



Dr. D. Y. Patil Pratishthan's

DR. D. Y. PATIL INSTITUTE OF ENGINEERING, MANAGEMENT & RESEARCH

Approved by A.I.C.T.E, New Delhi , Maharashtra State Government, Affiliated to Savitribai Phule Pune University
Sector No. 29, PCNTDA , Nigidi Pradhikaran, Akurdi, Pune 411044. Phone: 020-27654470, Fax: 020-27656566
Website : www.dypiemr.ac.in Email : principal.dypiemr@gmail.com

Department of Computer Engineering

LAB MANUAL Laboratory Practice II (TE 2019 pattern) Semester II

**Prepared by:
Mrs. Ketaki Bhoyar
Mrs. Swati Rajput
Mr. Prateek Meshram**



Laboratory Practice II

Course Code	Course Name	Teaching Scheme (Hrs./ Week)	Credits
310258	Laboratory Practice II	4	2

Course Objectives:

- To learn and apply various search strategies for AI
- To Formalize and implement constraints in search problems
- To understand the concepts of Information Security / Augmented and Virtual Reality/Cloud Computing/Software Modeling and Architectures

Course Outcomes:

On completion of the course, learner will be able to–

- CO1: Design system using different informed search / uninformed search or heuristic approaches
- CO2: Apply basic principles of AI in solutions that require problem solving, inference, perception, knowledge representation, and learning
- CO3: Design and develop an expert system
- CO4: Use tools and techniques in the area of Cloud Computing OR Use tools and techniques in the area Software Modeling and Architectures
- CO5: Use the knowledge of Cloud Computing for problem solving OR Use the knowledge of Software Modeling and Architectures for problem solving
- CO6: Apply the concepts Cloud Computing to design and develop applications OR Apply the concepts Software Modeling and Architectures to design and develop applications

Operating System recommended: 64-bit Windows OS and Linux

Programming tools recommended:

Cloud Computing: -

Software Modeling and Architectures: Front end:HTML5, Bootstrap, jQuery, JS etc. Backend: MySQL/MongoDB/NodeJS

Table of Contents

Sr. No	Title of Experiment	CO Mapping	Page No
Part I : Artificial Intelligence			
Group A			
All assignments are compulsory			
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.	CO1	
2	Implement A star Algorithm for any game search problem.	CO1	
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 logical operators etc.).	CO1, CO2	
Group B			
4	Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.	CO2	
5	Develop an elementary chatbot for any suitable customer interaction application.	CO2, CO3	
Group C			
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	CO2, CO3	
Part II			
Elective II – Cloud Computing			
(All assignments are compulsory)			
1	Case study on Microsoft azure to learn about Microsoft Azure is a cloud computing platform and infrastructure, created by Microsoft, for building, deploying and managing applications and services through a global network of Microsoft-managed data centers. OR Case study on Amazon EC2 and learn about Amazon EC2 web services develop a recursive algorithm for searching all the vertices of a graph or tree data structure.	CO4	
2	Installation and configure Google App Engine. OR	CO5	

3	Creating an Application in Salesforce.com using Apex programming Language.	CO5, CO6	
4	Design and develop custom Application (Mini Project) using Salesforce Cloud.	CO6	
5	<p style="text-align: center;">Mini-Project</p> <p>Setup your own cloud for Software as a Service (SaaS) over the existing LAN in your laboratory. In this assignment you have to write your own code for cloud controller using open-source technologies to implement with HDFS. Implement the basic operations may be like to divide the file in segments/blocks and upload/ download file on/from cloud in encrypted form.</p>	CO6	
Elective II – Software Modelling and Design (Problem statement 1, 2 , 3 or 4, Problem statement 5 and 6 are mandatory)			
1	Consider a library, where a member can perform two operations: issue book and return it. A book is issued to a member only after verifying his credentials. Develop a use case diagram for the given library system by identifying the actors and use cases and associate the use cases with the actors by drawing a use case diagram. Use UML tool.	CO4	
2	Consider online shopping system. Perform the following tasks and draw the class diagram using UML tool. Represent the individual classes, and objects Add methods Represent relationships and other classifiers like interfaces.	CO4,CO5	
3	Consider the online shopping system in the assignment 2. Draw the sequence diagram using UML tool to show message exchanges.	CO5	
4	<p>Consider your neighboring travel agent from whom you can purchase flight tickets. To book a ticket you need to provide details about your journey i.e., on which date and at what time you would like to travel. You also need to provide your address. The agency has recently been modernized. So, you can pay either by cash or by card. You can also cancel a booked ticket later if you decide to change your plan. In that case you need to book a new ticket again. Your agent also allows you to book a hotel along with flight ticket. While cancelling a flight ticket you can also cancel hotel booking. Appropriate refund as per policy is made in case of cancellation.</p> <p>Perform the following tasks and draw the use case diagram using UML tool.</p> <p>a. Identify the use cases from a given non-trivial problem statement.</p> <p>b. Identify the primary and secondary actors for a system.</p> <p>c. Use to generalization of use cases and «include» stereotypes to prevent redundancy in the coding phase</p>	CO4,CO5	
5	<p style="text-align: center;">Mini-Project</p> <p>Select a moderately complex system and narrate concise requirement Specification for the same. Design the system indicating system elements organizations using applicable architectural styles and design patterns with the help of a detailed Class diagram depicting logical architecture. Specify and document the architecture and design pattern with the help of templates. Implement the system features and judge the benefits of the design patterns accommodated.</p>	CO6	

Part I

Artificial Intelligence

Group A

Lab Assignment No.	A1
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.
Roll No.	
Class	TE
Date of Completion	
Subject	Laboratory Practice II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: A1

Title: Implement depth first search algorithm and Breadth First Search algorithm.

Problem Statement: 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:

- Understand and implement BFS/DFS algorithm.
- Analyze time complexity of the search algorithm.

Outcomes:

- Ability to choose an appropriate problem-solving method and knowledge representation technique.

Software Required:

- Python

Theory:

- Depth First Search:

Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph.

➤ Depth First Search Algorithm:

A standard DFS implementation puts each vertex of the graph into one of two categories:

- Visited
- Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

➤ Depth First Search Example:

Let's see how the Depth First Search algorithm works with an example. We use an undirected graph with 5 vertices.

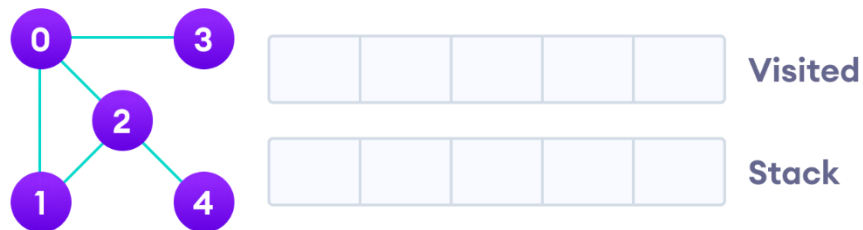


Figure 1: Undirected graph with 5 vertices

We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.

