**** 

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING& RESEARCH

DEPARTMENT OF COMPUTER ENGINEERING

SUBJECT CODE: 310258

LAB MANUAL

Laboratory Practice-II

**(Artificial Intelligence&Cloud Computing)**

**Semester – II, Academic Year**

|  |  |  |
| --- | --- | --- |
| **PCCOE&R Logo** | **Pimpri Chinchwad Education Trust’s**  **Pimpri Chinchwad College of Engineering & Research Ravet, Pune** |  |
| **Academic Year: 2021 - 22** | **Course Outcome LP-II** | **Term:**  **II** |

|  |  |  |  |
| --- | --- | --- | --- |
| **DDepartment: Computer Engineering** | | | **Class: TE-A Batch:T1,T2,T3,T4** |
| **SSubject Name: Laboratory Practice-II** | | **Subject Code:310258** | |
| **Teaching Scheme:-** | **Tut/Pract /week:4 Hrs** | | |
| **Examination Scheme:-** | **PR: 25 Marks** | **TW:50 Marks** | |
| **Name of Faculty:** | **Mrs. Shweta Koparde,Mrs.Vaishali Latke** | | |

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

|  |  |  |
| --- | --- | --- |
| **Subject Teacher**  **Mrs.Shweta A.Koparde**  **Mrs.Vaishali Latke** | **Course Coordinator**  **Mrs.Shweta Koparde**  **Mrs.Vaishali Latke** | **HOD**  **Dr. A.A. Chaugule** |

PimpriChinchwad College of Engineering & Research

Department of ComputerEngineering

**310258: LaboratoryPractice-II**

**TeachingScheme: Examination Scheme:**

Practical:4Hrs/Week Term work: 25Marks

Credits:02 Practical: 25Marks

**List of Laboratory Assignments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.**  **No.** | **Group A** | **Page**  **No.** | **CO** |
| 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. | 3 | C318.1 |
| 2 | Implement A star Algorithm for any game search problem | 7 | C318.1 |
| 3 | Implement Greedy search algorithm for any of the following application:   1. Selection Sort 2. Minimum Spanning Tree 3. Single-Source Shortest Path Problem 4. Job Scheduling Problem 5. Prim's Minimal Spanning Tree Algorithm 6. Kruskal's Minimal Spanning Tree Algorithm   Dijkstra's Minimal Spanning Tree Algorithm | 12 | 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. | 16 | C318.2 |
| 5 | Develop an elementary catboat for any suitable customer interaction application. | 19 | C318.3 |
| 6 | Implement any one of the following Expert System   1. Information management 2. Hospitals and medical facilities 3. Help desks management 4. Employee performance evaluation 5. Stock market trading   Airline scheduling and cargo schedules | 21 | C318.3 |

|  |  |
| --- | --- |
| **Assignment No.:01** | |
| **ProblemStatement:** 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. | |
| **Objectives:**   1. To study the various Search Algorithms. 2. To study the depth first search algorithm. 3. To study the breadth first search algorithm. | |
| **Theory:** **Uninformed Search Algorithms:** The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called **Blind search**.  The following uninformed search algorithms are discussed in this section.   1. Depth First Search 2. Breadth First Search 3. Uniform Cost Search   Each of these algorithms will have:   * A problem **graph,**containing the start node S and the goal node G. * A **strategy,**describing the manner in which the graph will be traversed to get to G. * A **fringe,**which is a data structure used to store all the possible states (nodes) that you can go from the current states. * A **tree,**that results while traversing to the goal node. * A solution **plan,**which the sequence of nodes from S to G.  [**Depth First Search**](https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/): Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.  **Performance Measure:**  *d = the depth of the search tree = the number of levels of the search tree.  ni= number of nodes in level .*  ***Time complexity:****Equivalent to the number of nodes traversed in DFS.*  *T(n) = 1 + n2 + n3+ n4+ …..+nd = O(nd)*  ***Space complexity:****Equivalent to how large can the fringe get.*  *S(n) = O(n\*d)*  ***Completeness:****DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.* ***Optimality:****DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.* [**Breadth First Search**](https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/): 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 all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.  ***d = the depth of the shallowest solution.  ni= number of nodes in level .******Time complexity:****Equivalent to the number of nodes traversed in BFS until the shallowest solution.*  *T(n) = 1 + n2 + n3+ n4+ …..+nd = O(nd)* ***Space complexity:****Equivalent to how large can the fringe get.*  *S(n) = O(nd)* ***Completeness:****BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.*  ***Example:*Find path to move from node S to node G *using DFS and BFS.*** *the costs of all edges are equal.*  **DFS:**    **BFS:** | |
| **Algorithm/Flowchart:**  **DFS:**   * **Step 1** – Push a starting node on stack, mark it visited. * **Step 2** - Visit the adjacent unvisited vertex of start node. Mark it as visited. Display it. Push it in a stack. * **Step 3** − If no adjacent vertex is found, pop up a vertex from the stack. Repeat Step 2 * **Step 4** − Repeat Step 2 and Step 3 until the stack is empty.   **BFS**   * **Step 1** – Insert start node in Queue, mark it visited. * **Step 2** − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue. * **Step 3** − If no adjacent vertex is found, remove the first vertex from the queue. * **Step 4** − Repeat Step 3 and Step 4 until the queue is empty. | |
| **Design diagrams (if any):** | |
| **Input:**  Graph: no of nodes, no of edges  *n = 4, e = 6*  *Enter adjacent node information*  *0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3*  **Output:**  **DFS from vertex 2 – 2, 0, 1, 3**  **BFS from vertex 2 – 2, 0, 3, 1** | |
| **Software Requirement:**   1. Python3/Java 2. Code Editor or IDE – Eclipse/ IntelliJ IDEA/ VSCode | |
| **Hardware Requirement:**  Not specific | |
| **Frequently Asked Questions:**   1. What are the commonly used uninformed search algorithms? 2. Which data structure is used in BFS? 3. Which data structure is used in DFS? 4. What is the performance measure for DFS and BFS? | |
|  | |
| **Conclusion:**  We have successfully implemented depth first search and breadth first search algorithm for a graph | |

