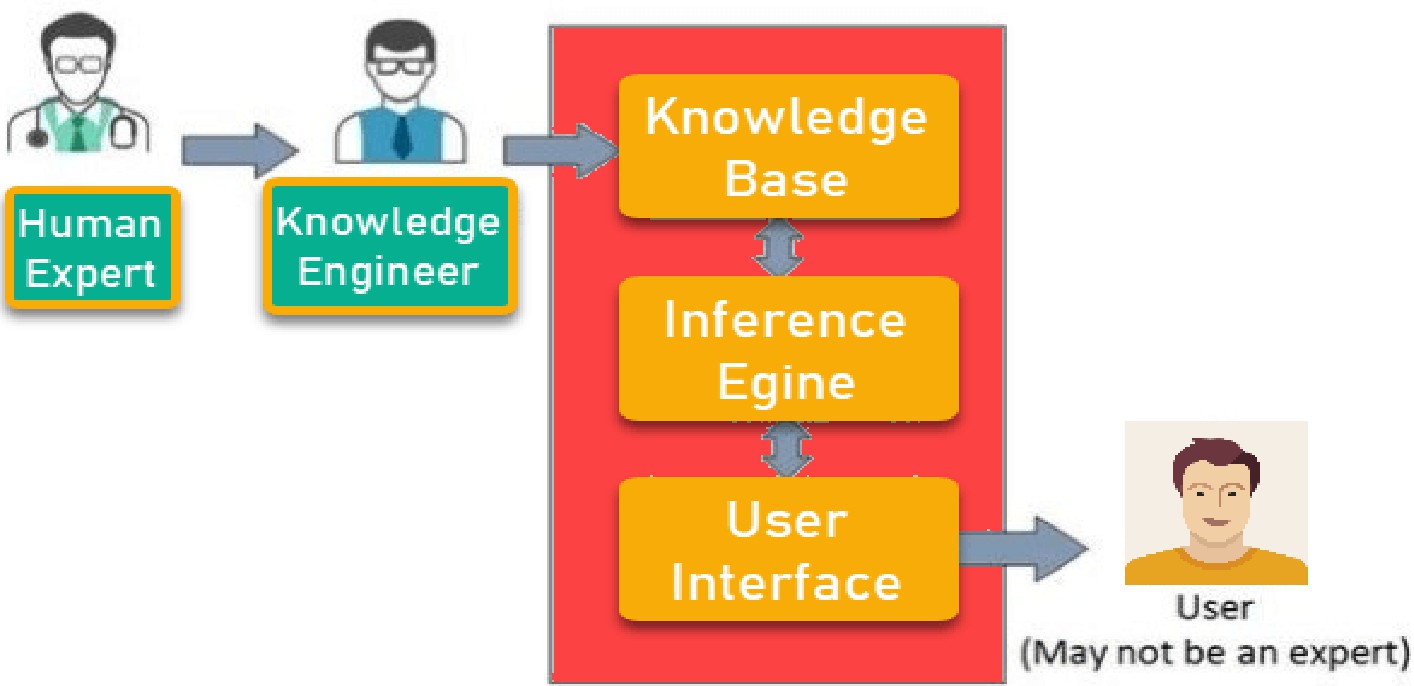
Why Expert Systems are required?

### Characteristics of Expert System

Following are the important Characteristics of Expert System in AI:

* + **The Highest Level of Expertise:** The Expert system in AI offers the highest level of expertise. It provides efficiency, accuracy and imaginative problem-solving.
  + **Right on Time Reaction:** An Expert System in Artificial Intelligence interacts in a very reasonable period of time with the user. The total time must be less than the time taken by an expert to get the most accurate solution for the same problem.
  + **Good Reliability:** The Expert system in AI needs to be reliable, and it must not make any a mistake.
  + **Flexible:** It is vital that it remains flexible as it the is possessed by an Expert system.
  + **Effective Mechanism:** Expert System in Artificial Intelligence must have an efficient mechanism to administer the compilation of the existing knowledge in it.
  + **Capable of handling challenging decision & problems:** An expert system is capable of handling challenging decision problems and delivering solutions.