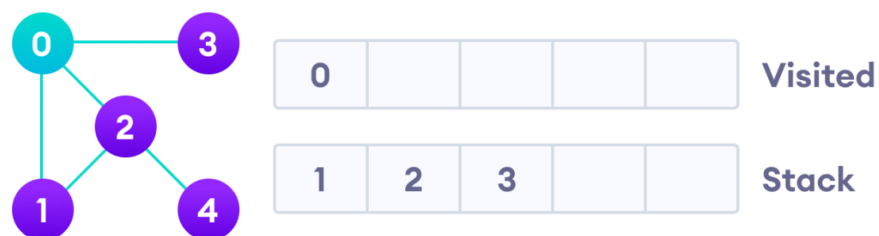


Figure 2: Visit the element and put it in the visited list

Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

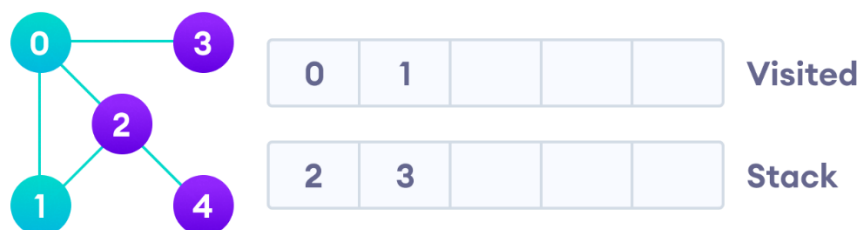


Figure 3: Visit the element at the top of stack

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

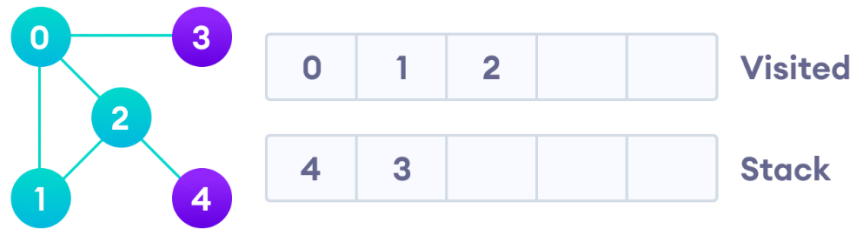


Figure 4: Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

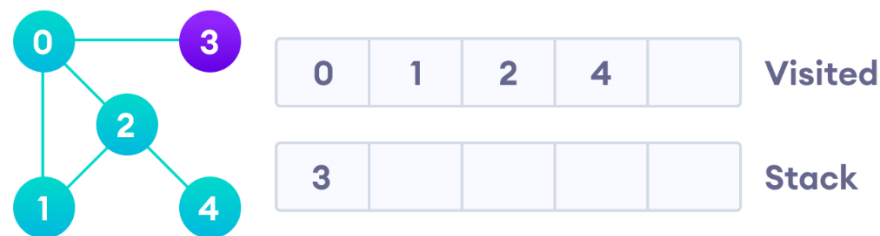


Figure 5: Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.

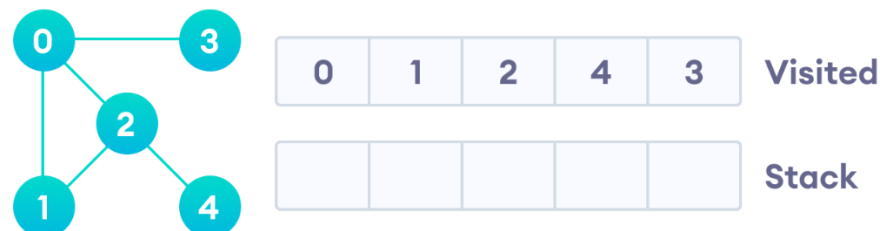


Figure 6: After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.

➤ DFS Pseudocode (recursive implementation):

The pseudocode for DFS is shown below. In the `init()` function, notice that we run the DFS function on every node. This is because the graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the DFS algorithm on every node.

DFS(G, u)

```
u.visited = true
for each v ∈ G.Adj[u]
    if v.visited == false
        DFS(G,v)

init() {
    For each u ∈ G
        u.visited = false
    For each u ∈ G
        DFS(G, u)
}
```

- Complexity of Depth First Search:
 - Time complexity of the DFS algorithm: $O(V + E)$, where V is the number of nodes and E is the number of edges.
 - Space complexity of the algorithm: $O(V)$.
- Application of DFS Algorithm:
 - For finding the path
 - To test if the graph is bipartite
 - For finding the strongly connected components of a graph
 - For detecting cycles in a graph

- Breadth First Search:

Traversal means visiting all the nodes of a graph. Breadth First Traversal or Breadth First Search is a recursive algorithm for searching all the vertices of a graph or tree data structure.

- Breadth First Search Algorithm:

A standard BFS implementation puts each vertex of the graph into one of two categories:

 - Visited
 - Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The BFS algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

➤ Breadth First Search Example:

Let's see how the Breadth First Search algorithm works with an example. We use an undirected graph with 5 vertices.

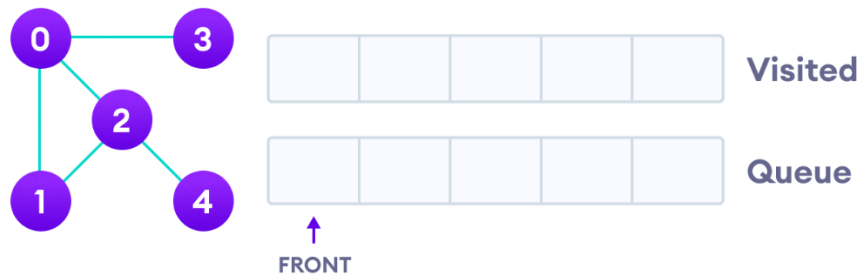


Figure 1: Undirected graph with 5 vertices

We start from vertex 0, the BFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the queue.

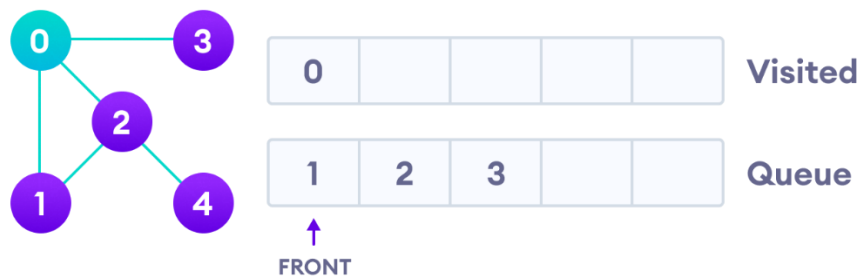


Figure 2: Visit start vertex and add its adjacent vertices to queue

Next, we visit the element at the front of queue i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

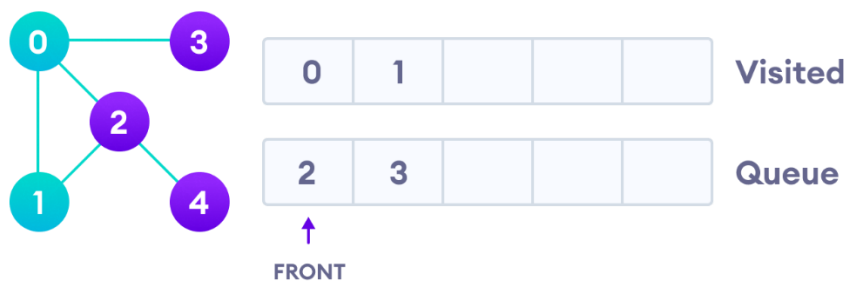


Figure 3: Visit the first neighbour of start node 0, which is 1

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the back of the queue and visit 3, which is at the front of the queue.

