



Pimpri Chinchwad Education Trust's  
**Pimpri Chinchwad College of Engineering &  
Research Ravet, Pune**  
**IQAC PCCOER**



**DEPARTMENT OF COMPUTER ENGINEERING**

# **Lab Manual**



## **310258: Laboratory Practice II**

### **(Artificial Intelligence)**

**Third Year of Computer Engineering**

**(2019 Course)**

(Prepared by – Dr. Archana. K)

	<b>Pimpri Chinchwad Education Trust's Pimpri Chinchwad College of Engineering &amp; Research Ravet, Pune IQAC PCCOER</b>	
<b>Academic Year:</b> 2022 – 23 <b>Term: II</b>	<b>Vision – Mission of Department</b>	<b>Record No.:</b> <b>ACAD/R/00</b>

Date: 1.2.2023

**Department: Computer Engineering**

### **Vision – Mission of the Institute**

#### **Vision**

To be a premier institute of technical education and research to serve the need of society and all the stakeholders.

#### **Mission**

To establish state-of-the-art facilities to create an environment resulting in individuals who are technically sound having professionalism, research and innovative aptitude with high moral and ethical values.

### **Vision – Mission of the Computer Department**

#### **Vision**

To strive for excellence in the field of Computer Engineering and Research through Creative Problem Solving related to societal needs

#### **Mission:**

1. Establish strong fundamentals, domain knowledge and skills among the students with analytical thinking, conceptual knowledge, social awareness and expertise in the latest tools & technologies to serve industrial demands.
2. Establish leadership skills, team spirit and high ethical values among the students to serve industrial demands and societal needs.
3. Guide students towards Research and Development, and a willingness to learn by connecting themselves to the global society.

**Program Educational Objectives:**



<b>PO</b>	<b>Program Educational Outcomes</b>
<b>PEO1</b>	To prepare globally competent graduates having strong fundamentals, domain knowledge, updated with modern technology to provide the effective solutions for engineering problems.
<b>PEO2</b>	To prepare the graduates to work as a committed professional with strong professional ethics and values, sense of responsibilities, understanding of legal, safety, health, societal, cultural and environmental issues.
<b>PEO3</b>	To prepare committed and motivated graduates with research attitude, lifelong learning, investigative approach, and multidisciplinary thinking.
<b>PEO4</b>	To prepare the graduates with strong managerial and communication skills to work effectively as individual as well as in teams.

**Program Specific Outcomes**

<b>PSO</b>	<b>Program Specific Outcomes</b>
<b>PSO1</b>	The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality project.
<b>PSO2</b>	The ability to understand, analyze and develop computer programs in the areas related to algorithms, software testing, application software, web design, data analytics, IOT and networking for efficient design of computer-based systems.
<b>PSO3</b>	The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies and to generate IPR & Deliver a quality project.

## Program Outcomes

PO	Program Outcomes
PO1	<b>Engineering knowledge:</b> Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	<b>Problem analysis:</b> Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	<b>Design/development of solutions:</b> Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	<b>Conduct investigations of complex problems:</b> Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	<b>Modern tool usage:</b> Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	<b>The engineer and society:</b> Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	<b>Environment and sustainability:</b> Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	<b>Ethics:</b> Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	<b>Individual and team work:</b> Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	<b>Communication:</b> Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	<b>Project management and finance:</b> Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	<b>Life-long learning:</b> Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

	<b>Pimpri Chinchwad Education Trust's Pimpri Chinchwad College of Engineering &amp; Research Ravet, Pune IQAC PCCOER</b>	
<b>Academic Year: 2022 - 23</b>	<b>Course Objectives &amp; Course Outcomes</b>	<b>Record No.: ACAD/R/08-A Term: II</b>

**Department: Computer Engineering**  
**Subject Name: Laboratory Practice -II**

**Class – TE**  
**Subject Code:[ 310258]**

**Name of Faculty: Dr.Archana.K (Artificial Intelligence)**  
**Mrs.Vaishali Latke (Cloud Computing)**

**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:**

CO	Statements	Cognitive level of learning
<b>C318.1</b>	Design a system using different informed search / uninformed search or heuristic approaches	(Design)
<b>C318.2</b>	Apply basic principles of AI in solutions that require problem solving, inference, perception, knowledge representation, and learning	(Apply)
<b>C318.3</b>	Design and develop an interactive AI application	(Apply)
<b>C318.4</b>	Use tools and techniques in the area of Cloud Computing	(Apply)
<b>C318.5</b>	Use cloud computing services for problem solving	(Apply)
<b>C318.6</b>	Design and develop applications on cloud	(Apply)

**CO-PO Mapping:**

CO\PO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO1 0	PO1 1	PO1 2	PSO 1	PSO 2	PSO 3
<b>C318.1</b>	3	3	3	2	2	-	-	-	2	-	2	2	2	-	2
<b>C318.2</b>	3	3	3	-	3	-	-	-	2	-	2	2	-	-	2
<b>C318.3</b>	3	3	3	2	3	-	-	-	2	-	2	2	2	-	2
<b>C318.4</b>	3	2	-	-	3	-	-	-	2	-	2	2	2	-	2
<b>C318.5</b>	3	2	2	2	3	-	-	-	2	-	2	2	2	-	2
<b>C318.6</b>	3	3	3	-	3	-	-	-	2	-	2	2	2	-	2
<b>AVERAGE:</b>	3	2.67	2.80	2	2.83	-	-	-	2	-	2	2	2	-	2

<b>Savitribai Phule Pune University</b> <b>Third Year of Computer Engineering (2019 Course)</b> <b>310258: Laboratory Practice II</b>		
<b>Teaching Scheme</b> <b>Practical: 04</b> <b>Hours/Week</b>	<b>Credit Scheme 02</b>	<b>Examination Scheme and Marks</b> <b>Term Work: 50 Marks Practical:</b> <b>25 Marks</b>
<b>Companion Course: Artificial Intelligence (310253), Elective II (310245)</b>		
<b>Course Objectives:</b> <ul style="list-style-type: none"> <li>● To learn and apply various search strategies for AI</li> <li>● To Formalize and implement constraints in search problems</li> <li>● To understand the concepts of Cloud Computing</li> </ul>		

### **Course Outcomes:**

On completion of the course, learner will be able to

#### **Artificial Intelligence**

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

#### **Cloud Computing**

CO4: Use tools and techniques in the area of Cloud Computing

CO5: Use the knowledge of Cloud Computing for problem solving

CO6: Apply the concepts Cloud Computing to design and develop applications

### **Guidelines for Instructor's Manual**

The instructor's manual is to be developed as a reference and hands-on resource. It should include prologue (about University/program/ institute/ department/foreword/ preface), curriculum of the course, conduction and Assessment guidelines, topics under consideration, concept, objectives, outcomes, set of typical applications/assignments/ guidelines, and references.

### **Guidelines for Student's Laboratory Journal**

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of Certificate, table of contents, and handwritten write-up of each assignment (Title, Date of Completion, Objectives, Problem Statement, Software and Hardware requirements, Assessment

grade/marks and assessor's sign, Theory- Concept in brief, algorithm, flowchart, test cases, Test Data Set(if applicable), mathematical model (if applicable), conclusion/analysis. Program codes with sample output of all performed assignments are to be submitted as softcopy. As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journal must be avoided. Use of DVD containing students programs maintained by Laboratory In-charge is highly encouraged. For reference one or two journals may be maintained with program prints in the Laboratory.

### **Guidelines for Laboratory /Term Work Assessment**

Continuous assessment of laboratory work should be based on overall performance of Laboratory assignments by a student. Each Laboratory assignment assessment will assign grade/marks based on parameters, such as timely completion, performance, innovation, efficient codes, and punctuality.

### **Guidelines for Practical Examination**

Problem statements must be decided jointly by the internal examiner and external examiner. During practical assessment, maximum weightage should be given to satisfactory implementation of the problem statement. Relevant questions may be asked at the time of evaluation to test the student's understanding of the fundamentals, effective and efficient implementation. This will encourage, transparent evaluation and fair approach, and hence will not create any uncertainty or doubt in the minds of the students. So, adhering to these principles will consummate our team efforts to the promising start of student's academics.

### **Guidelines for Laboratory Conduction**

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy need to address the average students and inclusive of an element to attract and promote the intelligent students. Use of open source software is encouraged. Based on the concepts learned. Instructor may also set one assignment or mini-project that is suitable to respective branch beyond the scope of syllabus.

**Operating System recommended: - 64-bit Windows OS and Linux Programming tools recommended. Artificial Intelligence: C++/Python/Java Cloud Computing Front end: HTML5, Bootstrap, jQuery, JS etc. Backend: MySQL/MongoDB/NodeJS**

Practical Assignment No.	Practical Assignment	Content Delivery Methods (CDM)	COs Mapping to the Contents
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.	Demonstration, Hand's on	C318.1
2	Implement A star Algorithm for any game search problem.	Demonstration, Hand's on	C318.1
3	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	Demonstration, Hand's on	C318.1
4	Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.	Demonstration, Hand's on	C318.2
5	Develop an elementary chatboat for any suitable customer interaction application.	Demonstration, Hand's on	C318.3
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 scheduling	Demonstration, Hand's on	C318.3



## Assignment No: 01

---

**Title of the Assignment:** DFS and BFS using recursive 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:**

- To understand the concept BFS and DFS search techniques.
- To implement BFS and DFS search techniques using recursive functions on undirected graph or tree.

**Theory:**

**Depth-first search (DFS):**

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root for a graph) and explore as far as possible along each branch before backtracking.

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.

Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.

This recursive nature of DFS can be implemented using stacks.

The basic idea is as follows: Pick a starting node and push all its adjacent nodes into a stack. Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack. Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

Note: Show an example for solving for DFS. Take an undirected graph/tree and show the traversal.

**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.

### Complexity Analysis:

- **Time complexity:**  $O(V + E)$ , where  $V$  is the number of vertices and  $E$  is the number of edges in the graph.
- **Space Complexity:**  $O(V)$ , since an extra visited array of size  $V$  is required.

### Pseudocode of recursive BFS:

```
DFS(adjacent[[]], source, visited[], key) {
    if(source == key) return true //We found the key
    visited[source] = True

    FOR node in adjacent[source]:
        IF visited[node] == False:
            DFS(adjacent, node, visited)
        END IF
    END FOR
    return false    // If it reaches here, then all nodes have been explored
                    //and we still havent found the key.
}
```

### Breadth-first search (BFS):

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first before moving to the next-level neighbors.

Note: Show an example for solving for BFS. Take an undirected graph/tree and show the traversal.

### Algorithm:

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.

### Complexity Analysis:

- **Time complexity:** The worst case breadth-first search has to consider all paths to all possible nodes the time complexity of breadth-first search is  $O(|E| + |V|)$  where  $|V|$  and  $|E|$  is the cardinality of set of vertices and edges respectively.
- **Space complexity:** The space complexity can also be expressed as  $O(|V|)$  where  $|V|$  is the cardinality of the set of vertices

**Pseudocode of recursive BFS:**

```

recursiveBFS(Graph graph, Queue q, boolean[] visited, int key){
    if (q.isEmpty())
        return "Not Found";

    // pop front node from queue and print it
    int v = q.poll();
    if(v==key) return "Found";

    // do for every neighbors of node v
    for ( Node u in graph.get(v))
    {
        if (!visited[u])
        {
            // mark it visited and push it into queue
            visited[u] = true;
            q.add(u);
        }
    }
    // recurse for other nodes
    recursiveBFS(graph, q, visited, key);
}

Queue q = new Queue();
q.add(s);
recursiveBFS(graph, q, visited, key);

```

**Conclusion:**

We have implemented BFS and DFS using recursive algorithm for undirected graph.