### Components of Expert System

**SRTTC FOE LABORATORY PRACTICE II** 50

The Expert System in AI consists of the following given components:

### User Interface

The user interface is the most crucial part of the Expert System Software. This component takes the user’s query in a readable form and passes it to the inference engine. After that, it displays the results to the user. In other words, it’s an interface that helps the user communicate with the expert system.

### Inference Engine

The inference engine is the brain of the expert system. Inference engine contains rules to solve a specific problem. It refers the knowledge from the Knowledge Base. It selects facts and rules to apply when trying to answer the user’s query. It provides reasoning about the information in the knowledge base. It also helps in deducting the problem to find the solution. This component is also helpful for formulating conclusions.

### Knowledge Base

The knowledge base is a repository of facts. It stores all the knowledge about the problem domain. It is like a large container of knowledge which is obtained from different experts of a specific field.

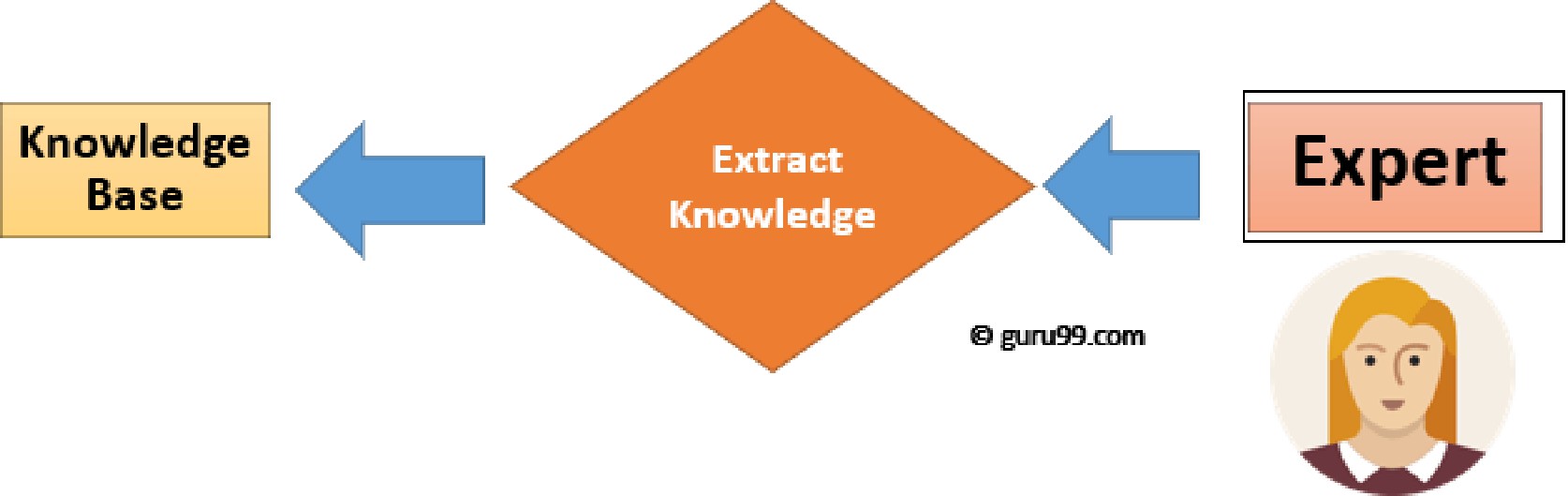
Thus we can say that the success of the Expert System Software mainly depends on the highly accurate and precise knowledge.

### Other Key terms used in Expert Systems Facts and Rules

A fact is a small portion of important information. Facts on their own are of very limited use. The rules are essential to select and apply facts to a user problem.

### Knowledge Acquisition

The term knowledge acquisition means how to get required domain knowledge by the expert system. The entire process starts by extracting knowledge from a human expert, converting the acquired knowledge into rules and injecting the developed rules into the knowledge base.