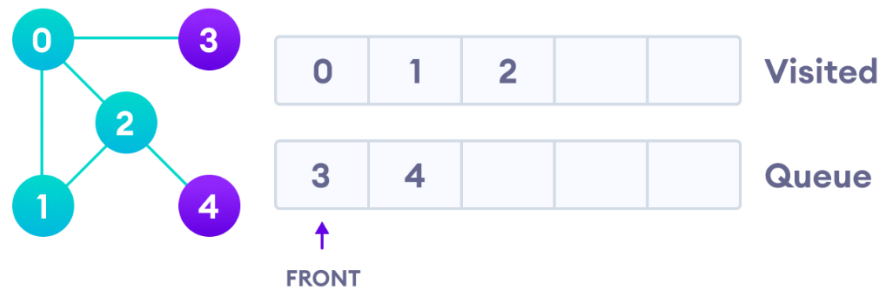


Figure 4: Visit 2 which was added to queue earlier to add its neighbors

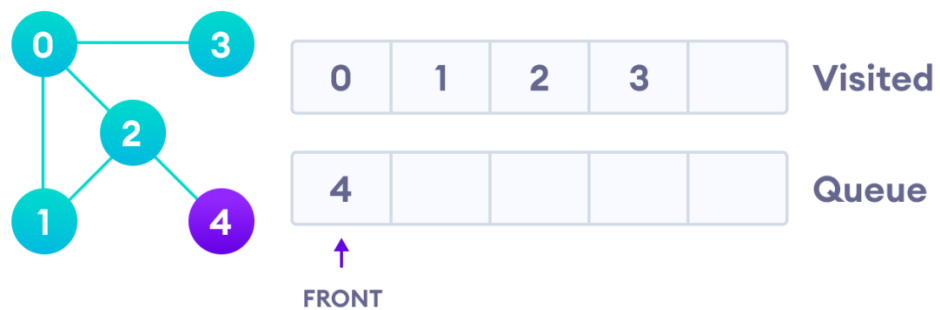


Figure 5: 4 remains in the queue

Only 4 remains in the queue since the only adjacent node of 3 i.e. 0 is already visited. We visit it.

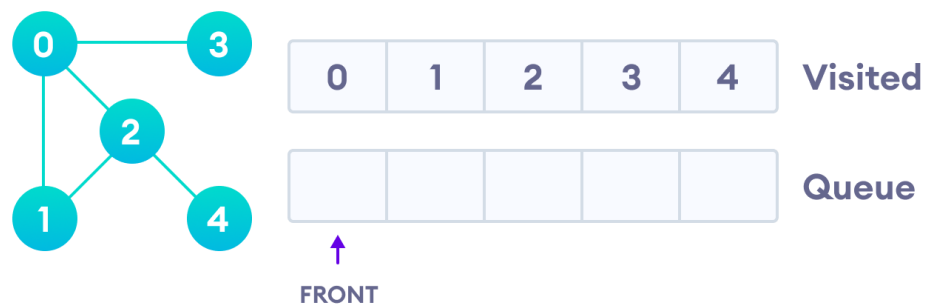


Figure 6: Visit last remaining item in the queue to check if it has unvisited neighbors

Since the queue is empty, we have completed the Breadth First Traversal of the graph.

➤ BFS Pseudocode:

```
create a queue Q
mark v as visited and put v into Q
while Q is non-empty
    remove the head u of Q
    mark and enqueue all (unvisited) neighbours of u
```

➤ Complexity of Breadth First Search:

- Time complexity of the BFS algorithm: $O(V + E)$,
where V is the number of nodes and E is the number of edges.
- Space complexity of the BFS algorithm: $O(V)$.

➤ Application of DFS Algorithm:

- To build index by search index
- For GPS navigation
- Path finding algorithms
- In Ford-Fulkerson algorithm to find maximum flow in a network
- Cycle detection in an undirected graph
- In minimum spanning tree

Conclusion: Implemented depth first search algorithm and Breadth First Search algorithm for an undirected graph using recursive algorithm for searching all the vertices of a graph or tree data structure.

Lab Assignment No.	A2
Title	Implement A star (A*) Algorithm for any game search problem.
Roll No.	
Class	TE
Date of Completion	
Subject	Laboratory Practice II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: A2

Title: Implement A star (A*) Algorithm for any game search problem.

Problem Statement: Implement A star (A*) Algorithm for any game search problem.

Objective:

- Understand the working of informed search algorithm
- Implement A* search algorithm

Outcome:

- Ability to choose an appropriate problem solving method and knowledge representation technique

Software Required:

- Python

Theory:

An informed search strategy-one that uses problem-specific knowledge-can find solutions more efficiently. A key component of these algorithms is a heuristic function $h(n)$

$h(n)$ = estimated cost of the cheapest path from node n to a goal node.

Admissible /heuristic never over estimated i.e. $h(n) \leq$ Actual cost. For example, Distance between two nodes(cities) \Rightarrow straight line distance and for 8-puzzel problem- Admissible heuristic can be number of misplaced tiles $h(n)= 8$.

- **A* Search technique:**

It is informed search technique. It uses additional information beyond problem formulation and tree. Search is based on Evaluation function $f(n)$. Evaluation function is based on both heuristic function $h(n)$ and $g(n)$.

$$f(n)=g(n) + h(n)$$

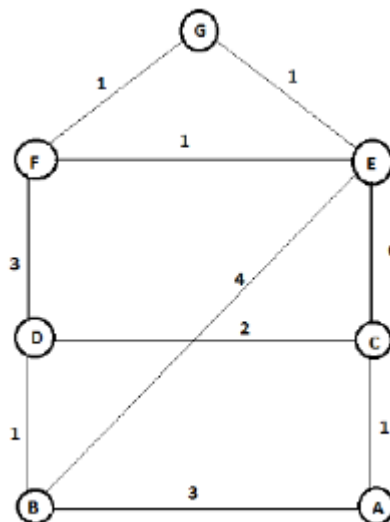
It uses two queues for its implementation: open, close Queue. Open queue is a priority queue which is arranged in ascending order of $f(n)$

- **Algorithm:**

1. Create a single member queue comprising of Root node
2. If FIRST member of queue is goal then goto step 5
3. If first member of queue is not goal then remove it from queue and add to close queue.
4. Consider its children if any, and add them to queue in ascending order of evaluation function $f(n)$.
5. If queue is not empty then goto step 2.

6. If queue is empty then goto step 6
 7. Print 'success' and stop
 8. Print 'failure' and stop.
- Performance Comparison:
 - Completeness: yes
 - Optimality: yes
 - Limitation:
 - • It generate same node again and again
 - • Large Memory is required
 - Observation:

Although A* generate many nodes it never uses those nodes for which $f(n) > c^*$ where c^* is optimum cost.
 - Consider an example as below



OPEN/FRINGE

[A]

[C,B]

[D,B,E,A]

[F,E,B,C,A]

[G,E,B,C,A,D]

CLOSE

[]

[A]

[A,C]

[A,C,D]

[A,C,D,F]

SUCCESS

Node A:

$$f(B) = g(B) + h(B) = 3 + 5 = 8$$

$$f(C) = g(C) + h(C) = 1 + 6 = 7$$

Node C:

$$f(A) = g(A) + h(A) = 2 + 7 = 10$$

$$f(D) = g(D) + h(D) = 3 + 4 = 7$$

$$f(E) = g(E) + h(E) = 7 + 1 = 8$$

Node D:

$$f(F) = g(F) + h(F) = 6 + 1 = 7$$

$$f(C) = g(C) + h(C) = 5 + 6 = 11$$

$$f(B) = g(B) + h(B) = 4 + 5 = 9$$

Node F:

$$f(E) = g(E) + h(E) = 7 + 1 = 8$$

$$f(D) = g(D) + h(D) = 9 + 4 = 13$$

$$f(G) = g(G) + h(G) = 7 + 0 = 7$$

Final path: $A \rightarrow C \rightarrow D \rightarrow F \rightarrow G$

Total cost = 7

Conclusion: A good example of heuristic search is A* search algorithm. The performance of such heuristic search algorithm depends upon the quality of heuristic function. A* algorithm is executed and the path with minimum cost from source node to destination node is calculated.

Lab Assignment No.	A3
Title	Implement Greedy search algorithm for any of the following application: <ul style="list-style-type: none">• Selection Sort• Minimum Spanning Tree• Single-Source Shortest Path Problem• Job Scheduling Problem• Prim's Minimal Spanning Tree Algorithm• Kruskal's Minimal Spanning Tree Algorithm• Dijkstra's Minimal Spanning Tree Algorithm
Roll No.	
Class	TE
Date of Completion	
Subject	Laboratory Practice II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: A3

Title: Implement Greedy search algorithm for any of the following application:

- Selection Sort
- Minimum Spanning Tree
- Single-Source Shortest Path Problem
- Job Scheduling Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm
- Dijkstra's Minimal Spanning Tree Algorithm

Problem Statement: Implement Greedy search algorithm for any of the following application:

- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm

Objective:

- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm

Outcome:

- Ability to choose an appropriate problem solving method and knowledge representation technique

Software Required:

- Python

Theory:

- Kruskal's Minimum Spanning Tree Algorithm:
 - What is a Spanning Tree?

A Spanning tree is a subset to a connected graph G , where all the edges are connected, i.e, we can traverse to any edge from a particular edge with or without intermediates. Also, a spanning tree must not have any cycle in it. Thus we can say that if there are n vertices in a connected graph then the no. of edges that a spanning tree may have is $n-1$.

- 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.

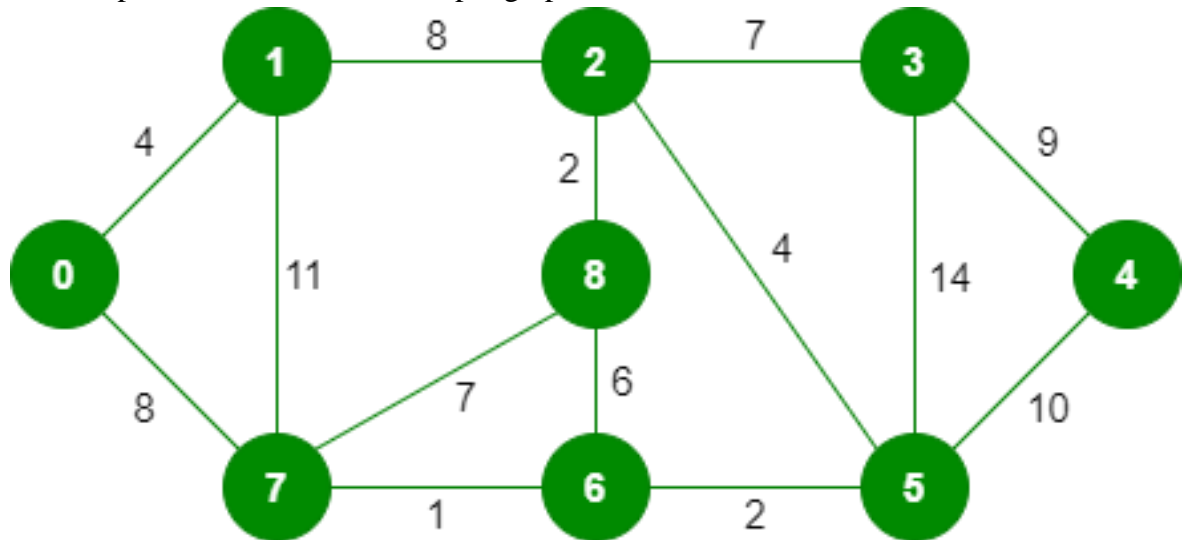
How many edges does a minimum spanning tree has?

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

What are the applications of the Minimum Spanning Tree?

See this for applications of MST.

- 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.
- 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.



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

Now pick all edges one by one from the sorted list of edges

1. Pick edge 7-6: No cycle is formed, include it.



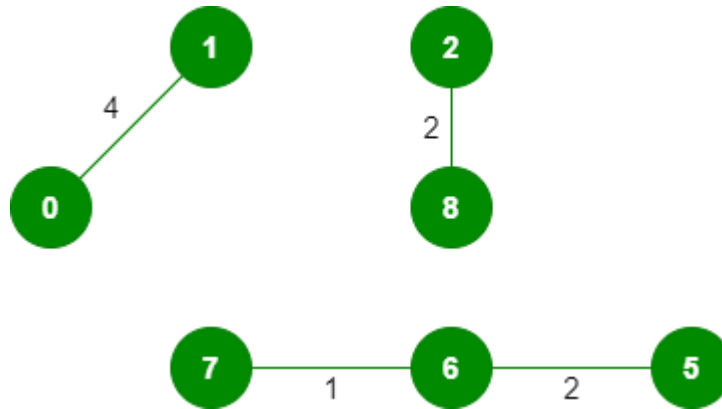
2. Pick edge 8-2: No cycle is formed, include it.



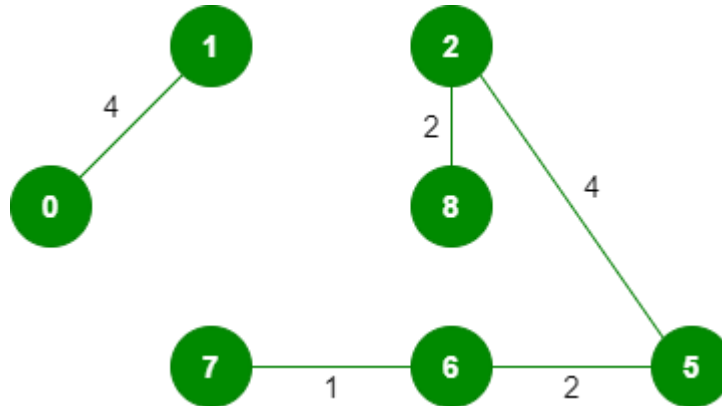
3. Pick edge 6-5: No cycle is formed, include it.



4. Pick edge 0-1: No cycle is formed, include it.

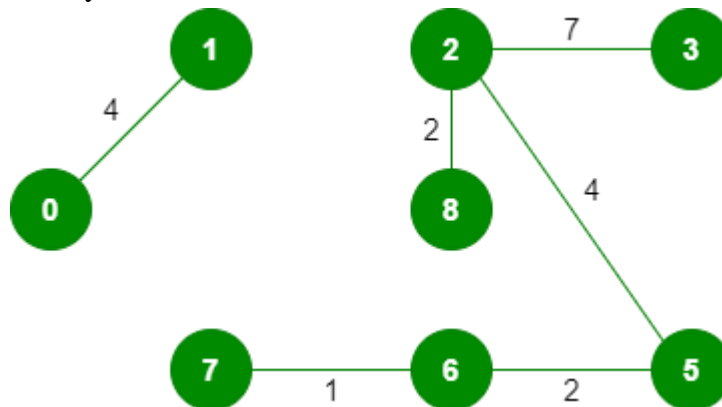


5. Pick edge 2-5: No cycle is formed, include it.



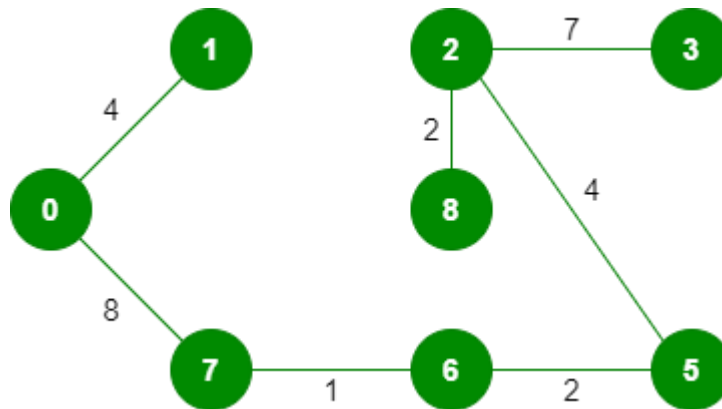
6. Pick edge 8-6: Since including this edge results in the cycle, discard it.

7. Pick edge 2-3: No cycle is formed, include it.



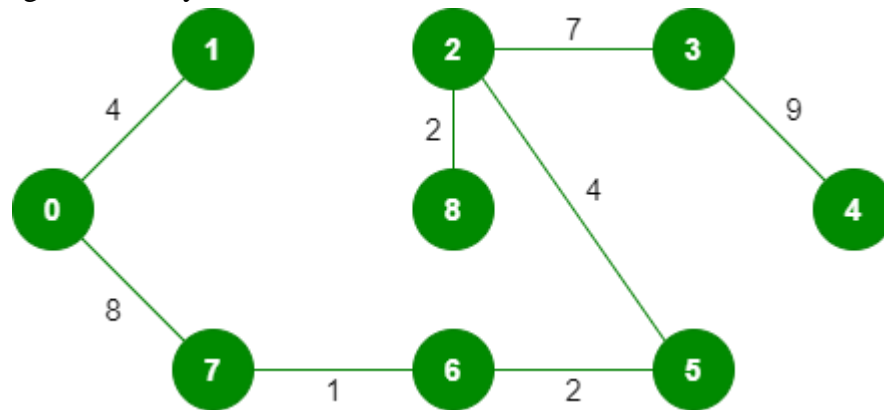
8. Pick edge 7-8: Since including this edge results in the cycle, discard it.

9. Pick edge 0-7: No cycle is formed, include it.



10. Pick edge 1-2: Since including this edge results in the cycle, discard it.

11. Pick edge 3-4: No cycle is formed, include it.



Since the number of edges included equals $(V - 1)$, the algorithm stops here.

- **Prim's Minimum Spanning Tree Algorithm:**

We have discussed Kruskal's algorithm for Minimum Spanning Tree. Like Kruskal's algorithm, Prim's algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

A group of edges that connects two sets of vertices in a graph is called cut in graph theory. So, at every step of Prim's algorithm, we find a cut (of two sets, one contains the vertices already included in MST and the other contains the rest of the vertices), pick the minimum weight edge from the cut, and include this vertex to MST Set (the set that contains already included vertices).

- **How does Prim's Algorithm Work?**

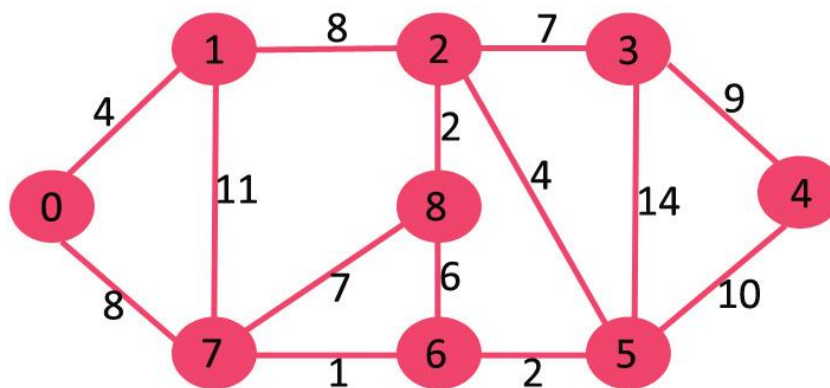
The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.

➤ Algorithm:

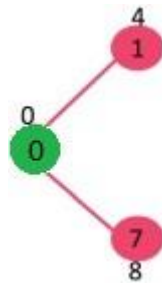
1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign the key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
 - a) Pick a vertex u which is not there in mstSet and has a minimum key value.
 - b) Include u to mstSet.
 - c) Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if the weight of edge u-v is less than the previous key value of v, update the key value as the weight of u-v

The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices that are not yet included in MST, the key value for these vertices indicates the minimum weight edges connecting them to the set of vertices included in MST.

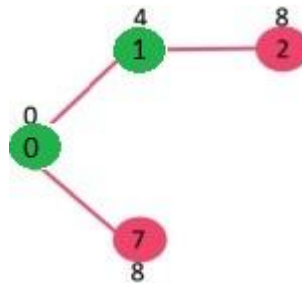
➤ Let us understand with the following example:



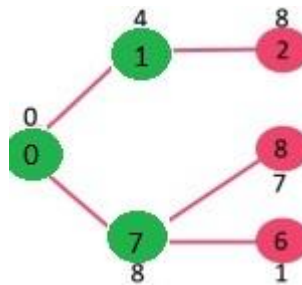
The set mstSet is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with the minimum key value. The vertex 0 is picked, include it in mstSet. So mstSet becomes {0}. After including to mstSet, update key values of adjacent vertices. Adjacent vertices of 0 are 1 and 7. The key values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



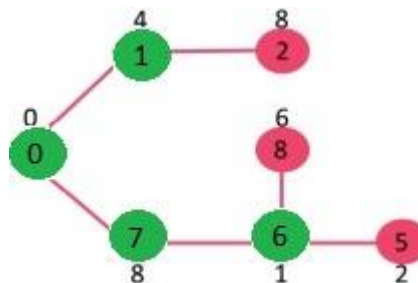
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8



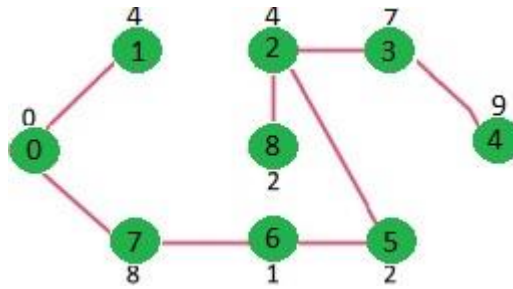
Pick the vertex with minimum key value and not already included in MST (not in mstSET). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstSet now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (1 and 7 respectively).



Pick the vertex with minimum key value and not already included in MST (not in mstSET). Vertex 6 is picked. So mstSet now becomes {0, 1, 7, 6}. Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.



We repeat the above steps until mstSet includes all vertices of given graph. Finally, we get the following graph.



Conclusion: Thus we have implemented Greedy search algorithm Prim's Minimal Spanning Tree Algorithm and Kruskal's Minimal Spanning Tree Algorithm.

Group B

Lab Assignment No.	B4
Title	Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.
Roll No.	
Class	TE
Date of Completion	
Subject	Laboratory Practice II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: B4

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

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

Objective:

- Understand and implement Constraint Satisfaction Problem using Branch and Bound for n-queens problem
- Understand and implement Constraint Satisfaction Problem using Backtracking for n-queens problem

Outcome:

- Ability to choose an appropriate problem solving method and knowledge representation technique

Software Required:

- Python

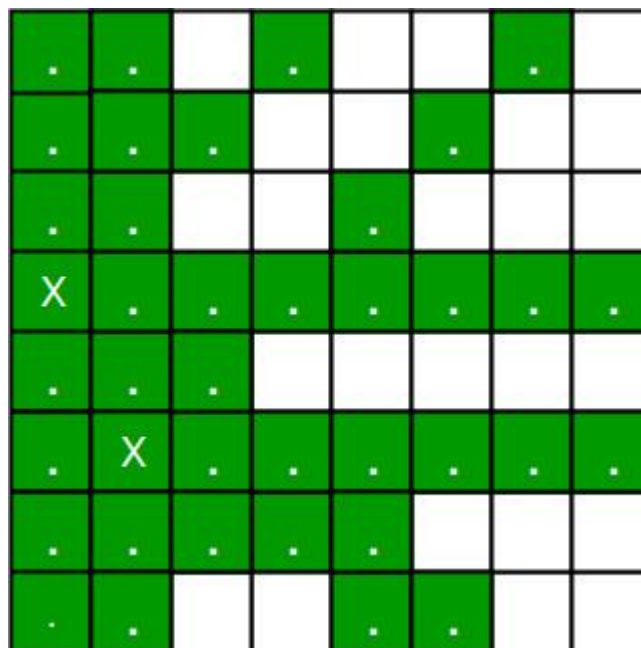
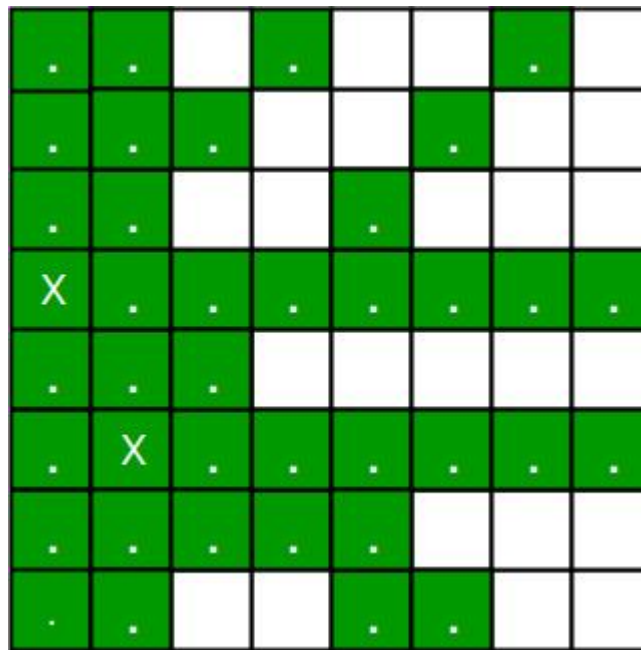
Theory:

- N Queen Problem using Branch and Bound:

The N queens puzzle is the problem of placing N chess queens on an $N \times N$ 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.
2. 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.
3. 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.