**Oral questions:**

1. Why do we prefer queues instead of other data structures while implementing BFS?
2. Why is time complexity more in the case of graph being represented as Adjacency Matrix?
3. What are the classifications of edges in a BFS graph?
4. What is the difference between DFS and BFS? When is DFS and BFS used?
5. Can BFS be used for finding shortest possible path?
6. Is BFS a complete algorithm?
7. Is BFS a optimal algorithm?
8. Why can we not implement DFS using Queues? Why do we prefer stacks instead of other data structures?
9. What are the classifications of edges in DFS of a graph?
10. Can DFS be used for finding shortest possible path?
11. Why can we not use DFS for finding shortest possible path?
12. Is DFS a complete algorithm?
13. Is DFS a optimal algorithm?
14. When is it best to use DFS? What are the applications of BFS and DFS?

## Assignment No: 02

---

### Title of the Assignment: A\* algorithm for 8 puzzle problem

**Problem statement:** Implement A star Algorithm for any game search problem.

#### Objective:

- To understand the concept of Informed search techniques.
- To implement A\* algorithm for 8 puzzle game problem.

#### Theory:

##### A\* algorithm:

A\* is a computer algorithm that is widely used in path finding and graph traversal, the process of plotting an efficiently traversable path between multiple points, called nodes.

Noted for its performance and accuracy, it enjoys widespread use.

The key feature of the A\* algorithm is that it keeps a track of each visited node which helps in ignoring the nodes that are already visited, saving a huge amount of time. It also has a list that holds all the nodes that are left to be explored and it chooses the most optimal node from this list, thus saving time not exploring unnecessary or less optimal nodes.

Here we use two lists namely 'open list' and 'closed list'. The open list contains all the nodes that are being generated and are not existing in the closed list and each node explored after its neighboring nodes are discovered is put in the closed list and the neighbors are put in the open list this is how the nodes expand. Each node has a pointer to its parent so that at any given point it can retrace the path to the parent. Initially, the open list holds the start (Initial) node. The next node chosen from the open list is based on its **f score**; the node with the least f score is picked up and explored.

$$\mathbf{f\text{-}score = h\text{-}score + g\text{-}score}$$

A\* uses a combination of heuristic value (h-score: how far the goal node is) as well as the g-score (i.e. the number of nodes traversed from the start node to current node).

- A\* Algorithm is one of the best path finding algorithms.
- But it does not produce the shortest path always.
- This is because it heavily depends on heuristics.

#### Algorithm:

1. Create a search graph G, consisting solely of the start node, no. Put no on a list called OPEN.
2. Create a list called CLOSED that is initially empty.
3. If OPEN is empty, exit with failure.
4. Select the first node on OPEN, remove it from OPEN, and put it on CLOSED. Called this node n.

5. If  $n$  is a goal node, exit successfully with the solution obtained by tracing a path along the pointers from  $n$  to  $n_0$  in  $G$ . (The pointers define a search tree and are established in Step 7)
6. Expand node  $n$ , generating the set  $M$ , of its successors that are not already ancestors of  $n$  in  $G$ . Install these members of  $M$  as successors of  $n$  in  $G$ .
7. Establish a pointer to  $n$  from each of those members of  $M$  that were not already in  $G$  (i.e., not already on either OPEN or CLOSED). Add these members of  $M$  to OPEN. For each member,  $m$ , of  $M$  that was already on OPEN or CLOSED, redirect its pointer to  $n$  if the best path to  $m$  found so far is through  $n$ . For each member of  $M$  already on CLOSED, redirect the pointers of each of its descendants in  $G$  so that they point backward along the best paths found so far to these descendants.
8. Reorder the list OPEN in order of increasing  $f$  values. (Ties among minimal  $f$  values are resolved in favor of the deepest node in the search tree.)
9. Go to Step 3.

### 8-Puzzle problem:

In our 8-Puzzle problem, we can define the **h-score** as the number of misplaced tiles by comparing the current state and the goal state or summation of the Manhattan distance between misplaced nodes.

**g-score** will remain as the number of nodes traversed from a start node to get to the current node. We can calculate the **h-score** by comparing the initial (current) state and goal state and counting the number of misplaced tiles.

Given an initial state of a 8-puzzle problem and final state to be reached-

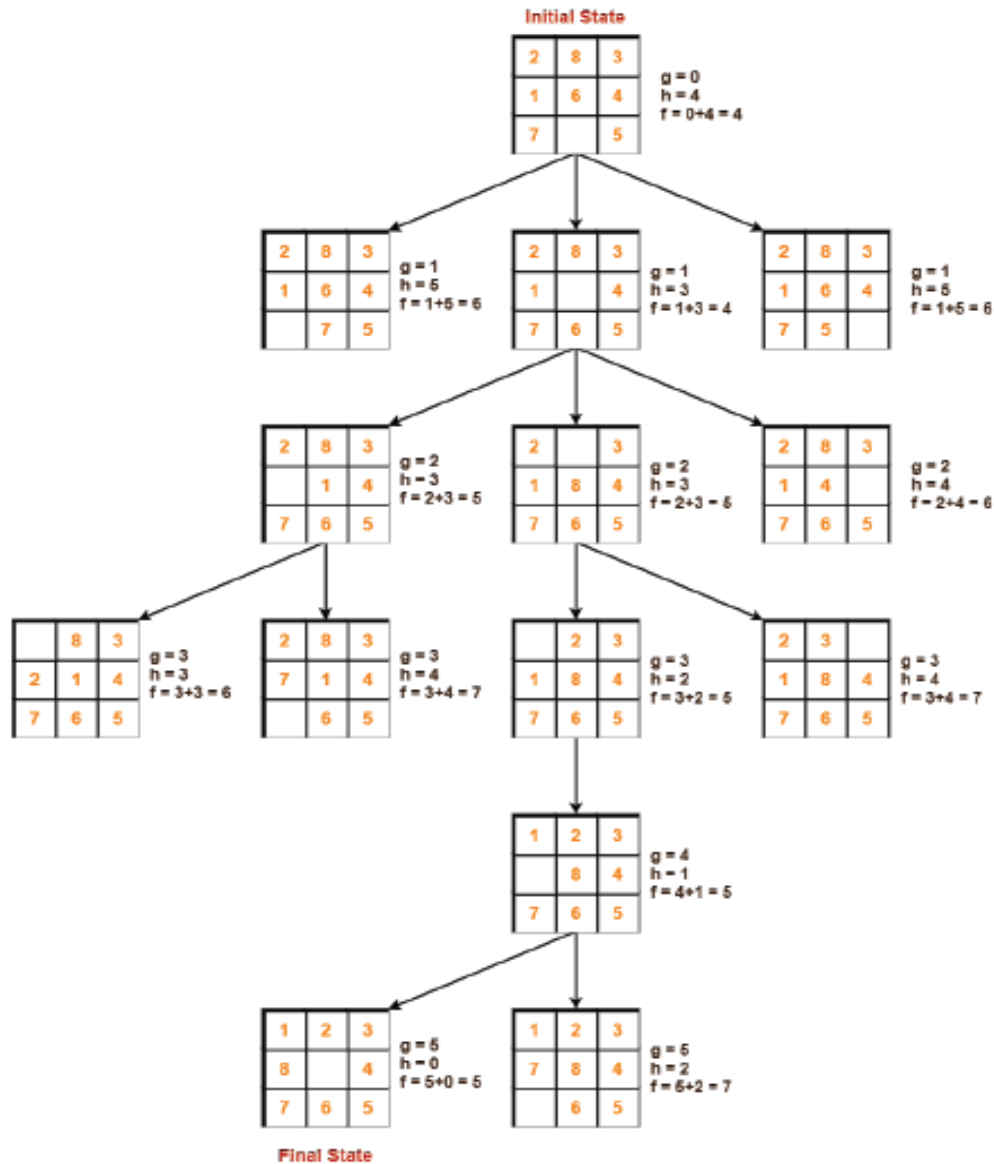
2	8	3
1	6	4
7		5

**Initial State**

1	2	3
8		4
7	6	5

**Final State**

Find the most cost-effective path to reach the final state from initial state using A\* Algorithm. Consider  $g(n)$  = Depth of node and  $h(n)$  = Number of misplaced tiles.

**Conclusion:**

We have implemented BFS and DFS using recursive algorithm for undirected graph.

**Oral questions:**

1. What are informed and uninformed search techniques?
2. List differences between informed and uninformed search techniques?
3. What is the evaluation function of A\* algorithm?
4. What are the applications of A\* algorithm?
5. What is problem solving agent?
6. Is A\* algorithm informed or uninformed search technique?
7. What is Best first search algorithm?
8. Where are searching techniques applied in AI?
9. What are Blind search algorithms?
10. What is meant by Heuristic?

## Assignment No: 03A

---

**Title of the Assignment:** Implement Greedy search algorithm.

**Problem statement:** Implement Dijkstra's Algorithm for finding the shortest path from a source node to all other nodes in a graph (single source shortest path).

**Objective:**

- To understand the concept of greedy search algorithm.
- To implement Dijkstra's shortest path algorithm.

**Theory:**

**Dijkstra algorithm:**

A common example of a graph-based path finding algorithm is Dijkstra's algorithm. This algorithm begins with a start node and an "open set" of candidate nodes. At each step, the node in the open set with the lowest distance from the start is examined. The node is marked "closed", and all nodes adjacent to it are added to the open set if they have not already been examined. This process repeats until a path to the destination has been found. Since the lowest distance nodes are examined first, the first time the destination is found, the path to it will be the shortest path.

A\* algorithm is a variant of Dijkstra's algorithm commonly used in games. A\* assigns a weight to each open node equal to the weight of the edge to that node plus the approximate distance between that node and the finish. This approximate distance is found by the heuristic, and represents a minimum possible distance between that node and the end. This allows it to eliminate longer paths once an initial path is found. If there is a path of length  $x$  between the start and finish, and the minimum distance between a node and the finish is greater than  $x$ , that node need not be examined. A\* uses this heuristic to improve on the behavior relative to Dijkstra's algorithm. When the heuristic evaluates to zero, A\* is equivalent to Dijkstra's algorithm.

Pathfinding is closely related to the shortest path problem, within graph theory, which examines how to identify the path that best meets some criteria (shortest, cheapest, fastest, etc) between two points in a large network.

Pathfinding or pathing is the plotting, by a computer application, of the shortest route between two points. It is a more practical variant on solving mazes. This field of research is based heavily on Dijkstra's algorithm for finding the shortest path on a weighted graph.

Dijkstra's algorithm has many variants but the most common one is to find the shortest paths from the source vertex to all other vertices in the graph.

**Algorithm:**

1. Set all vertices distances = infinity except for the source vertex, set the source distance = 0.

2. Push the source vertex in a min-priority queue in the form (distance, vertex), as the comparison in the min-priority queue will be according to vertices distances.
3. Pop the vertex with the minimum distance from the priority queue (at first the popped vertex = source).
4. Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance", then push the vertex with the new distance to the priority queue.
5. If the popped vertex is visited before, just continue without using it.
6. Apply the same algorithm again until the priority queue is empty.