|  |
| --- |
| **Assignment No.:02** |
| **Problem Statement:**  Implement A star Algorithm for any game search problem. |
| **Objectives:**   1. To study various Search Algorithms 2. To study the A\* Algorithm |
| **Theory:**  **A\* Algorithm** is the advanced form of the BFS algorithm (Breadth-first search), which searches for the shorter path first than, the longer paths. It is a **complete** as well as an **optimal** solution for solving path and grid problems.  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. So 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 (f(n))**, the node with the least f-score is picked up and explored.  Heuristic used in A\* http://img.brainkart.com/extra/iJOmJug.jpg  Where  g  (n) : The actual cost path from the start node to the current node.  h ( n) : The actual cost path from the current node to goal node.  f  (n) : The actual cost path from the start node to the goal node.  Performance Measure:  **Optimal** – find the least cost from the starting point to the ending point.  **Complete** – It means that it will find all the available paths from start to end.  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).  **Example: Puzzle Problem**  In our 8-Puzzle problem, we can define the **h-score(h(n))** 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(g(n))** will be number of nodes traversed from a start node to get to the current node.  From Fig 1, we can calculate the **h-score** by comparing the initial(current) state and goal state and counting the number of misplaced tiles. Thus, **h-score** = 5 and **g-score**= 0 as the number of nodes traversed from the start node to the current node is 0. Similarly, calculate for all nodes. |
| **Algorithm/Flowchart:**  // A\* Search Algorithm  1. Initialize the open list  2. Initialize the closed list  put the starting node on the open  list (you can leave its **f** at zero)  3. while the open list is not empty  a) find the node with the least **f** on  the open list, call it "q"  b) pop q off the open list  c) generate q's 8 successors and set their  parents to q  d) for each successor  i) if successor is the goal, stop search  ii) else, compute both **g** and **h** for successor  successor.**g** = q.**g** + distance between  successor and q  successor.**h** = distance from goal to  successor (This can be done using many  ways, we will discuss three heuristics-  Manhattan, Diagonal and Euclidean  Heuristics)  successor.**f** = successor.**g** + successor.**h**  iii) if a node with the same position as  successor is in the OPEN list which has a  lower **f** than successor, skip this successor  iV) if a node with the same position as  successor is in the CLOSED list which has  a lower **f** than successor, skip this successor  otherwise, add the node to the open list  end (for loop)  e) push q on the closed list  end (while loop) |
| **Design diagrams (if any):** |
| **Input:**  Start State:  1 2 3  \_ 4 6  7 5 8  Goal State:  1 2 3  4 5 6  7 8 \_  **Output:**  1 2 3  \_ 4 6  7 5 8  |  1 2 3  4 \_ 6  7 5 8  |  1 2 3  4 5 6  7 \_ 8  |  1 2 3  4 5 6  7 8 \_ |
| **Frequently Asked Questions:**   1. What is the heuristic used in A\* Algorithm? 2. Explain the performance measures of A\* Algorithm. 3. List applications of A\* Algorithms. 4. What are Informed Search Algorithms? Name the commonly used Informed Search Algorithms. |
| **Conclusion:**  Successfully implemented A\* Algorithm for 8-Puzzles Problem |

|  |
| --- |
| **Assignment No.:03** |
| **Problem Statement:**  **Implement Greedy search algorithm for any of the following application:**   1. Selection Sort 2. Minimum Spanning Tree 3. Single-Source Shortest Path Problem 4. Job Scheduling Problem 5. Prim's Minimal Spanning Tree Algorithm 6. Kruskal's Minimal Spanning Tree Algorithm   Dijkstra's Minimal Spanning Tree Algorithm |
| Objectives:  1.To study various Greedy Search Algorithms |
| Theory:  Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy.  The greedy method is one of the strategies like Divide and conquer used to solve the problems. This method is used for solving optimization problems. An optimization problem is a problem that demands either maximum or minimum results. Let's understand through some terms.  The Greedy method is the simplest and straightforward approach. It is not an algorithm, but it is a technique. The main function of this approach is that the decision is taken on the basis of the currently available information. 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.  Characteristics of Greedy method  **The following are the characteristics of a greedy method:**   * o construct the solution in an optimal way, this algorithm creates two sets where one set contains all the chosen items, and another set contains the rejected items. * A Greedy algorithm makes good local choices in the hope that the solution should be either feasible or optimal.   Components of Greedy Algorithm  **The components that can be used in the greedy algorithm are:**   * **Candidate set:** A solution that is created from the set is known as a candidate set. * **Selection function:** This function is used to choose the candidate or subset which can be added in the solution. * **Feasibility function:** A function that is used to determine whether the candidate or subset can be used to contribute to the solution or not. * **Objective function:** A function is used to assign the value to the solution or the partial solution. * **Solution function:** This function is used to intimate whether the complete function has been reached or not.   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.   Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.  The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.  This algorithm is not suitable for large data sets as its average and worst case complexities are of Ο(n2), where n is the number of items.  Consider the following depicted array as an example.    For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.    So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.    For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.    We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.    After two iterations, two least values are positioned at the beginning in a sorted manner.  The same process is applied to the rest of the items in the array.  Following is a pictorial depiction of the entire sorting process −    Now, let us learn some programming aspects of selection sort.  Algorithm  Step 1 − Set MIN to location 0  Step 2 − Search the minimum element in the list  Step 3 − Swap with value at location MIN  Step 4 − Increment MIN to point to next element  Step 5 − Repeat until list is sorted |
| Algorithm/Flowchart:  procedure selection sort  list : array of items  n : size of list  for i = 1 to n - 1  /\* set current element as minimum\*/  min = i    /\* check the element to be minimum \*/  for j = i+1 to n  if list[j] < list[min] then  min = j;  end if  end for  /\* swap the minimum element with the current element\*/  if indexMin != i then  swap list[min] and list[i]  end if |
| Frequently Asked Questions:   1. What is the Greedy Algorithm? 2. Explain the selection sort. |
| Conclusion:  Successfully implemented greedy approach for selection sort algorithm |
|  |
| **Assignment No.:04**  **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. |
| **Objectives:**  1.To study Constraint Satisfaction Problem using Branch and Bound Algorithms |
| **Theory:**  We have seen so many techniques like Local search, Adversarial search to solve different problems. The objective of every problem-solving technique is one, i.e., to find a solution to reach the goal. Although, in adversarial search and local search, there were no constraints on the agents while solving the problems and reaching to its solutions.  In this section, we will discuss another type of problem-solving technique known as Constraint satisfaction technique. By the name, it is understood that constraint satisfaction means solving a problem under certain constraints or rules.  Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem. Such type of technique leads to a deeper understanding of the problem structure as well as its complexity.  Constraint satisfaction depends on three components, namely:   * **X:** It is a set of variables. * **D:** It is a set of domains where the variables reside. There is a specific domain for each variable. * **C:** It is a set of constraints which are followed by the set of variables.   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, green and blue respectively. Then vertex c is colored as red as no adjacent vertex of c is colored red. Hence, we could color the graph by 3 colors. Hence, the chromatic number of the graph is |
| Algorithm/Flowchart:  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 |
| Frequently Asked Questions:   1. What is the Constraint satisfaction Algorithm? 2. Explain Graph coloring Problem in detail. |
| Conclusion:  Successfully implemented constraint satisfaction algorithm. |

|  |
| --- |
| **Assignment No.:05** |
| **Problem Statement:**  Develop an elementary catboat for any suitable customer interaction application. |
| **Objectives:**  1.To study elementary catboat for any suitable customer interaction application |
| **Theory:**  [Artificial intelligence chatbots](https://www.sofbang.com/solutions/chatbots-artificial-intelligence/) are text- or voice-based interfaces that provide support and connect human users with the services or information they need by simulating a traditional person-to-person conversation.  Text-based chatbots are often deployed online on websites and social media platforms to provide customer support and outreach. Voice-based chatbots, on the other hand, are most typically used for call deflection and sorting or over-the-phone customer service.  Most smartphones come equipped with a built-in chatbot, and smart speakers with chatbot functionality have been trendy gift-giving items for several years.  The most typical chatbot interaction occurs on a business site. These customer service bots usually pop up after a human user navigates around a site for a few minutes or exhibits behaviors that show that they have become “lost” or are having trouble connecting with the information they need.  Once the chatbot window presents itself, the user can enter their question in plain, syntactical English. [The bot’s language recognition functions break down the question](https://www.sofbang.com/utility-chatbot/) and, at the speed of light, compares the query to its data bank of previously asked questions to look for ways customers have achieved satisfying results in similar situations.  When that search is complete, the chatbot shares the best and most relevant information with the user.  Applications Chatbots are valuable for both businesses and consumers, as [they reduce barriers to](https://www.sofbang.com/technology/mobility/) [data access created by everything from physical disability to tech savviness, streamline](https://www.sofbang.com/technology/mobility/) [navigation to connect users with results](https://www.sofbang.com/technology/mobility/) as quickly as possible, and provide a cost-effective alternative to staffing massive numbers of support professionals or contracting with expensive call centers.  In fact, in many scenarios, [artificial intelligence chatbot services](https://www.sofbang.com/systems-integration/artificial-intelligence-chatbot/) can actually provide a faster, more straightforward experience than dealing with a human professional. If that sounds too good to be true, here are some examples of ways AI-based chatbots are deployed every day:   * **Software Requirements:**   NetBeans, program-ab library.   * **Set-up steps:**   Download the program-ab file.  Copy the bots folder into project folder.  Add the Ab.jar file to the project libraries.  Create the .aiml file in bots/super/aiml folder.   * **Program Execution:**   **Execute a ChatBot Jar file using following command**  **Java –jar dist/ChatBot.jar (Due to execution of jar file, aimlif folder updated automatically)**  **Aimlif contains csv(comma separated files) which are the filed used for early execution**  Human : Hello  Robot : Hi nice to see you!  Human : What are different investment options  Robot : Some of the investment options are Public provident fund Mutual fund Equity shares Real estate investment etc.  Human : I am interested in mutual fund  Robot : Yes we can discuss mutual fund  Human : Minimum investment period  Robot : Only applicable in case of close-ended and ELSS schemes  Human : Let us discuss equity shares  Robot : Yes we can discuss equity shares  Human : Maximum investment amount  Robot : No limit  Human : Bye  Robot : Goodbye ! It was nice to talk to you   * . * **Solution function:** This function is used to intimate whether the complete function has been reached or not. |
| **Frequently Asked Questions:**   1. What is the AI Chatbot? |
| **Conclusion:**  Successfully implemented chatbot application |

|  |
| --- |
| **Assignment No.:06** |
| **Problem Statement:**  Implement any one of the following Expert System   1. Information management 2. Hospitals and medical facilities 3. Help desks management 4. Employee performance evaluation 5. Stock market trading   Airline scheduling and cargo schedules |
| **Objectives:**  1.To study Expert System |
| **Theory:**  A system that uses human expertise to make complicated decisions.Simulates reasoning by applying knowledge and interfaces.Uses expert’s knowledge as rules and data within the system.Models the problem solving ability of a human expert.  Components of an ES:   1. Knowledge Base 2. Represents all the data and information inputed by experts in the field. 3. Stores the data as a set of rules that the system must follow to make decisions. 4. Reasoning or Inference Engine 5. Asks the user questions about what they are looking for. 6. Applies the knowledge and the rules held in the knowledge base. 7. Appropriately uses this information to arrive at a decision. 8. User Interface 9. Allows the expert system and the user to communicate. 10. Finds out what it is that the system needs to answer. 11. Sends the user questions or answers and receives their response. 12. Explanation Facility 13. Explains the systems reasoning and justifies its conclusions.   ?- go.  Does the patient has the symptom headache? : y.  Does the patient has the symptom runny\_nose? : |: n.  Does the patient has the symptom sore\_throat? : |: n.  Does the patient has the symptom abdominal\_pain? : |: y.  Does the patient has the symptom poor\_appetite? : |: y.  Does the patient has the symptom fever? : |: y.  Advices and Sugestions:  1: Chloramphenicol  2: Amoxicillin  3: Ciprofloxacin  4: Azithromycin  Please do complete bed rest and take soft diet because  It is suggested that the patient has typhoid  true . |
| **Frequently Asked Questions:**   1. What is the Expert System? |
| **Conclusion:**  Successfully implemented Expert system application. |