Knowledge Extraction Process

### Participant in Expert Systems Development

**Participant Role**

|  |  |
| --- | --- |
| Domain Expert | He is a person or group whose expertise and knowledge is taken to develop an expert system |
| Knowledge Engineer | Knowledge engineer is a technical person who integrates knowledge into computer systems |
| **End User** | It is a person or group of people who are using the expert system to get advice which will not be provided by the expert |

### The process of Building An Expert Systems

|  |  |
| --- | --- |
| **Conventional System** | **Expert System** |
| Knowledge and processing are combined in one unit. | Knowledge database and the  processing mechanism are two separate components. |

|  |  |
| --- | --- |
| **Human Expert** | **Artificial Expertise** |
| Perishable | Permanent |
| Difficult to Transfer | Transferable |
| Difficult to Document | Easy to Document |
| Unpredictable | Consistent |
| Expensive | Cost effective System |

Determining the characteristics of the problem

* Knowledge engineer and domain expert work in coherence to define the problem

∙ The knowledge engineer translates the knowledge into a computer-understandable language. He designs an inference engine, a reasoning structure, which can use knowledge when needed.

* Knowledge Expert also determines how to integrate the use of uncertain knowledge in the reasoning process and what type of explanation would be useful.

### Conventional System vs. Expert System

|  |  |
| --- | --- |
| The programme does not make errors (Unless  error in programming). | The Expert System may make a mistake. |
| The system is operational only when fully developed. | The expert system is optimized on an ongoing  basis and can be launched with a small number of rules. |
| Step by step execution according to fixed  algorithms is required. | Execution is done logically & heuristically. |
| It needs full information. | It can be functional with sufficient or insufficient  information. |

### 

**Human expert vs. Expert System**

**Advantages of Expert System**

Below are the main advantages/benefits of Expert Systems in Artificial Intelligence (AI):

* It improves the decision quality
* Cuts the expense of consulting experts for problem-solving
* It provides fast and efficient solutions to problems in a narrow area of specialization. ∙

It can gather scarce expertise and used it efficiently.

* Offers consistent answer for the repetitive problem
* Maintains a significant level of information
* Helps you to get fast and accurate answers
* A proper explanation of decision making
* Ability to solve complex and challenging issues
* Artificial Intelligence Expert Systems can steadily work without getting emotional, tensed or fatigued.

### Limitations of Expert System

Below are the disadvantages/limitations of Expert System in AI:

* Unable to make a creative response in an extraordinary situation
* Errors in the knowledge base can lead to wrong decision
* The maintenance cost of an expert system is too expensive
* Each problem is different therefore the solution from a human expert can also be different and more creative

### Applications of Expert Systems

Some popular Application of Expert System:

* Information management
* Hospitals and medical facilities
* Help desks management
* Employee performance evaluation
* Loan analysis
* Virus detection
* Useful for repair and maintenance projects
* Warehouse optimization
* Planning and scheduling
* The configuration of manufactured objects
* Financial decision-making Knowledge publishing
* Process monitoring and control
* Supervise the operation of the plant and controller
* Stock market trading
* Airline scheduling & cargo schedules

### Summary

* An Expert System is an interactive and reliable computer-based decision-making system which uses both facts and heuristics to solve complex decision-making problem ∙ Key components of an Expert System are

1. User Interface, 2) Inference Engine, 3) Knowledge Base
   * Key participants in Artificial Intelligence Expert Systems Development are
2. Domain Expert 2) Knowledge Engineer 3) End User
   * Improved decision quality, reduce cost, consistency, reliability, speed are key benefits of an Expert System
   * An Expert system can not give creative solutions and can be costly to maintain. ∙ An Expert System can be used for broad applications like Stock Market, Warehouse, HR, etc

If you want to learn about Artificial Intelligence, here’s a free tutorial you’ll want to check out: AI Tutorial

### Student Information Management System In C++ With Source Code Project: Student Information System

Student Information System is based on the concept to generate the Student’s Information records and to add & update it. Here User can add their Student’s detail with their courses safely and it’s not time-consuming. This System makes easy to store records of each and every employee. The whole project is designed in ‘C++’ language and different variables and strings have been used for the development of this project. This mini project is easy to operate and understand by the users.