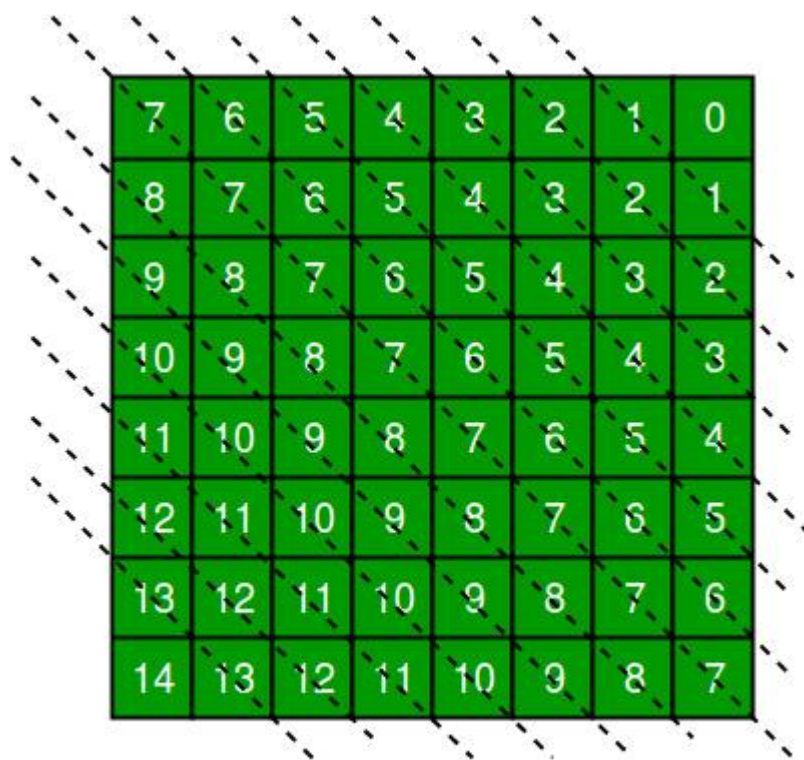
Lets do some pre-processing first. Let's create two $N \times N$ matrix one for / diagonal and other one for \ diagonal. Let's call them slashCode and backslashCode respectively. The trick is to fill them in such a way that two queens sharing a same /-diagonal will have the same value in matrix slashCode,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

For an $N \times N$ matrix, fill slashCode and backslashCode matrix using below formula –

$$\text{slashCode}[\text{row}][\text{col}] = \text{row} + \text{col}$$

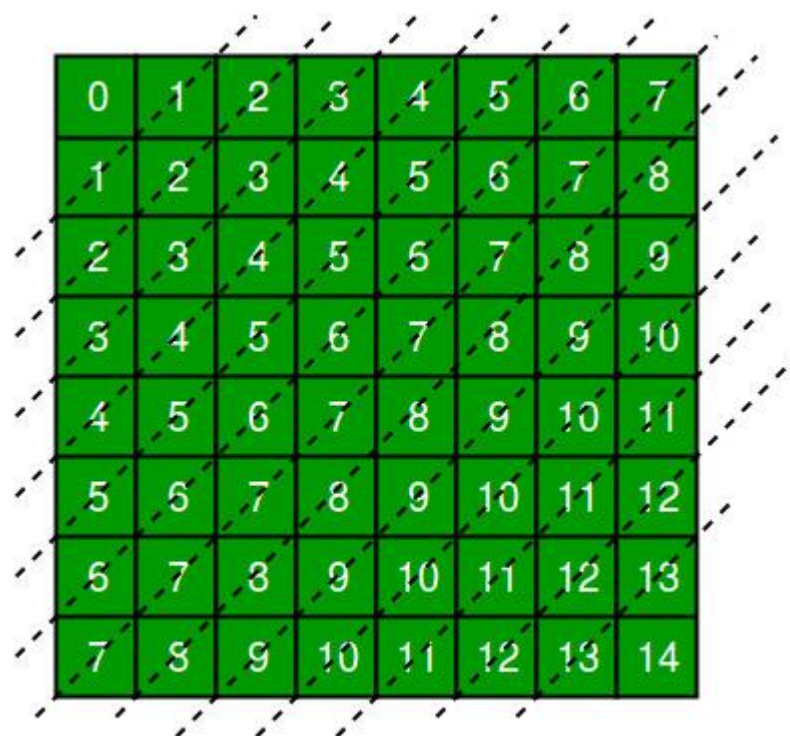
$$\text{backslashCode}[\text{row}][\text{col}] = \text{row} - \text{col} + (N-1)$$

Using above formula will result in below matrices



7	6	5	4	3	2	1	0
8	7	6	5	4	3	2	1
9	8	7	6	5	4	3	2
10	9	8	7	6	5	4	3
11	10	9	8	7	6	5	4
12	11	10	9	8	7	6	5
13	12	11	10	9	8	7	6
14	13	12	11	10	9	8	7

$$r - c + 7$$



0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14

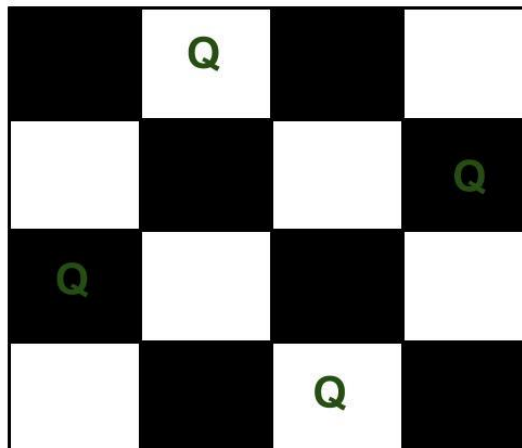
$$r + c$$

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.

- **N Queen Problem using Backtracking:**

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



The expected output is a binary matrix that has 1s for the blocks where queens are placed. For example, the following is the output matrix for the 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.

1. Start in the leftmost column
2. If all queens are placed
return true
3. Try all rows in the current column.
Do following for every tried row.
 - a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
 - b) If placing the queen in [row, column] leads to a solution then return true.
 - c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
4. If all rows have been tried and nothing worked, return false to trigger backtracking.

Conclusion: Thus we have Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem.

Lab Assignment No.	B5
Title	Develop an elementary chatbot for any suitable customer interaction application.
Roll No.	
Class	TE
Date of Completion	
Subject	Laboratory Practice II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: B5

Title: Develop an elementary chatbot for any suitable customer interaction application.

Problem Statement: Develop an elementary chatbot for any suitable customer interaction application.

Objective:

- Understand and Implement chatbot for any suitable customer interaction application

Outcome:

- Ability to choose an appropriate problem solving method and knowledge representation technique

Software Required:

- Python

Theory:

- Understanding the Chatbot:

A Chatbot is an Artificial Intelligence-based software developed to interact with humans in their natural languages. These chatbots are generally converse through auditory or textual methods, and they can effortlessly mimic human languages to communicate with human beings in a human-like way. A chatbot is considered one of the best applications of natural languages processing.

- Two category the Chatbots:

- Rule-based Chatbots:

The Rule-based approach trains a chatbot to answer questions based on a list of pre-determined rules on which it was primarily trained. These set rules can either be pretty simple or quite complex, and we can use these rule-based chatbots to handle simple queries but not process more complicated requests or queries.

- Self-learning Chatbots:

Self-learning chatbots are chatbots that can learn on their own. These leverage advanced technologies such as Artificial Intelligence (AI) and Machine Learning (ML) to train themselves from behaviours and instances. Generally, these chatbots are quite smarter

than rule-based bots. We can classify the Self-learning chatbots furtherly into two categories - Retrieval-based Chatbots and Generative Chatbots.

- Retrieval-based Chatbots:

A retrieval-based chatbot works on pre-defined input patterns and sets responses. Once the question or pattern is inserted, the chatbot utilizes a heuristic approach to deliver the relevant response. The model based on retrieval is extensively utilized to design and develop goal-oriented chatbots using customized features such as the flow and tone of the bot in order to enhance the experience of the customer.

- Generative Chatbots:

Unlike retrieval-based chatbots, generative chatbots are not based on pre-defined responses - they leverage seq2seq neural networks. This is constructed on the concept of machine translation, where the source code is converted from one language to another language. In the seq2seq approach, the input is changed into an output.

The first chatbot named ELIZA was designed and developed by Joseph Weizenbaum in 1966 that could imitate the language of a psychotherapist in only 200 lines of code. But as the technology gets more advance, we have come a long way from scripted chatbots to chatbots in Python today.

- Chatbot in present Generation:

Today, we have smart Chatbots powered by Artificial Intelligence that utilize natural language processing (NLP) in order to understand the commands from humans (text and voice) and learn from experience. Chatbots have become a staple customer interaction utility for companies and brands that have an active online existence (website and social network platforms).

With the help of Python, Chatbots are considered a nifty utility as they facilitate rapid messaging between the brand and the customer. Let us think about Microsoft's Cortana, Amazon's Alexa, and Apple's Siri. Aren't these chatbots wonderful? It becomes quite interesting to learn how to create a chatbot using the Python programming language.

Fundamentally, the chatbot utilizing Python is designed and programmed to take in the data we provide and then analyze it using the complex algorithms for Artificial Intelligence. It then delivers us either a written response or a verbal one. Since these bots can learn from experiences and behavior, they can respond to a large variety of queries and commands.

Although chatbot in Python has already started to rule the tech scenario at present, chatbots had handled approximately 85% of the customer-brand interactions by 2020 as per the prediction of Gartner.

In light of the increasing popularity and adoption of chatbots in the industry, we can increase the market value by learning how to create a chatbot in Python - among the most extensively utilized programming languages globally.

- Understanding the ChatterBot Library:

ChatterBot is a Python library that is developed to provide automated responses to user inputs. It makes utilization of a combination of Machine Learning algorithms in order to generate multiple types of responses. This feature enables developers to construct chatbots using Python that can communicate with humans and provide relevant and appropriate responses. Moreover, the ML algorithms support the bot to improve its performance with experience.

Another amazing feature of the ChatterBot library is its language independence. The library is developed in such a manner that makes it possible to train the bot in more than one programming language.

- Understanding the working of the ChatterBot library:

When a user inserts a particular input in the chatbot (designed on ChatterBot), the bot saves the input and the response for any future usage. This information (of gathered experiences) allows the chatbot to generate automated responses every time a new input is fed into it.

The program picks the most appropriate response from the nearest statement that matches the input and then delivers a response from the already known choice of statements and responses. Over time, as the chatbot indulges in more communications, the precision of reply progresses.

- Creating a Chatbot using Python:

We will follow a step-by-step approach and break down the procedure of creating a Python chat.

We will begin building a Python chatbot by importing all the required packages and modules necessary for the project. We will also initialize different variables that we want to use in it. Moreover, we will also be dealing with text data, so we have to perform data preprocessing on the dataset before designing an ML model.

This is where tokenizing supports text data - it converts the large text dataset into smaller, readable chunks (such as words). Once this process is complete, we can go for lemmatization to transform a word into its lemma form. Then it generates a pickle file in order to store the objects of Python that are utilized to predict the responses of the bot.

Another major section of the chatbot development procedure is developing the training and testing datasets.

- Why Chatbots are important for a Business or a Website:
 - Quick resolution for a complaint or a problem.
 - Improve business branding thereby achieving great customer satisfaction.
 - Answering questions and answers for customers.
 - Making a reservation at hotel or at restaurant.
 - Save human effort 24×7.
 - Enhance business revenue by providing ideas and inspirations.
 - Finding details about business such as hours of operation, phone number and address.
 - Automate sales and lead generation process.
 - Reduce customer agents waiting time answering phone calls.
- Benefits of using Chatbots:
 - 24×7 availability.
 - Instant answers to queries.
 - Support multi-language to enhance businesses.
 - Simple and Easy to Use UI to engage more customers.
 - Cost effective and user interactive.
 - Avoid communication with call agents thereby reducing the time consuming tasks.
 - Understand the Customer behavior
 - Increase sales of business by offering promo codes or gifts.

Conclusion: Thus we have developed an elementary chatbot for any suitable customer interaction application.

Group C

Lab Assignment No.	C6
Title	Mini Project: Implement any one of the following Expert System <ul style="list-style-type: none">• Information management• Hospitals and medical facilities• Help desks management• Employee performance evaluation• Stock market trading• Airline scheduling and cargo schedules
Roll No.	
Class	TE
Date of Completion	
Subject	Laboratory Practice II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: C6

Title: Implement any one of the following Expert System

- Information management
- Hospitals and medical facilities
- Help desks management
- Employee performance evaluation
- Stock market trading
- Airline scheduling and cargo schedules

Problem Statement: Implement any one of the following Expert System

- Information management
- Hospitals and medical facilities
- Help desks management
- Employee performance evaluation
- Stock market trading
- Airline scheduling and cargo schedules

Objective:

- Understand and implement Expert System

Outcome:

- Ability to choose an appropriate problem solving method and knowledge representation technique

Software Required:

- Python

Theory:

- Expert Systems:

Artificial Intelligence is a piece of software that simulates the behaviour and judgement of a human or an organization that has experts in a particular domain is known as an expert system. It does this by acquiring relevant knowledge from its knowledge base and interpreting it according to the user's problem. The data in the knowledge base is added by humans that are expert in a particular domain and this software is used by a non-expert user to acquire some information. It is widely used in many areas such as medical diagnosis, accounting, coding, games etc.

An expert system is AI software that uses knowledge stored in a knowledge base to solve problems that would usually require a human expert thus preserving a human expert's knowledge in its knowledge base. They can advise users as well as provide explanations to them about how they reached a particular conclusion or advice. Knowledge Engineering is the term used to define the process of building an Expert System and its practitioners are called Knowledge Engineers. The primary role of a knowledge engineer is to make sure that the computer possesses all the knowledge required to solve a problem. The knowledge engineer must choose one or more forms in which to represent the required knowledge as a symbolic pattern in the memory of the computer.

- **Example :**

There are many examples of an expert system. Some of them are given below –

- **MYCIN –**

One of the earliest expert systems based on backward chaining. It can identify various bacteria that can cause severe infections and can also recommend drugs based on the person's weight.

- **DENDRAL –**

It was an artificial intelligence-based expert system used for chemical analysis. It used a substance's spectrographic data to predict its molecular structure.

- **R1/XCON –**

It could select specific software to generate a computer system wished by the user.

- **PXDES –**

It could easily determine the type and the degree of lung cancer in a patient based on the data.

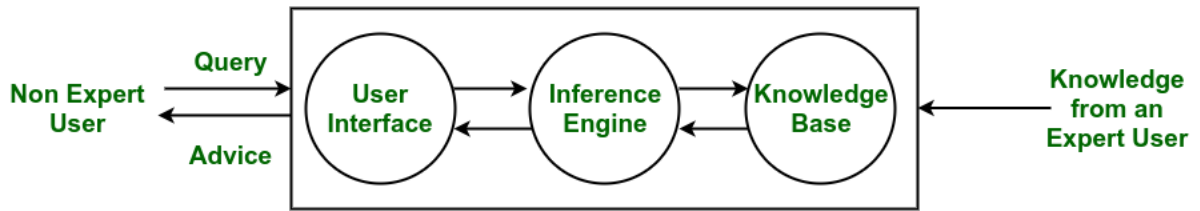
- **CaDet –**

It is a clinical support system that could identify cancer in its early stages in patients.

- **DXplain –**

It was also a clinical support system that could suggest a variety of diseases based on the findings of the doctor.

- **Components of an Expert System:**



Architecture of an Expert System

➤ Knowledge Base –

The knowledge base represents facts and rules. It consists of knowledge in a particular domain as well as rules to solve a problem, procedures and intrinsic data relevant to the domain.

➤ Inference Engine –

The function of the inference engine is to fetch the relevant knowledge from the knowledge base, interpret it and to find a solution relevant to the user's problem. The inference engine acquires the rules from its knowledge base and applies them to the known facts to infer new facts. Inference engines can also include an explanation and debugging abilities.

➤ Knowledge Acquisition and Learning Module –

The function of this component is to allow the expert system to acquire more and more knowledge from various sources and store it in the knowledge base.

➤ User Interface –

This module makes it possible for a non-expert user to interact with the expert system and find a solution to the problem.

➤ Explanation Module –

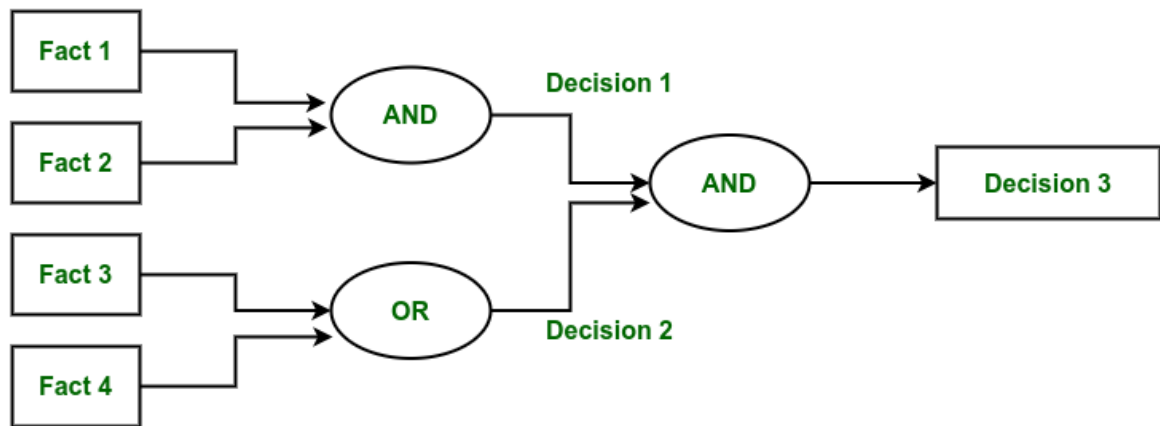
This module helps the expert system to give the user an explanation about how the expert system reached a particular conclusion.

- Two strategies:

The Inference Engine generally uses two strategies for acquiring knowledge from the Knowledge Base, namely –

➤ Forward Chaining:

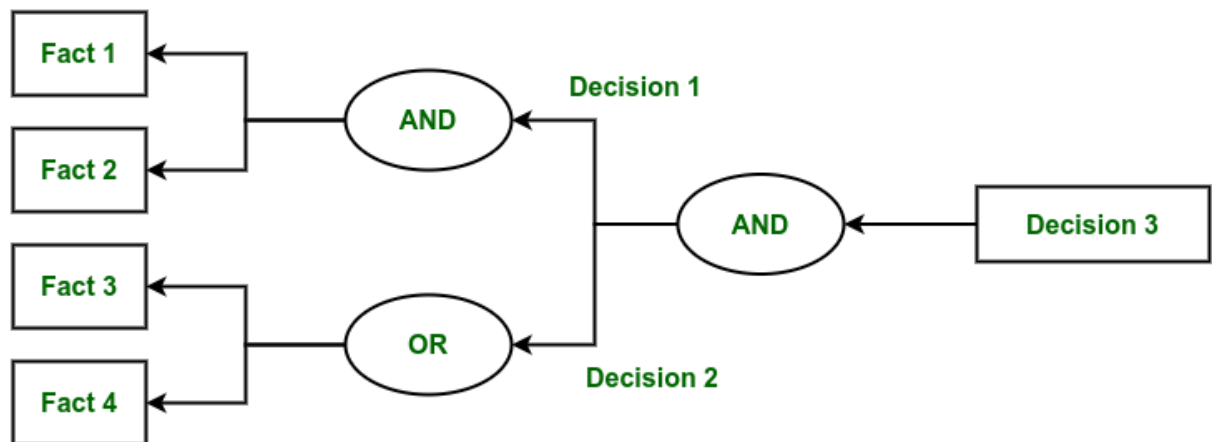
It is a strategic process used by the Expert System to answer the questions – What will happen next. This strategy is mostly used for managing tasks like creating a conclusion, result or effect. Example – prediction or share market movement status.



Forward Chaining

➤ Backward Chaining

It is a storage used by the Expert System to answer the questions – Why this has happened. This strategy is mostly used to find out the root cause or reason behind it, considering what has already happened. Example – diagnosis of stomach pain, blood cancer or dengue, etc.



Backward Chaining

• Characteristics of an Expert System:

- Human experts are perishable, but an expert system is permanent.
- It helps to distribute the expertise of a human.
- One expert system may contain knowledge from more than one human experts thus making the solutions more efficient.
- It decreases the cost of consulting an expert for various domains such as medical diagnosis.
- They use a knowledge base and inference engine.

- Expert systems can solve complex problems by deducing new facts through existing facts of knowledge, represented mostly as if-then rules rather than through conventional procedural code.
- Expert systems were among the first truly successful forms of artificial intelligence (AI) software.
- Limitations:
 - Do not have human-like decision-making power.
 - Cannot possess human capabilities.
 - Cannot produce correct result from less amount of knowledge.
 - Requires excessive training.
- Advantages:
 - Low accessibility cost.
 - Fast response.
 - Not affected by emotions, unlike humans.
 - Low error rate.
 - Capable of explaining how they reached a solution.
- Disadvantages:
 - The expert system has no emotions.
 - Common sense is the main issue of the expert system.
 - It is developed for a specific domain.
 - It needs to be updated manually. It does not learn itself.
 - Not capable to explain the logic behind the decision.
- Applications:

The application of an expert system can be found in almost all areas of business or government. They include areas such as –

- Different types of medical diagnosis like internal medicine, blood diseases and show on.
- Diagnosis of the complex electronic and electromechanical system.
- Diagnosis of a software development project.
- Planning experiment in biology, chemistry and molecular genetics.
- Forecasting crop damage.
- Diagnosis of the diesel-electric locomotive system.
- Identification of chemical compound structure.
- Scheduling of customer order, computer resources and various manufacturing task.
- Assessment of geologic structure from dip meter logs.
- Assessment of space structure through satellite and robot.

- The design of VLSI system.
- Teaching students specialize task.
- Assessment of log including civil case evaluation, product liability etc.

Expert systems have evolved so much that they have started various debates about the fate of humanity in the face of such intelligence, with authors such as Nick Bostrom (Professor of Philosophy at Oxford University), pondering if computing power has transcended our ability to control it.

Conclusion: Thus we have implemented Expert System