### Pseudo Code:

```

Dijkstra_Algorithm(source, G):
    """
    parameters: source node--> source, graph--> G
    return: List of cost from source to all other nodes--> cost
    """
    unvisited_list = []           // List of unvisited vertices
    cost = []
    cost[source] = 0              // Distance (cost) from source to source will be 0
    for each vertex v in G:      // Assign cost as INFINITY to all vertices
        if v ≠ source
            cost[v] = INFINITY
            add v to unvisited_list // All nodes pushed to unvisited_list initially

    while unvisited_list is not empty: // Main loop
        v = vertex in unvisited_list with min cost[v] // v is the source node for first iteration
        remove v from unvisited_list // Marking node as visited

        for each neighbor u of v: // Assign shorter path cost to neighbor u
            cost_value = Min( cost[u], cost[v] + edge_cost(v, u) ]
            cost[u] = cost_value // Update cost of vertex u

    return cost

```

### Algorithmic Analysis:

- Worst case time complexity:  $\Theta(E+V \log V)$
- Average case time complexity:  $\Theta(E+V \log V)$
- Best case time complexity:  $\Theta(E+V \log V)$
- Space complexity:  $\Theta(V)$
- Time complexity is  $\Theta(E+V^2)$  if priority queue is not used.

Note: Solve an example problem using Dijkstra's Minimal Spanning Tree algorithm for weighted graph.



**Conclusion:**

We have implemented Dijkstra's Shortest Path Algorithm.

**Oral questions:**

1. Which kind of AI problems can be solved by Dijkstra's Algorithm?
2. Which is the most commonly used data structure for implementing Dijkstra's Algorithm?
3. What is the time and space complexity of Dijkstra's algorithm?
4. In a weighted graph, assume that the shortest path from a source's' to a destination't' is correctly calculated using a shortest path algorithm. Is the following statement true? If we increase weight of every edge by 1, the shortest path always remains same?
5. Given a directed graph where weight of every edge is same, we can efficiently find shortest path from a given source to destination using which algorithm?
6. Is the following statement valid? *Given a weighted graph where weights of all edges are unique (no two edge have same weights), there is always a unique shortest path from a source to destination in such a graph.*

## Assignment No: 03B

**Title of the Assignment:** Implement Greedy search algorithm.

**Problem statement:** Represent graph using adjacency list or matrix and generate minimum spanning tree using Prim's algorithm.

**Objective:**

- To understand the concept of greedy search algorithm.
- To implement Prim's algorithm.

**Theory:**

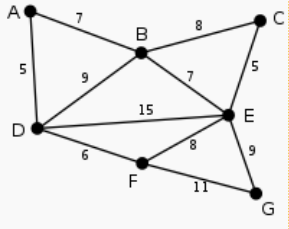
Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm was developed in 1930 by Czech mathematician Vojtěch Jarník and later independently by computer scientist Robert C. Prim in 1957 and rediscovered by Edsger Dijkstra in 1959. Therefore it is also sometimes called the DJP algorithm, the Jarník algorithm, or the Prim–Jarník algorithm.

**Algorithm:**

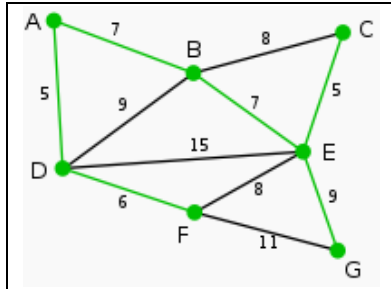
The only spanning tree of the empty graph (with an empty vertex set) is again the empty graph. The following description assumes that this special case is handled separately. The algorithm continuously increases the size of a tree, one edge at a time, starting with a tree consisting of a single vertex, until it spans all vertices.

- Input: A non-empty connected weighted graph with vertices  $V$  and edges  $E$  (the weights can be negative).
- Initialize:  $V_{\text{new}} = \{x\}$ , where  $x$  is an arbitrary node (starting point) from  $V$ ,  $E_{\text{new}} = \{\}$
- Repeat until  $V_{\text{new}} = V$ :
  - Choose an edge  $(u, v)$  with minimal weight such that  $u$  is in  $V_{\text{new}}$  and  $v$  is not (if there are multiple edges with the same weight, any of them may be picked)
  - Add  $v$  to  $V_{\text{new}}$ , and  $(u, v)$  to  $E_{\text{new}}$
- Output:  $V_{\text{new}}$  and  $E_{\text{new}}$  describe a minimal spanning tree

**Example:**

Image	Description
	<p>This is our original weighted graph. The numbers near the edges indicate their weight.</p>

	<p>Vertex <b>D</b> has been arbitrarily chosen as a starting point. Vertices <b>A</b>, <b>B</b>, <b>E</b> and <b>F</b> are connected to <b>D</b> through a single edge. <b>A</b> is the vertex nearest to <b>D</b> and will be chosen as the second vertex along with the edge <b>AD</b>.</p>
	<p>The next vertex chosen is the vertex nearest to <i>either</i> <b>D</b> or <b>A</b>. <b>B</b> is 9 away from <b>D</b> and 7 away from <b>A</b>, <b>E</b> is 15, and <b>F</b> is 6. <b>F</b> is the smallest distance away, so we highlight the vertex <b>F</b> and the arc <b>DF</b>.</p>
	<p>The algorithm carries on as above. Vertex <b>B</b>, which is 7 away from <b>A</b>, is highlighted</p>
	<p>In this case, we can choose between <b>C</b>, <b>E</b>, and <b>G</b>. <b>C</b> is 8 away from <b>B</b>, <b>E</b> is 7 away from <b>B</b>, and <b>G</b> is 11 away from <b>F</b>. <b>E</b> is nearest, so we highlight the vertex <b>E</b> and the arc <b>BE</b>.</p>
	<p>Here, the only vertices available are <b>C</b> and <b>G</b>. <b>C</b> is 5 away from <b>E</b>, and <b>G</b> is 9 away from <b>E</b>. <b>C</b> is chosen, so it is highlighted along with the arc <b>EC</b>.</p>
	<p>Vertex <b>G</b> is the only remaining vertex. It is 11 away from <b>F</b>, and 9 away from <b>E</b>. <b>E</b> is nearer, so we highlight it and the arc <b>EG</b>.</p>



Now all the vertices have been selected and the minimum spanning tree is shown in green. In this case, it has weight 39.

### ALGORITHMIC ANALYSIS:

Minimum edge weight data structure	Time complexity(total)
adjacency matrix, searching	$O(V^2)$
binary heap and adjacency list	$O((V+E)\log(V)) = O(E \log(V))$
Fibonacci heap and adjacency list	$O(E+V \log(V))$

### INPUT AND EXPECTED OUTPUT:

Enter number of vertices: 8

Enter number of edges: 10

Enter the list of edges:

Enter an edge (u,v, weight) :0 1 2

Enter an edge (u,v, weight) :0 2 1

Enter an edge (u,v, weight) :1 3 4

Enter an edge (u,v, weight) :1 4 2

Enter an edge (u,v, weight) :2 5 2

Enter an edge (u,v, weight) :2 6 3

Enter an edge (u,v, weight) :3 7 4

Enter an edge (u,v, weight) :4 7 3

Enter an edge (u,v, weight) :5 7 6

Enter an edge (u,v, weight) :6 7 2

Spanning tree matrix

List of edges in the spanning tree:

(u,v, weight)=(0,1,2)

(u,v, weight)=(0,2,1)

(u,v, weight)=(1,3,4)

(u,v, weight)=(1,4,2)

(u,v, weight)=(2,5,2)

(u,v, weight)=(2,6,3)

(u,v, weight)=(6,7,2)

Total cost of spanning tree= 16

## **CONCLUSION:**

Implementation of Prim's algorithm to find a minimum spanning tree.

## **Oral questions:**

1. What is a minimum spanning tree?
2. What is Time complexity of Prim's algorithm?
3. Prim's algorithm is also known as?

## Assignment No: 03C

**Title of the Assignment:** Implement Greedy search algorithm.

**Problem statement:** Implement Kruskal's algorithm to find minimum spanning tree.

**Objective:**

- To understand the concept of greedy search algorithm.
- To implement Kruskal's algorithm.

**Theory:**

Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

- form a tree that includes every vertex
- has the minimum sum of weights among all the trees that can be formed from the graph

It falls under a class of algorithms called greedy algorithms that find the local optimum in the hopes of finding a global optimum.

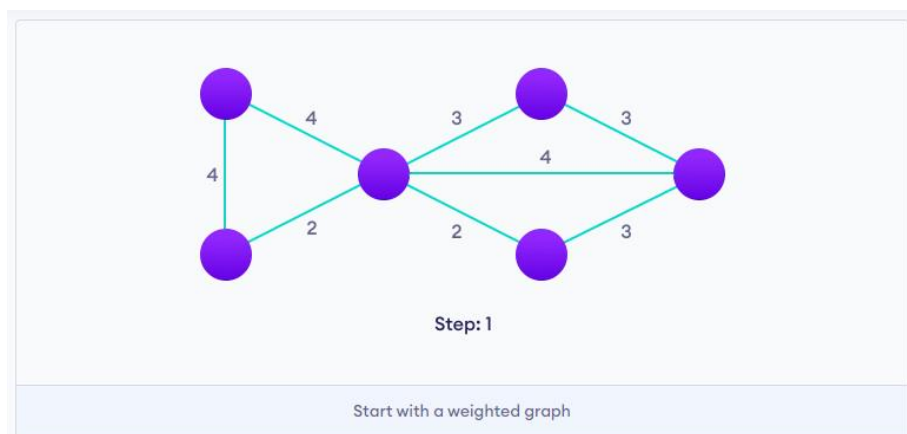
We start from the edges with the lowest weight and keep adding edges until we reach our goal.

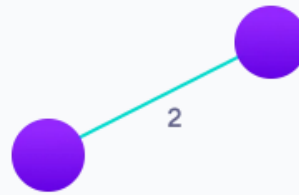
**Algorithm steps:**

The steps for implementing Kruskal's algorithm are as follows:

1. Sort all the edges from low weight to high
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

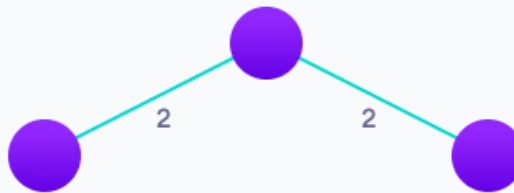
**Example of Kruskal's algorithm:**





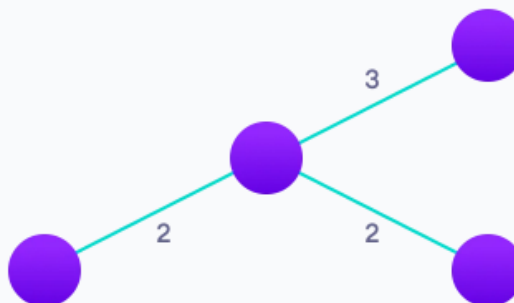
**Step: 2**

Choose the edge with the least weight, if there are more than 1, choose anyone



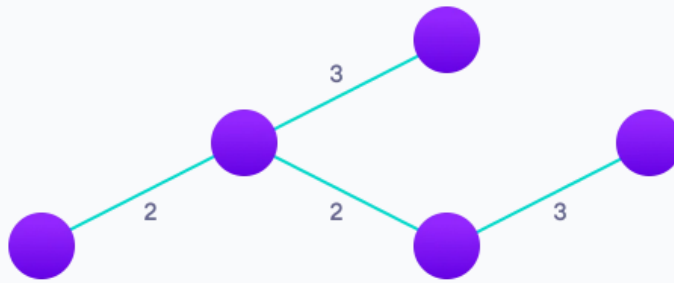
**Step: 3**

Choose the next shortest edge and add it



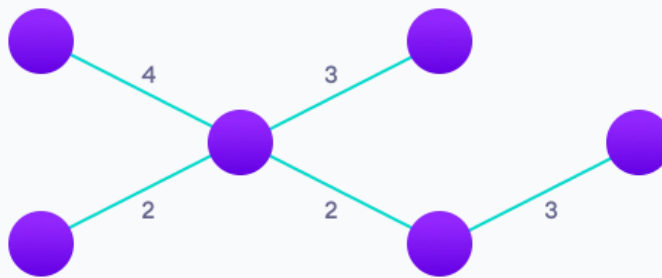
**Step: 4**

Choose the next shortest edge that doesn't create a cycle and add it



Step: 5

Choose the next shortest edge that doesn't create a cycle and add it



Step: 6

Repeat until you have a spanning tree

### Kruskal Algorithm Pseudocode:

Any minimum spanning tree algorithm revolves around checking if adding an edge creates a loop or not.

```

KRUSKAL (G):
A = ∅
For each vertex v ∈ G.V:
    MAKE-SET(v)
For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
    if FIND-SET(u) ≠ FIND-SET(v):
        A = A ∪ {(u, v)}
        UNION (u, v)
return A
  
```



### Algorithm Complexity: Running Time of Kruskal's

Lets assume that we are finding MST of a  $N$  vertices graph using Kruskal's.

- To check edges we need to sort the given edges based on weights of edges. The best way to sort has a order of  $O(N \log(N))$ .
- To Check one edge if it needs to be in MST or not, we apply Union-find to check if it forms a circle with edges present and add to MST **exactly once** and apply Union-Find algorithm of order  $\log(E)$ .
- Since we perform atmost  $N$  checks for a graph total complexity is  $O(N \log(E))$  for the checkings.
- So, total complexity is  $O(N \log(E) + N \log(N))$

### Best and Worst Cases for Kruskal's

For regular Kruskal's, time complexity will be  $O(N \log(E) + N \log(N))$  in all cases. For Kruskal's :

- In best case scenario, we have  $N$  no cycles and we have to run  $N-1$  iterations to determine MST.
- Time complexity will be  $*O((N-1)\log(E) + E\log(E))$  in this case.
- In worst case we will have to check all  $E$  edges. Time complexity in such case would be  $O(E \log(E) + N \log(N))$

### Space Complexity of Kruskal's

While sorting, we need an extra array to store sorted array of edges (Space complexity of  $O(E)$ ), Another array for Union-Find of size  $O(E)$ . So, total Space Complexity would be  $O(\log(E))$ .

### Kruskal's Algorithm Applications

- In order to layout electrical wiring
- In computer network (LAN connection)

### Conclusion:

We have implemented Kruskal's algorithm to find minimum spanning tree.

### Oral Questions:

1. What problems is Kruskal's algorithm used to solve?
2. What is Kruskal algorithm also called?
3. Which algorithm is best for spanning tree?
4. What is the difference between Kruskal and Prim's algorithm?
5. What are the applications of Prim's and Kruskal algorithm?
6. Which is better Prim's or Kruskal or Dijkstra?

## Assignment No: 03D

---

**Title of the Assignment:** Implement Greedy search algorithm.

**Problem statement:** Implement Selection Sort algorithm.

**Objective:**

- To understand the concept of greedy search algorithm and sorting algorithms.
- To implement Selection sort algorithm.

**Theory:**

**Greedy Method/Technique:**

The greedy method is one of the strategies like Divide and conquer used to solve the problems. This method is used for solving optimization problems. An optimization problem is a problem that demands either maximum or minimum results.

The Greedy method is the simplest and straightforward approach. It is not an algorithm, but it is a technique. The main function of this approach is that the decision is taken on the basis of the currently available information. Whatever the current information is present, the decision is made without worrying about the effect of the current decision in future.

This technique is basically used to determine the feasible solution that may or may not be optimal. The feasible solution is a subset that satisfies the given criteria. The optimal solution is the solution which is the best and the most favorable solution in the subset. In the case of feasible, if more than one solution satisfies the given criteria then those solutions will be considered as the feasible, whereas the optimal solution is the best solution among all the solutions.

**Applications of Greedy Algorithm:**

- It is used in finding the shortest path.
- It is used to find the minimum spanning tree using the prim's algorithm or the Kruskal's algorithm.
- It is used in a job sequencing with a deadline.
- This algorithm is also used to solve the fractional knapsack problem.

"A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum." (Kleinberg and Tardos, Algorithm Design, 2005)

## Selection Sort:

A greedy algorithm is one that makes the locally optimal choice at each step with the hope of finding a global optimum. In selection sort, the algorithm selects the smallest (or largest) element from the unsorted portion of the array and places it at the beginning of the sorted portion of the array. This is done repeatedly until the entire array is sorted.

At each step, selection sort chooses the locally optimal choice (the smallest element) and places it in its correct position in the sorted portion of the array. While this choice may not always lead to the globally optimal solution, it is the best choice based on the information available at that particular step of the algorithm. Selection sort does follow the problem-solving heuristic of making the locally optimal choice at each stage, and this is why it is considered a greedy method.

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

## Procedure for Selection Sort

1. Swap the first element with the smallest element in the array. We have now  $n-1$  elements to sort
2. Find the smallest element in array from array[1] to array [ $n-1$ ] and swap it with the second position
3. For the next element, find the smallest element in the remaining array and swap it.

## Working of Selection Sort

1. Set the first element as minimum

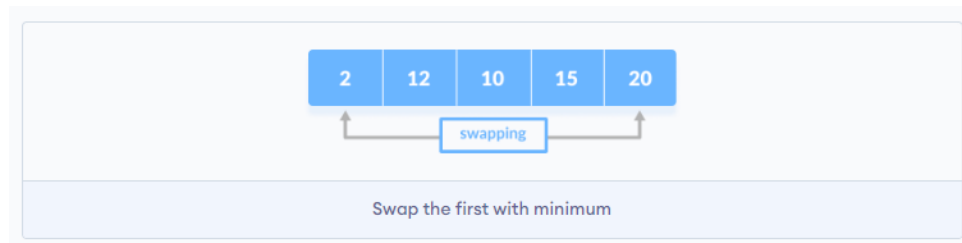


2. Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.

Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing. The process goes on until the last element.

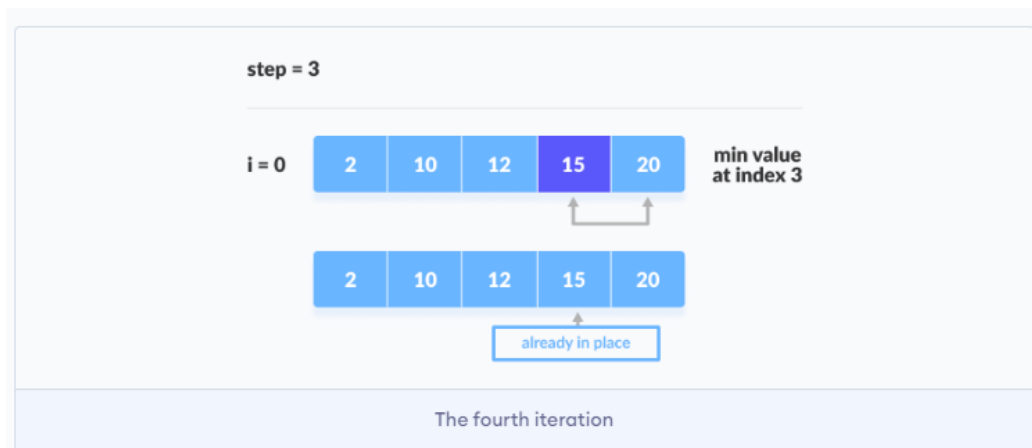
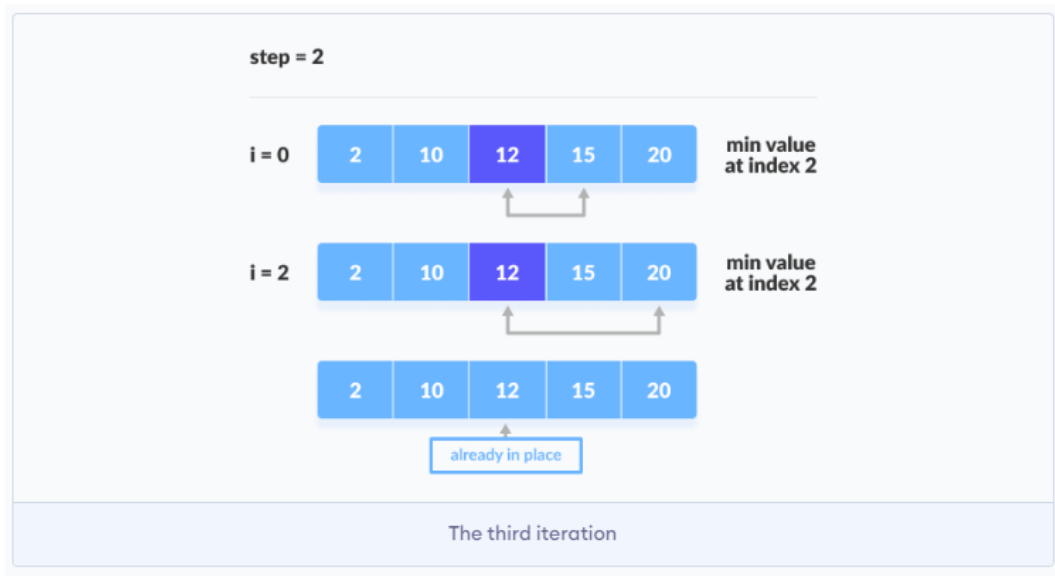


3. After each iteration, **minimum** is placed in the front of the unsorted list.



4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.





## Selection Sort Algorithm

```
selectionSort(array, size)
  repeat (size - 1) times
    set the first unsorted element as the minimum
    for each of the unsorted elements
      if element < currentMinimum
        set element as new minimum
    swap minimum with first unsorted position
end selectionSort
```

### Selection Sort Complexity:

#### Time Complexity

Best  $O(n^2)$

Worst  $O(n^2)$

Average  $O(n^2)$

Space Complexity  $O(1)$

Stability No

#### Conclusion:

We have implemented Selection sort algorithm.

#### Oral Questions:

1. What is selection sort and how does it work?
2. What is the time complexity of selection sort and why?
3. Explain the algorithm for selection sort step-by-step?
4. How is selection sort different from other sorting algorithms, such as bubble sort and insertion sort?
5. Can you provide an example of how selection sort can be used in a real-world application?
6. Is selection sort stable or unstable? Can you explain why?
7. How would you analyze the efficiency of selection sort for large input sizes?

## Assignment No: 04

---

**Title of the Assignment:** Constraint Satisfaction Problem

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

**Objective:**

- To study Constraint Satisfaction Problem using Branch and Bound and Backtracking.
- To implement graph coloring problem using CSP.

**Theory:**

**Constraint Satisfaction Problem:**

A constraint satisfaction problem (CSP) is defined by a set of variables,  $x_1, x_2, \dots, x_n$ , and a set of constraints,  $c_1, c_2, \dots, c_m$ . Each variable has a nonempty domain of possible values. Each constraint involves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables,  $\{x_i=v_i, x_j=v_j, \dots\}$ . An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints.

In constraint satisfaction, domains are the spaces where the variables reside, following the problem specific constraints. These are the three main elements of a constraint satisfaction technique. The constraint value consists of a pair of **{scope, rel}**. The **scope** is a tuple of variables which participate in the constraint and **rel** is a relation which includes a list of values which the variables can take to satisfy the constraints of the problem.

**Graph Coloring:** The problem where the constraint is that no adjacent sides can have the same color.

Graph coloring is the procedure of assignment of colors to each vertex of a graph  $G$  such that no adjacent vertices get same color. The objective is to minimize the number of colors while coloring a graph. The smallest number of colors required to color a graph  $G$  is called its chromatic number of that graph. Graph coloring problem is a NP Complete problem.

**Method to Color a Graph:**

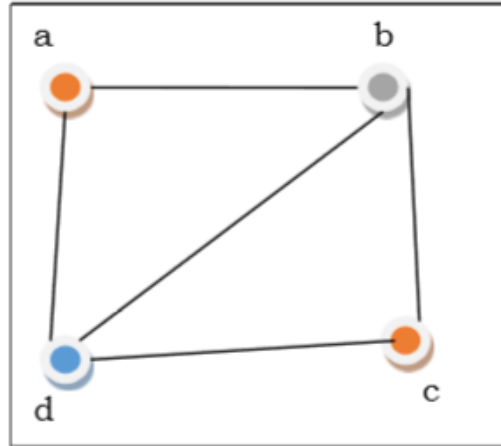
The steps required to color a graph  $G$  with  $n$  number of vertices are as follows –

**Step 1** – Arrange the vertices of the graph in some order.

**Step 2** – Choose the first vertex and color it with the first color.

**Step 3** – Choose the next vertex and color it with the lowest numbered color that has not been colored on any vertices adjacent to it. If all the adjacent vertices are colored with this color, assign a new color to it. Repeat this step until all the vertices are colored.

**Example:**



In the above figure, at first vertex *a* is colored red. As the adjacent vertices of vertex *a* are again adjacent, vertex *b* and vertex *d* are colored with different color, gray and blue respectively. Then vertex *c* is colored as red as no adjacent vertex of *c* is colored red. We could color the graph by 3 colors. Hence, the chromatic number of the graph is 3.

**Pseudocode:**

```
def isSafeToColor(graph, color):
    for i in range(V):
        for j in range(i + 1, V):
            if graph[i][j] == 1 and color[j] == color[i]:
                return False
    return True

def printColorArray(color):
    print("Solution colors are: ")
    for i in range(len(color)):
        print(color[i], end=" ")

def graphColoring(graph, m, i, color):
    if i == V:
        if isSafeToColor(graph, color):
            printColorArray(color)
            return True
        return False
    for j in range(1, m + 1):
        color[i] = j
```



```

    if graphColoring(graph, m, i + 1, color):
        return True
    color[i] = 0
return false

```

**Pseudocode:** Backtracking search

```

function Backtracking-Search(csp) returns solution/failure
    return Recursive-Backtracking({ }, csp)
function Recursive-Backtracking(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← Select-Unassigned-Variable(Variables[csp], assignment, csp)
    for each value in Order-Domain-Values(var, assignment, csp) do
        if value is consistent with assignment given Constraints[csp] then
            add {var = value} to assignment
            result ← Recursive-Backtracking(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure

```

Backtracking = DFS + variable-ordering + fail-on-violation

Backtracking enables us the ability to solve a problem as big as 25-queens

### Conclusion:

Constraint satisfaction problems (CSP)s are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods. CSPs are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyze and solve problems of many unrelated families. CSPs often exhibit high complexity, requiring a combination of heuristics and combinatorial search methods to be solved in a reasonable time.

We have implemented graph coloring problem using CSP.

### Oral questions:

1. Consider a problem of preparing a schedule for a class of student. What type of problem is this?
2. Define Constraint satisfaction problem.
3. What are the variations on CSP formalism?
4. What are the issues that need to be addressed for solving CSP efficiently?
5. Explain Backtracking search in CSP for n queens.
6. Detail the concepts of backtracking and constrain propagation and solve the n-queen problem using these algorithms.

## Assignment No: 05

---

**Title of the Assignment:** Chatbot application

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

**Objective:**

- To understand the Chatbot application.
- To implement an elementary Chatbot application for customer interaction.

**Theory:**

A chatbot is software that simulates human-like conversations with users via text messages on chat. Its key task is to help users by providing answers to their questions. Chatbots are programs built to automatically engage with received messages. Chatbots can be programmed to respond the same way each time, to respond differently to messages containing certain keywords and even to use machine learning to adapt their responses to fit the situation.

Chatbots leverage chat mediums like SMS text, website chat windows and social messaging services across platforms like Facebook and Twitter to receive and respond to messages.

**Why add a chatbot to website?**

When businesses add a chatbot to their support offerings, they're able to serve more customers, improve first response time, and increase agent efficiency. Chatbots help mitigate the high volume of rote questions that come through via email, messaging, and other channels by empowering customers to find answers on their own and guiding them to quick solutions.

When chatbots take simple, repetitive questions off a support team's plate, they give agents time back to provide more meaningful support—nothing kills team productivity like forcing employees to do work that could be automated. Bots can also integrate into global support efforts and ease the need for international hiring and training. They're a cost-effective way to deliver instant support that never sleeps—over the weekends, on holidays, and in every time zone.

**The Value of Chatbots:**

One way to stay competitive in modern business is to automate as many of your processes as possible. Evidence of this is seen in the rise of self-checkout at grocery stores and ordering kiosks at restaurants.

Amazon just opened a store without any cashiers or self-checkouts, limiting human interactions to those only absolutely necessary.

The value in chatbots comes from their ability to automate conversations throughout the organization.

Below are five key benefits businesses realize when using chatbots.

1. Save Time & Money
2. Generate Leads & Revenue
3. Guide Users to Better Outcomes
4. Provide 'After Hours' Support
5. Engage Users in a Unique Way

### Limitations with A Chatbot

With increasing advancements, there also comes a point where it becomes fairly difficult to work with the chatbots. Following are a few limitations we face with the chatbots.

- **Domain Knowledge** – Since true artificial intelligence is still out of reach, it becomes difficult for any chatbot to completely fathom the conversational boundaries when it comes to conversing with a human.
- **Personality** – Not being able to respond correctly and fairly poor comprehension skills has been more than frequent errors of any chatbot, adding a personality to a chatbot is still a benchmark that seems far far away. But we are more than hopeful with the existing innovations and progress-driven approaches.

The chatbots can be defined into two categories; following are the two categories of chatbots:

1. **Rule-Based Approach** – In this approach, a bot is trained according to rules. Based on this a bot can answer simple queries but sometimes fails to answer complex queries.
2. **Self-Learning Approach** – These bots follow the machine learning approach which is rather more efficient and is further divided into two more categories.
  - **Retrieval-Based Models** – In this approach, the bot retrieves the best response from a list of responses according to the user input.
  - **Generative Models** – These models often come up with answers than searching from a set of answers which makes them intelligent bots as well.

### Few applications across Industries:

According to a new survey, 80% of businesses want to integrate chatbots in their business model by 2020. According to a chatbot, these major areas of direct-to-consumer engagement are prime:

### Chatbots in Restaurant and Retail Industries

Famous restaurant chains like Burger King and Taco bell has introduced their Chatbots to stand out of competitors of the Industry as well as treat their customers quickly. Customers of these restaurants are greeted by the resident Chatbots, and are offered the menu options- like a counter

order, the Buyer chooses their pickup location, pays, and gets told when they can head over to grab their food. Chatbots also works to accept table reservations, take special requests and go take the extra step to make the evening special for your guests. Chatbots are not only good for the restaurant staff in reducing work and pain but can provide a better user experience for the customers.

### **Chatbots in Hospitality and Travel**

For hoteliers, automation has been held up as a solution for all difficulties related to productivity issues, labour costs, a way to ensure consistently, streamlined production processes across the system. Accurate and immediate delivery of information to customers is a major factor in running a successful online Business, especially in the price sensitive and competitive Travel and Hospitality industry. Chatbots particularly have gotten a lot of attention from the hospitality industry in recent months.

### **Chatbots in Health Industry**

Chatbots are a much better fit for patient engagement than Standalone apps. Through these Health-Bots, users can ask health related questions and receive immediate responses. These responses are either original or based on responses to similar questions in the database. The impersonal nature of a bot could act as a benefit in certain situations, where an actual Doctor is not needed. Chatbots ease the access to healthcare and industry has favourable chances to serve their customers with personalised health tips. It can be a good example of the success of Chatbots and Service Industry combo.

### **Chatbots in E-Commerce**

Mobile messengers- connected with Chatbots and the E-commerce business can open a new channel for selling the products online. E-commerce Shopping destination “Spring” was the early adopter. E-commerce future is where brands have their own Chatbots which can interact with their customers through their apps.

### **Chatbots in Fashion Industry**

Chatbots, AI and Machine Learning pave a new domain of possibilities in the Fashion industry, from Data Analytics to Personal Chatbot Stylists. Fashion is such an industry where luxury goods can only be bought in a few physical boutiques and one to one customer service is essential. The Internet changed this dramatically, by giving the customers a seamless but a very impersonal experience of shopping. This particular problem can be solved by Chatbots. Customers can be treated personally with bots, which can exchange messages, give required suggestions and information. Famous fashion brands like Burberry, Tommy Hilfiger have recently launched Chatbots for the London and New York Fashion Week respectively. Sephora a famous cosmetics brand and H&M– a fashion clothing brand have also launched their Chatbots.

## Chatbots in Finance

Chatbots have already stepped in Finance Industry. Chatbots can be programmed to assist the customers as Financial Advisor, Expense Saving Bot, Banking Bots, Tax bots, etc. Banks and Fintech have ample opportunities in developing bots for reducing their costs as well as human errors. Chatbots can work for customer's convenience, managing multiple accounts, directly checking their bank balance and expenses on particular things. Further about Finance and Chatbots have been discussed in our earlier blog: Chatbots as your Personal Finance Assistant.

## Chatbots in Fitness Industry

Chat based health and fitness companies using Chatbot, to help their customers get personalised health and fitness tips. Tech based fitness companies can have a huge opportunity by developing their own Chatbots offering huge customer base with personalised services. Engage with your fans like never before with news, highlights, game-day info, roster and more. Chatbots and Service Industry together have a wide range of opportunities and small to big all size of companies using chatbots to reduce their work and help their customers better.

## Chatbots in Media

Big publisher or small agency, our suite of tools can help your audience chatbot experience rich and frictionless. Famous News and Media companies like The Wall Street Journal, CNN, Fox news, etc have launched their bots to help you receive the latest news on the go.

## Chatbot in Celebrity

With a chatbot you can now have one-on-one conversation with millions of fans.

## Languages and technologies:

Good knowledge of back-end technologies and analytics.

Languages used for developing chatbots are Java, C#, Python, and Node JS.

To be able to answer arbitrary questions and to develop these smart robots, a deep understanding of machine learning, artificial intelligence, Natural Language Understanding (NLU), and Google Cloud Natural Language API (Application Programming Interface) is required.

## Sample Chatbot program:

```
def greet(bot_name, birth_year):
    print("Hello! My name is {0}.".format(bot_name))
    print("I was created in {0}.".format(birth_year))

def remind_name():
    print('Please, remind me your name.')
```

```

name = input()
print("What a great name you have, {0}!".format(name))

def guess_age():
    print('Let me guess your age.')
    print('Enter remainders of dividing your age by 3, 5 and 7.')

    rem3 = int(input())
    rem5 = int(input())
    rem7 = int(input())
    age = (rem3 * 70 + rem5 * 21 + rem7 * 15) % 105

    print("Your age is {0}; that's a good time to start
programming!".format(age))

def count():
    print('Now I will prove to you that I can count to any number you
want.')
    num = int(input())

    counter = 0
    while counter <= num:
        print("{0} !".format(counter))
        counter += 1

def test():
    print("Let's test your programming knowledge.")
    print("Why do we use methods?")
    print("1. To repeat a statement multiple times.")
    print("2. To decompose a program into several small subroutines.")
    print("3. To determine the execution time of a program.")
    print("4. To interrupt the execution of a program.")

    answer = 2
    guess = int(input())
    while guess != answer:
        print("Please, try again.")
        guess = int(input())

    print('Completed, have a nice day!')
    print('.....')
    print('.....')
    print('.....')

def end():
    print('Congratulations, have a nice day!')
    print('.....')
    print('.....')
    print('.....')
    input()

```

```
greet('Sbot', '2021') # change it as you need
remind_name()
guess_age()
count()
test()
end()
```

**Output:**

Hello! My name is Sbot.  
I was created in 2021.  
Please, remind me your name.

**Conclusion:**

We have understood concept of chat bot and implemented an elementary chatbot application for customer interaction.

**Oral questions:**

1. What is a chatbot?
2. How does the chatbot understand what the customer is trying to convey?
3. How does the bot know when it needs to converse with a human?
4. Can the bot be controlled once it is live?
5. What are the key benefits of chatbots to business applications?
6. What languages and technologies should a chatbot developer be well-versed in to build chatbots?
7. Will chatbots replace mobile apps in the future?
8. What are some of the most popular companies using chatbots?

## Assignment No: 06

---

**Title of the Assignment:** Expert System

**Problem statement:** Implement any one of the following Expert System for Hospitals and medical facilities

**Objective:**

- To understand the concept of expert systems and AI.
- To implement an expert system for disease diagnosis.

**Theory:**

An expert system, also known as a knowledge based system, is a computer program that contains the knowledge and analytical skills of one or more human experts, related to a specific subject. Thus an expert system asks you questions until it can identify an object that makes your answers. More generally an expert system attempts to advice its user its subject of expertise. This system is used to diagnose a disease and also give preliminary medical prescription to the patient. It accepts symptoms from the user and generates the diagnosis.

Typically, an expert system incorporates a knowledge base containing accumulated experience and an inference or rules engine -- a set of rules for applying the knowledge base to each particular situation that is described to the program. The system's capabilities can be enhanced with additions to the knowledge base or to the set of rules. Current systems may include machine learning capabilities that allow them to improve their performance based on experience, just as humans do.

The concept of expert systems was first developed in the 1970s by Edward Feigenbaum, professor and founder of the Knowledge Systems Laboratory at Stanford University. Feigenbaum explained that the world was moving from data processing to "knowledge processing," a transition which was being enabled by new processor technology and computer architectures.

Expert systems have played a large role in many industries including in financial services, telecommunications, healthcare, customer service, transportation, video games, manufacturing, aviation and written communication. Two early expert systems broke ground in the healthcare space for medical diagnoses: Dendral, which helped chemists identify organic molecules, and MYCIN, which helped to identify bacteria such as bacteremia and meningitis, and to recommend antibiotics and dosages.



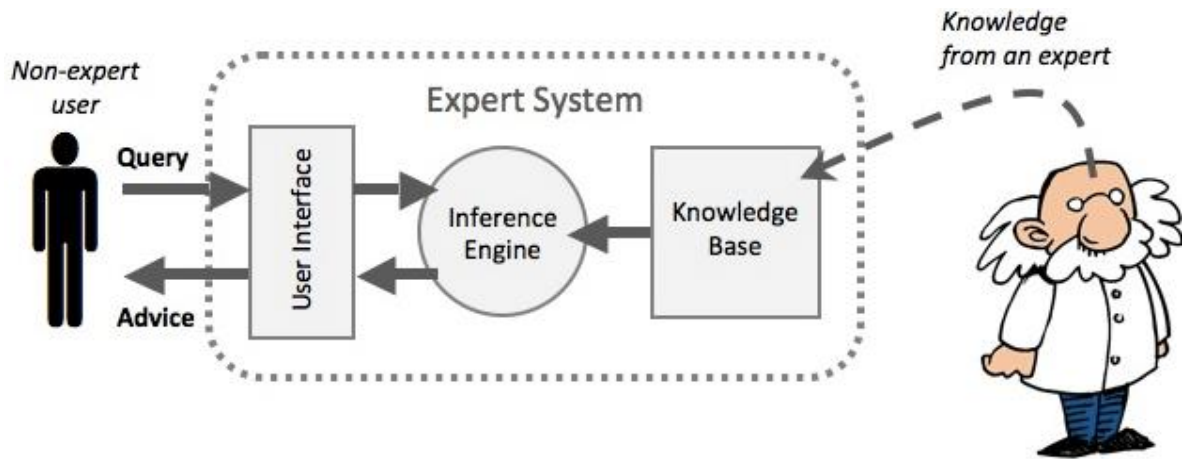


Figure: Components of Expert System

## Components of Expert Systems:

### 1. User Interface (UI)

With the assistance of a UI, the master framework communicates with the client, accepts inquiries as a contribution to a clear arrangement, and passes it to the deduction motor.

In the wake of getting the reaction from the deduction motor, it shows the yield to the client. As it were, it is an interface that helps a non-master client to speak with the master framework to discover an answer.

It takes the client's inquiry in a coherent structure and forwards it to the inference engine. From that point onward, it shows the outcomes to the client, as such, it's an interface that enables the client to speak with the master framework.

The (UI) is the space that encourages correspondence between the framework and its clients. It's synonymous with your PC work area or cell phone home screen.

### 2. Inference Engine

Inference Engine is the mind behind the UI. It contains a predefined set of rules to tackle a particular issue and alludes to the information from the Knowledge Base.

It chooses realities and rules to apply when attempting to answer the client's inquiry. Inference Engine gives thinking about the data in the information base.

It likewise helps in deducting the issue to discover the arrangement. This part is additionally useful for detailing ends.

## The two basic strategies used in inference engines are:

### a. *Forward Chaining*

This strategy used to determine the probable outcome in the future. With the given inputs and conditions, this strategy utilizes expert systems to find out the probable outcome. This helps to extract data till a particular goal is reached.

### b. *Backward Chaining*

This strategy used to determine why a particular event take would happen with the current circumstances provided. It is utilized in automated theorem provers, inference engines, proof assistants, and other AI applications.

## 3. Knowledge Base

It is where data contributed by specialists from the required domains is put away. Think about an information base as a book or an article. To make entries from a book sound, you need to refer to data from specialists to make it more credible. It contains both factual and heuristic knowledge.

## Characteristics of Expert Systems in AI:

### 1. High performance

The first and foremost characteristic of an expert system is to deliver high performance 24×7

### 2. Understandable

The expert system should be easy to comprehend for all the people using it.

### 3. Reliable

An expert system has to be reliable in the sense that it is error-free so that it is trustable.

### 4. Highly Responsive

An expert system has to be proactive and provide responses for each and every detail of the problem.

## Advantages:

- Provides consistent answers for repetitive decisions, processes and tasks
- Holds and maintains significant levels of information
- Encourages organizations to clarify the logic of their decision-making
- Never "forgets" to ask a question, as a human might

## Limitations:

- Lacks common sense needed in some decision making
- Cannot make creative responses as human expert would in unusual circumstances

- Domain experts not always able to explain their logic and reasoning
- Errors may occur in the knowledge base, and lead to wrong decisions
- Cannot adapt to changing environments, unless knowledge base is changed

### **Applications of Expert Systems:**

- Expert systems are being used in designing and manufacturing domain for the production of vehicles and gadgets like cameras.
- In the knowledge domain, Expert Systems are used for delivering the required knowledge to the client. The knowledge can be legal advice, tax advice, or something other than that.
- In the banking and finance sector, expert systems are widely used for the detection of frauds.
- Expert Systems can also use in the diagnosis and troubleshooting of medical equipment.
- Apart from this, Expert Systems can also have use cases in Planning and Scheduling tasks.

### **Languages and technologies:**

Expert systems have been constructed using various general-purpose programming languages as well as specific tools. LISP and PROLOG have been used widely.

SWI-Prolog offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications.

Python and Java programming languages are also used along with ML.

### **Sample program: The animal identification game (simple expert system)**

```
go :- hypothesize(Animal),
    write('I guess that the animal is: '),
    write(Animal),
    nl,
    undo.

/* hypotheses to be tested */
hypothesize(cheetah)    :- cheetah, !.
hypothesize(tiger)     :- tiger, !.
hypothesize(giraffe)   :- giraffe, !.
hypothesize(zebra)     :- zebra, !.
hypothesize(ostrich)   :- ostrich, !.
hypothesize(penguin)   :- penguin, !.
hypothesize(albatross) :- albatross, !.
hypothesize(unknown) . /* no diagnosis */

/* animal identification rules */
cheetah :- mammal,
```

```

        carnivore,
        verify(has_tawny_color),
        verify(has_dark_spots).
tiger :- mammal,
        carnivore,
        verify(has_tawny_color),
        verify(has_black_stripes).
giraffe :- ungulate,
        verify(has_long_neck),
        verify(has_long_legs).
zebra :- ungulate,
        verify(has_black_stripes).

ostrich :- bird,
        verify(does_not_fly),
        verify(has_long_neck).
penguin :- bird,
        verify(does_not_fly),
        verify(swims),
        verify(is_black_and_white).
albatross :- bird,
        verify(appears_in_story_Ancient_Mariner),
        verify(flys_well).

/* classification rules */
mammal    :- verify(has_hair), !.
mammal    :- verify(gives_milk).
bird      :- verify(has_feathers), !.
bird      :- verify(flys),
        verify(lays_eggs).
carnivore :- verify(eats_meat), !.
carnivore :- verify(has_pointed_teeth),
        verify(has_claws),
        verify(has_forward_eyes).
ungulate  :- mammal,
        verify(has_hooves), !.
ungulate  :- mammal,
        verify(chews_cud).

/* how to ask questions */
ask(Question) :-
    write('Does the animal have the following attribute: '),
    write(Question),
    write('? '),
    read(Response),
    nl,
    ( (Response == yes ; Response == y)
      ->
        assert(yes(Question)) ;
        assert(no(Question)), fail).

:- dynamic yes/1,no/1.

/* How to verify something */
verify(S) :-
    (yes(S)
    ->

```

```

    true ;
    (no(S)
    ->
    fail ;
    ask(S))).

/* undo all yes/no assertions */
undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.

```

## Output:

```

SWI-Prolog (Multi-threaded, version 8.4.2)
File Edit Settings Run Debug Help
Warning: Previously defined at c:/users/administrator/appdata/local/temp/xpcs2:121
?- go.
Does the animal have the following attribute: has_hair? y.
Does the animal have the following attribute: eats_meat? |: y.
Does the animal have the following attribute: has_tawny_color? |: n.
Does the animal have the following attribute: has_hooves? |: n.
Does the animal have the following attribute: chews_cud? |: n.
Does the animal have the following attribute: has_feathers? |: y.
Does the animal have the following attribute: does_not_fly? |: y.
Does the animal have the following attribute: has_long_neck? |: n.
Does the animal have the following attribute: swims? |: n.
Does the animal have the following attribute: appears_in_story_Ancient_Mariner? Does the animal have the following attribute: appears_in_story_Ancient_Mariner?n.
I guess that the animal is: unknown
true.
?-

```

## Conclusion:

We have understood the concept of Expert system and implemented an expert system for Hospitals and medical facilities.

**Oral questions:**

1. What is an expert system?
2. What are the different stages/components/blocks of expert system?
3. What is meant by a knowledge base?
4. How a knowledge base is different from a database?
5. What is an inference engine?
6. Whether an expert system can replace human experts? Yes/No
7. What is an expert shell?
8. What are the desirable properties of a good expert system?
9. What is MYCIN?
10. What is Prolog?
11. What Are The Features Of Prolog Language?

**Reference:**

<https://www.swi-prolog.org/>

[https://www.cpp.edu/~jrfisher/www/prolog\\_tutorial/pt\\_framer.html](https://www.cpp.edu/~jrfisher/www/prolog_tutorial/pt_framer.html)

## Add on Practical Assignments

### Assignment 1

**Title:** Constraint Satisfaction Problem

**Objective:** Implementation of Constraint Satisfaction Problem for solving Crypt-arithmetic Problems.

**Theory:**

**Constraint Satisfaction Problem** - A constraint satisfaction problem (CSP) is defined by a set of variables,  $x_1, x_2, \dots, x_n$ , and a set of constraints,  $c_1, c_2, \dots, c_m$ . Each variable has a nonempty domain of possible values. Each constraint involves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables,  $\{x_i=v_i, x_j=v_j, \dots\}$ . An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints.

**Crypt-arithmetic**– Crypt-arithmetic is a CSP problem in which letters are substituted by digits such that each letter represents a unique digit, and the actual problem is to find a proper sequence of digits assigned to different letters satisfying the conditions of the arithmetic operation. One of the well-known instances of this problem is shown in Fig 1.

$$\begin{array}{r} \text{SEND} \\ + \text{MORE} \\ \hline \text{MONEY} \end{array}$$

Figure 1. An example of Cryptarithmic problem.

Assigning digits to letters in following way would be an acceptable solution.

O=0, M=1, Y=2, E=5, N=6, D=7, R=8 and S=9.

Hence, the result would be as shown in Fig. 2.

$$\begin{array}{r} 9567 \\ + 1085 \\ \hline 10652 \end{array}$$

Figure. 2. The solution to problem illustrated in Fig.1.

Constraints of the problem are as follows:

1. Each letter should be bound to a unique digit and no two different letters could be bounded to the same digit.

2. As strings are representing numbers the first letter of strings could not be assigned to zero.
3. The resulting numbers should satisfy the problem meaning that the result of the two first numbers (operands) under the specified arithmetic operation (plus operator) should be the third number.

Examples of simple problems that can be modeled as a constraint satisfaction problem:

- Eight queens puzzles
- Map coloring problem
- Sudoku

### **Conclusion:**

Constraint satisfaction problems (CSP)s are mathematical problems defined as a set of objects whose state must satisfy a number of *constraints* or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods. CSPs are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyze and solve problems of many unrelated families. CSPs often exhibit high complexity, requiring a combination of heuristics and combinatorial search methods to be solved in a reasonable time.



## Assignment 2

**Title:** MinMax Search Procedure with alpha beta pruning for finding the solutions of games.

**Objective:** Implementation of MinMax Search Procedure with alpha beta pruning for finding the solutions of games. (TIC-TAC-TOE)

### Theory:

**Minimax** (sometimes **minmax**) is a decision rule used in decision theory, game theory, statistics and philosophy for *minimizing* the possible loss while *maximizing* the potential gain. Alternatively, it can be thought of as maximizing the minimum gain (**maximin**). Originally formulated for two-player sum game, covering both the cases where players take alternate moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision making in the presence of uncertainty.

A **minimax algorithm** is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is A's turn to move, A gives a value to each of his legal moves.

A simple version of the minimax *algorithm*, stated below, deals with games such as tic-tac-toe, where each player can win, lose, or draw. If player A *can* win in one move, his best move is that winning move. If player B knows that one move will lead to the situation where player A *can* win in one move, while another move will lead to the situation where player A can, at best, draw, then player B's best move is the one leading to a draw. Late in the game, it's easy to see what the "best" move is. The Minimax algorithm helps find the best move, by working backwards from the end of the game. At each step it assumes that player A is trying to **maximize** the chances of A winning, while on the next turn player B is trying to **minimize** the chances of A winning (i.e., to maximize B's own chances of winning).

### Alpha-beta pruning:

**Alpha-beta pruning** is a search algorithm which seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. *When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.*

### Pseudo code:

**functional**alphabeta(node, depth,  $\alpha$ ,  $\beta$ , Player)

```

if depth = 0 or node is a terminal node
return the heuristic value of node
if Player = MaxPlayer
for each child of node
 $\alpha := \max(\alpha, \text{alphabeta}(\text{child}, \text{depth}-1, \alpha, \beta, \text{not}(\text{Player})))$ 
if  $\beta \leq \alpha$ 
break(* Beta cut-off *)
return  $\alpha$ 
else
for each child of node
 $\beta := \min(\beta, \text{alphabeta}(\text{child}, \text{depth}-1, \alpha, \beta, \text{not}(\text{Player})))$ 
if  $\beta \leq \alpha$ 
break(* Alpha cut-off *)
return  $\beta$ 
(* Initial call *)
alphabeta(origin, depth, -infinity, +infinity, MaxPlayer)

```

### Tic - Tac – Toe:

**Tic-tac-toe**, also rendered **wick wack woe** (in some Asian countries), or **noughts and crosses/Xs and Os** as it is known in the UK, Australia and New Zealand, is a pencil-and-paper game for two players, **X** and **O**, who take turns marking the spaces in a 3×3 grid. The **X** player usually goes first. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game.

The following example game is won by the first player, X:

		X	O		X	O		X	O		X	O		X	O		X
								O			O			O			O
			X			X			X		X	X		X	X		X

The simplicity of tic-tac-toe makes it ideal as a pedagogical tool for teaching the concepts of combinatorial game theory and the branch of artificial intelligence that deals with the searching of game trees.

### Conclusion:

The performance of the naïve minimax algorithm may be improved dramatically, without affecting the result, by the use of alpha-beta pruning. Other heuristic pruning methods can also be used, but not all of them are guaranteed to give the same result as the un-pruned search.

## Assignment 3

**Title:** Implementation of parsing methods to categorize the text.

**Objective:** Prolog grammar for simple English phrase structures.

### Theory:

Prolog has the capacity to load definite clause grammar rules (DCG rules) and automatically convert them to Prolog parsing rules. We consider the following little grammar for a fragment of English, the boy who sits reads a book.

```
s --> np, vp.                /* sentence */

np --> pn.                    /* noun phrase */
np --> d, n, rel.

vp --> tv, np.                /* verb phrase */
vp --> iv.

rel --> [].                   /* relative clause */
rel --> rpn, vp.

pn --> [PN], {pn(PN)}.        /* proper noun */
pn(mary).
pn(henry).

rpn --> [RPN], {rpn(RPN)}.    /* relative pronoun */
rpn(that).
rpn(which).
rpn(who).

iv --> [IV], {iv(IV)}.        /* intransitive verb */
iv(runs).
iv(sits).

d --> [DET], {d(DET)}.        /* determiner */
d(a).
d(the).

n --> [N], {n(N)}.            /* noun */
n(book).
n(girl).
n(boy).

tv --> [TV], {tv(TV)}.        /* transitive verb */
tv(gives).
tv(reads).
```

The grammar most likely looks very familiar in many respects. We have both grammar rules (formed with '-->') and ordinary Prolog rules (formed with ':'). This file is loaded into Prolog in the usual way, and the grammar rules are converted to parsing rules.

```
?- s([the,boy,who,sits,reads,a,book],[ ]).
```

```

yes

?- s([henry,reads],[ ]).
no

?- listing([np,d]).

np(A,B) :-
    pn(A,B) .

np(A,B) :-
    d(A,C) ,
    n(C,D) ,
    rel(D,B) .

d(A,B) :-
    'C'(DET,A,B) ,
    d(DET) .

d(a) .
d(the) .

yes

```

The first goal leads to a successful parse, whereas the second, although it is a good English sentence, does not conform to the grammar as given -- the grammar requires transitive verbs to have objects. The last two goals request the internal Prolog clause definitions for 'np' and for 'd'. For example, the correspondence between the second 'np' grammar rule and the Prolog parse rule looks like this when the two are aligned on the page:

```

np          --> det,          noun,          rel.

np(A,B) :- det(A,C) , noun(C,D) , rel(D,B) .

```

Doing the same for the 'd' category we get:

```

d          --> [DET] ,          {d(DET)} .

det(A,B)  :- 'C'(DET,A,B) , d(DET) .

```

The 'd' in the head of the grammar rule is a grammar category(determiner), whereas the 'd' in the body of the grammar rule is encapsulated inside braces { } and is therefore a Prolog literal (an embedded Prolog goal). Terminal data in the grammar database, such as 'n(boy).', already have their Prolog form so do not need to be translated. The 'C' predicate is built-in; its definition is effectively given by the clause:

```
'C'(X, [X|R], R) .
```

Thus, 'C' means that X "connects" all of a list [X,...] with its tail.

A third kind of grammar rule is like the first one for 'rel'. Such a rule specifies an optional grammatical construction.

```
rel --> [ ] .
```

```
rel(A,A).
```

A successful parse for the goal

```
?- np([the,boy,who,sits],[])
```

will unify this main goal with the head of the second 'np' Prolog rule. Here is a clause tree showing how the list of words is parsed or analyzed from left to right by the appropriate grammar rules.

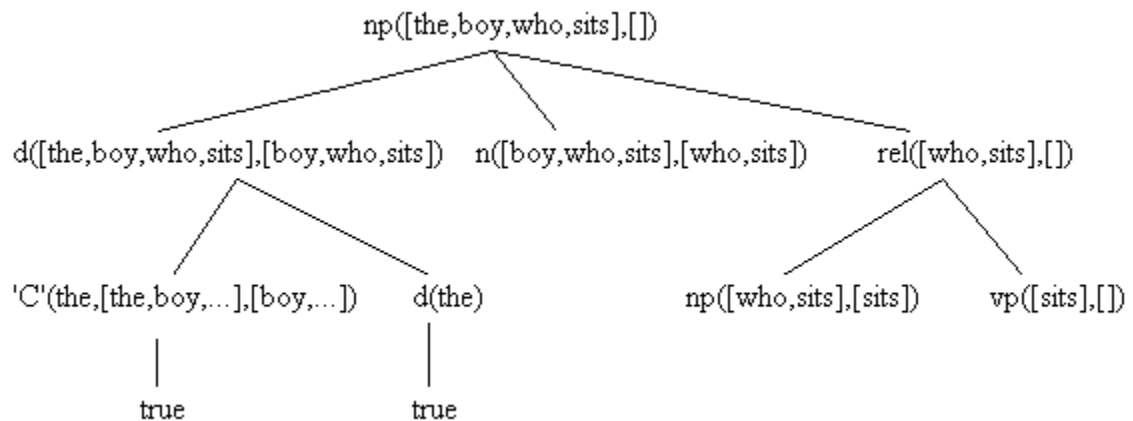


Fig.1

The program clause tree gives a visual representation to how the variables in the Prolog parsing rules are supposed to match initial portions of the remaining input that is being parsed. If we inspect a relevant instance of the 'det' rule we could have the following:

DCG rules can contain arguments, using auxiliary variables. The creation of parse trees is one use for auxiliary variables. Another use is to force number agreement for subject and verb.

### Program:

#### Eng.pro

```

s(s(NP,VP)) --> np(Num,NP), vp(Num,VP).

np(Num,np(PN)) --> pn(Num,PN).
np(Num,NP) -->
    d(Det),
    n(Num,N),
    rel(Num,Rel),
    {build_np(Det,N,Rel,NP)}. /* embedded Prolog goal */

/* Prolog rules for build_np */
build_np(Det,N,rel(nil),np(Det,N)).
build_np(Det,N,rel(RP,VP),np(Det,N,rel(RP,VP))).
  
```

```

vp (Num, vp (TV, NP) ) -->
    tv (Num, TV) ,
    np ( _ , NP) .
vp (Num, vp (IV) ) --> iv (Num, IV) .

rel ( _Num, rel (nil) ) --> [] .
rel (Num, rel (RP, VP) ) -->
    rp (RP) , vp (Num, VP) .

pn (sing, pn (PN) ) --> [PN] , {pn (PN, _X) } .
pn (plu, pn (PN) ) --> [PN] , {pn ( _X, PN) } .
pn (mary, marys) .
pn (henry, henrys) .

rpn (rpn (RPN) ) --> [RPN] , {rpn (RPN) } .
rpn (that) .
rpn (which) .
rpn (who) .

iv (sing, iv (IV) ) --> [IV] , {iv (IV, _X) } .
iv (plu, iv (IV) ) --> [IV] , {iv ( _X, IV) } .
iv (runs, run) .
iv (sits, sit) .

d (d (DET) ) --> [DET] , {d (DET) } .
d (a) .
d (the) .

n (sing, n (N) ) --> [N] , {n (N, _X) } .
n (plu, n (N) ) --> [N] , {n ( _X, N) } .
n (book, books) .
n (girl, girls) .
n (boy, boys) .

tv (sing, tv (TV) ) --> [TV] , {tv (TV, _X) } .
tv (plu, tv (TV) ) --> [TV] , {tv ( _X, TV) } .
tv (gives, give) .
tv (reads, read) .

:- ['read_line'].

parse :- write('Enter English input: '),
    read_line(Input),
    trim_period(Input, I),
    nl,
    s(Parse_form, I, []),
    write(Parse_form),
    nl, nl.

trim_period([], []).
trim_period([X|R], [X|T]) :- trim_period(R, T).

```

### **read\_line.pl**

```

Examples:
% read_line(L).

```

```

The sky was blue, after the rain.
L = [the,sky,was,blue,',','after,the,rain, '.']
% read_line(L).
Which way to the beach?
L = [which,way,to,the, beach, '?']

read_line(Words) :- get0(C),
                    read_rest(C,Words).

/* A period or question mark ends the input. */
read_rest(46,['.']) :- !.
read_rest(63,['?']) :- !.

/* Spaces and newlines between words are ignored. */
read_rest(C,Words) :- ( C=32 ; C=10 ) , !,
                      get0(C1),
                      read_rest(C1,Words).

/* Commas between words are absorbed. */
read_rest(44,[' '|Words]) :- !,
                          get0(C1),
                          read_rest(C1,Words).

/* Otherwise get all of the next word. */
read_rest(C,[Word|Words]) :- lower_case(C,LC),
                             read_word(LC,Chars,Next),
                             name(Word,Chars),
                             read_rest(Next,Words).

/* Space, comma, newline, period or question mark separate words. */
read_word(C,[],C) :- ( C=32 ; C=44 ; C=10 ;
                     C=46 ; C=63 ) , !.

/* Otherwise, get characters, convert alpha to lower case. */
read_word(C,[LC|Chars],Last) :- lower_case(C,LC),
                                get0(Next),
                                read_word(Next,Chars,Last).

/* Convert to lower case if necessary. */
lower_case(C,C) :- ( C < 65 ; C > 90 ) , !.
lower_case(C,LC) :- LC is C + 32.

/* for reference ...
newline(10).
comma(44).
space(32).
period(46).
question_mark(63).
*/

```

**?-[‘Eng.pro’].**

**?- parse.**

Enter English input: The boy who sits reads the book.

```
s(np(d(the),n(boy),rel(rpn(who),vp(iv(sits))))),vp(tv(reads),np(d(a),n(book))))
```

yes

**Reference:**

Pereira, Fernando C. N., and Stuart M. Shieber. 1987. *Prolog and natural-language analysis*.



## Assignment 4

**Title:** Unification Algorithm

**Objective:** Implementation of Unification algorithm by considering Resolution concept.

**Theory:**

**The Unification Algorithm:**

In propositional logic it is easy to determine that two literals cannot be true at the same time. Simply look for L & not L. In predicate logic, this matching process is more complicated since the arguments of the predicates most is considered. For example, man (John) and not man (John) is a contradiction. Thus, in order to determine contradiction, we need a matching procedure that compares 2 literals and discovers whether there exists a set of substitutions that makes them identical. There is a straight forward recursive procedure called the unification algorithm that does this job.

The basic idea of unification is simple. To attempt to unify two literals, we first check if their initial predicate symbols are same. If so, we can proceed otherwise there is no way they can be unified, regardless of their arguments.

**Example:**

try assassinate (M, C)

hate (M, C)

cannot be unified.

If predicate symbols match, then we must check the arguments one pair at a time. If the first matches, we can continue with the second and so on. To test each argument pair, we can simply call the unification procedure recursively.

The only complication in this procedure is that we must find a single, consistent substitution for the entire literal, not separate ones for each piece of it. To do this, we must take each substitution that we find and apply it to the remainder of the literals before we continue trying to unify them.

**Example:** Suppose we want to unify expressions:

$P(x, x)$

$P(y, z)$

The two instances of P match fine. Next we compare x and y and decide that if we substitute y for x they could match. We will write that substitution as:  $y/x$ .

But now, if we simply continue and match x and z we produce the substitution  $z/x$ . But cannot substitute both y and z for x.

The object of unification procedure is to discover at least one substitution that causes two literals to match. Usually if there is one such substitution there are many.

**For example:** The literals:

hate (x, y)

hate (Marcus, z)

could be unified with any of the following substitutions:

(Marcus/x, x/y)

(Marcus/x, y/z)

(Marcus/x, Caser/y, Caser/z)

(Marcus/x, Polonius/y, Polonius/z)

The first two of these are equivalent except for lexical variation. But the second two, although they produce a match, also produce a substitution that is more restrictive than absolutely necessary for the match. Because the final substitution produced by the unification process will be used by the resolution procedure, it is useful to generate the most general unifier possible.

### **Algorithm:**

Unify (L1, L2)

1. If L1 or L2 are both variables or constants then:
  - a. If L1 and L2 are identical return NIL.
  - b. Else if L1 is variable, then if L1 occurs in L2 then return {FAIL} else return (L2/L1).
  - c. Else if L2 is variable then if L2 occurs in L1 then return {FAIL} else return (L1/L2).
  - d. Else return {FAIL}.
2. If the initial predicate symbols in L1 and L2 are not identical then return {FAIL}.
3. If L1 and L2 have a different number of arguments the return {FAIL}.
4. Set SUBST to NIL.
5. For i <- 1 to number of arguments in L1:
  - a. Calls unify with the i<sup>th</sup> argument of L1 and the i<sup>th</sup> arguments of L2 putting result in S.
  - b. If S contains FAIL, then return {FAIL}.

c. If S is not equal to NIL then:

- i) Apply S to the remainder of both L1 and L2.
- ii) SUBST=APPEND (S, SUBST)

6. Return SUBST.

### **Resolution Using Unification Algorithm:**

#### **What is Resolution?**

Resolution is a proof procedure that produces proof by refutation. That is, it tries to prove the validity of a statement by showing that the negation of the statement produces a contradiction with the already known statements.

However Resolution doesn't work directly with predicate logic statements, rather it uses a more flatter and simpler form known as the "Clause Form".

#### **How does Resolution work?**

The resolution procedure is an iterative process, which compares two clauses and yields a new clause that has been inferred from them. Suppose there are two clauses in the system:

1. **winter  $\vee$  summer** ( $\vee$  - or)

2.  **$\sim$ winter  $\vee$  cold**

Now both of these clauses are true. However, only one of winter and  $\sim$ winter is true at any given point in time. If winter is true, that guarantees the truth of cold in the second clause. If  $\sim$ winter is true, it guarantees the truth of summer in the first clause. Hence from these 2 clauses we can derive that

**summer  $\vee$  cold**

Hence what Resolution essentially does is look for the same literal, in 2 clauses, positive in one, and negative in another. For e.g. in the above example, winter is the literal we are looking for. The resultant is obtained by combining all of the literals of the two parent clauses except the ones that cancel. If a contradiction exists, then the resultant will be an empty clause and the procedure will terminate.

### **Resolution Algorithm**

1. Convert all the statements of F to clause form
2. negate S and convert the result to clause form. Add it to the set of clauses obtained in 1.
3. Repeat until either a contradiction is found, no progress can be made

- a. Select two clauses. Call these the parent clauses
- b. Resolve them together. The resolvent will be the disjunction of all of the literals of both of the parent clauses with appropriate substitutions performed and with the following exception. If there is a pair of literals  $T1$  and  $\sim T2$  such that one of the parent clauses contains  $T1$  and the other contains  $T2$  and if  $T1$  and  $T2$  are unifiable, then neither  $T1$  nor  $T2$  should appear in the resolvent. We will call  $T1$  and  $T2$  complementary literals. Use the substitution produced by the unification to create the resolvent.
- c. If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Predicate logic complicates matters as there are variables involved. Hence we cannot directly cancel out the same literals. The variables of the literal in question have to be bound to the same constant. This is what Unification Algorithm tries to achieve.

For the purpose of understanding the Unification Algorithm consider that each predicate logic statement is a list, where the first element is the name of the predicate and the remaining elements are the arguments each of which is either a single element or it is another list. For e.g.

( tryassasinate Marcus Caesar )  $\rightarrow$  This has all the elements as atoms

( tryassasinate Marcus (rulerof Rome) )  $\rightarrow$  This has the last element as another list.

### The Unification Algorithm

1. Compare the first elements ( predicates ) of the two statements. If they are not the same then there is no way of unifying the 2 literals, so exit with failure.
2. If the first elements are the same then we have to unify the rest of the literals moving from the second, to the third and so on. Unification is simple; we just have to keep calling the unification procedure recursively. The matching rules are as follows:
  - a) Different constants, functions or predicates cannot match, identical ones can.
  - b) A variable can match another variable, any constant, function or predicate expression with the restriction that the function or predicate expression must not contain any instances of the variable being matched.
  - c) Once we make a substitution, we need to apply it to the remaining literal before calling the unification algorithm again.

This unification algorithm is one step in the entire Resolution algorithm.

### Conclusion

Unification has deep mathematical roots and is a useful operation in many AI programs for e.g. Theorem proving and natural language processing. And as a result efficient data structure and algorithm for the unification have been developed.

## References

### Text Books:

1. Stuart Russell and Peter Norvig, “Artificial Intelligence: A Modern Approach”, Third edition, Pearson, 2003, ISBN :10: 0136042597
2. Deepak Khemani, “A First Course in Artificial Intelligence”, McGraw Hill Education(India), 2013, ISBN : 978-1-25-902998-1
3. Elaine Rich, Kevin Knight and Nair, “Artificial Intelligence”, TMH, ISBN-978-0-07-008770-5.

### Reference Books:

1. Nilsson Nils J , “Artificial Intelligence: A new Synthesis”, Morgan Kaufmann Publishers Inc. San Francisco, CA, ISBN: 978-1-55-860467-4
2. Patrick Henry Winston, “Artificial Intelligence”, Addison-Wesley Publishing Company, ISBN: 0- 201-53377-4
3. Andries P. Engelbrecht, “Computational Intelligence: An Introduction”, 2nd Edition-Wiley India ISBN: 978-0-470-51250-0
4. Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, “Data Structures and Algorithms in Python” John Wiley & Sons, Incorporated, 2013 ISBN 1118476735, 9781118476734.
5. Robert Lafore “Data Structures and Algorithms in Java”, Second Edition Copyright © 2003 by Sams Publishing.
6. Adam Drozdek “Data Structures and Algorithms in C++”, ISBN-13: 978-1-133-60842-4 ISBN-10: 1-133-60842-6 Cengage Learning.
7. Building-Chatbots-with-Python-Using-Natural-Language-Processing-and-Machine-Learning-by-Sumit-Raj, ISBN No.:978-1484240953.