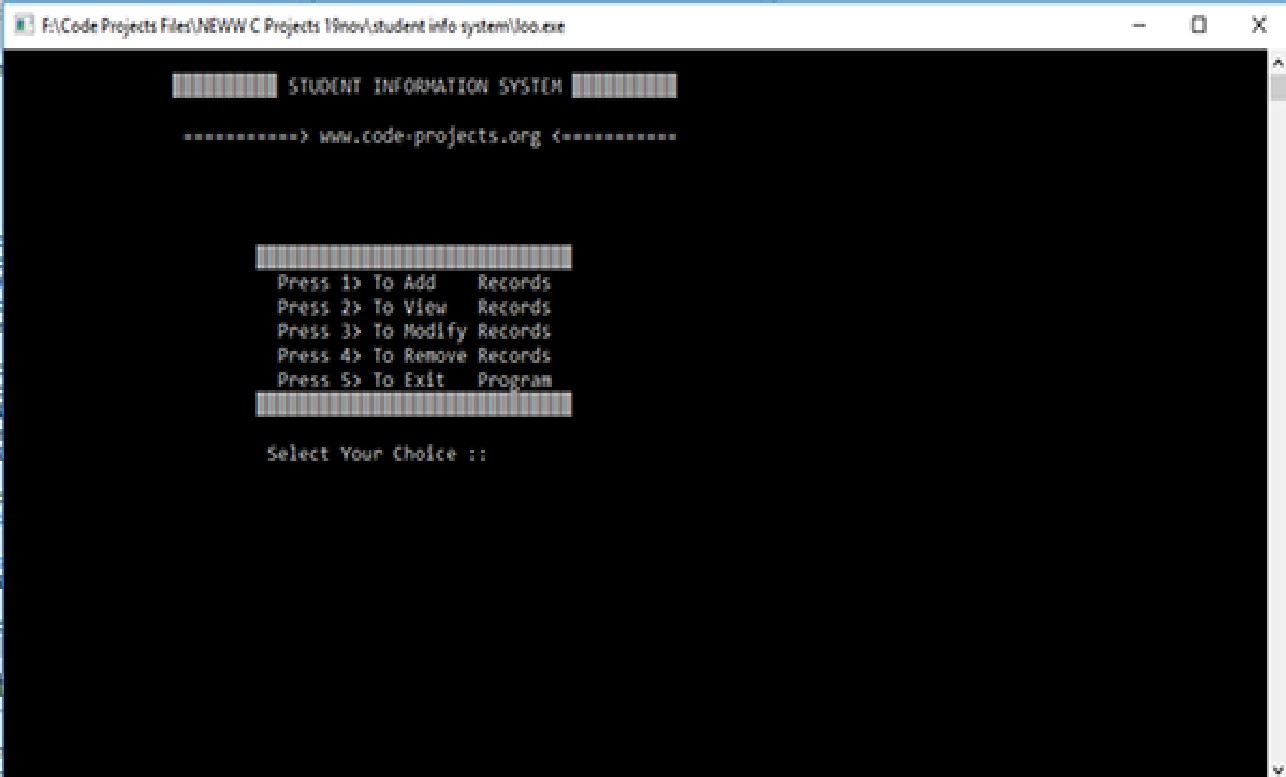
### Features:

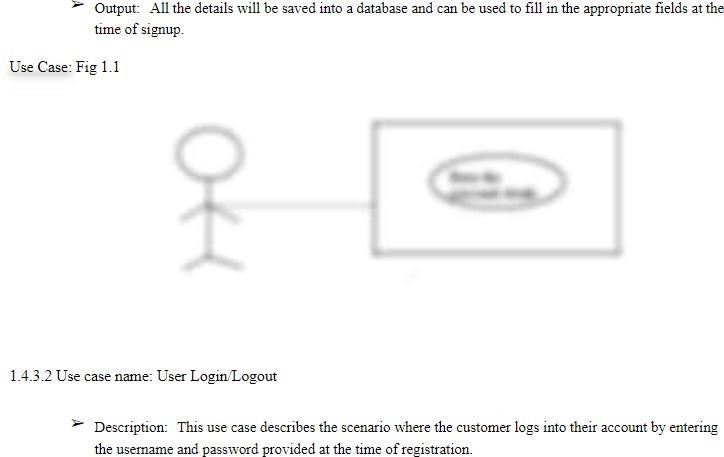
1. Proper Log-In System.
2. Easy To Add, Modify, List And Delete Student’s Information Records.
3. Password Protected.

### DOWNLOAD STUDENT INFORMATION SYSTEM WITH SOURCE CODE : CLICK THE BUTTON BELOW

DOWNLOAD PROJECT:

<https://download.code-projects.org/details/a98005e8-1910-4dd5-b893-f925404d99e8>

 OUTPUT:



### 

### Conclusion: