Prcatical No. 1: Manual Content		
	Guru Gobind Singh Foundation's **Guru Gobind Singh College of Engineering and Research Center, Nashik**	
Experiment No: 01		
Title of Experiment: 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..		

Student Name:			
Class:	**TE (Computer)**		
Div:	**-**	**Batch:**	**CO**
Roll No.:			
Date of Attendance (Performance):			
Date of Evaluation:			
Marks (Grade) Attainment of CO Marks out of 10			
CO Mapped	CO1. Design a system using different informed search / uninformed search or heuristic approaches		
Signature of Subject Teacher			

TITLE: Implement DFS and BFS Algorithm. Use and Undirected Graph and develop a Recursive Algorithm for searching all the vertices of the graph or tree data structure.

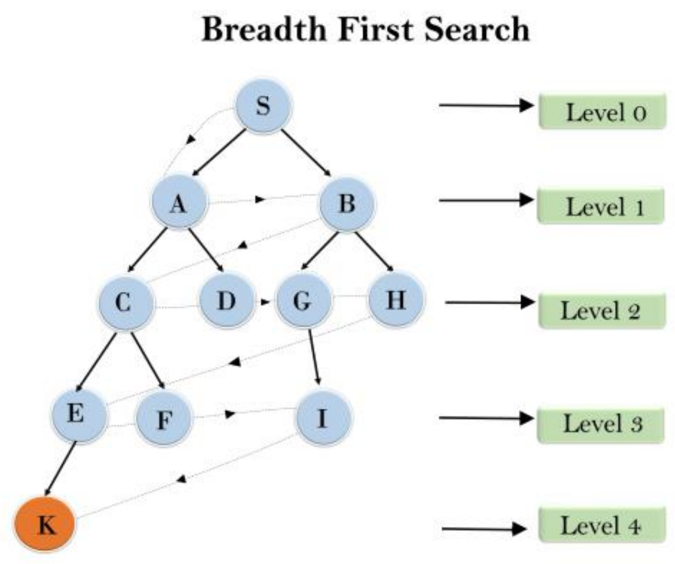
AIM: Aim of this practical is to develop DFS and BFS algorithm programs in programming language.

OBJECTIVES: Based on above main aim following are the objectives

1. To study BFS
2. To study DFS
3. To apply algorithmic logic in implementation of program.

1. Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.



In the above tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K

- **Time Complexity:** Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.
- **$T(b) = 1+b^2+b^3+.......+ b^d= O(b^d)$**
- **Space Complexity:** Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.
- **Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.
- **Optimality:** BFS is optimal if path cost is a non-decreasing function of the depth of the node.

Algorithm:

1. Pick any node, visit the adjacent unvisited vertex, mark it as visited, display it, and insert it in a queue.
2. If there are no remaining adjacent vertices left, remove the first vertex from the queue.
3. Repeat step 1 and step 2 until the queue is empty or the desired node is found.

Program:

```
graph = {
  'A' : ['B','C'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'D' : [],
  'E' : ['F'],
  'F' : []
}
visited = [] # List to keep track of visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)
  while queue:
    s = queue.pop(0)
    print (s, end = " ")

    for neighbour in graph[s]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

# Driver Code
print("Following is the Path using Breadth-First Search")
bfs(visited, graph, 'A')
```

Output:

```
Following is the Path using Breadth-First Search
A B C D E F
```

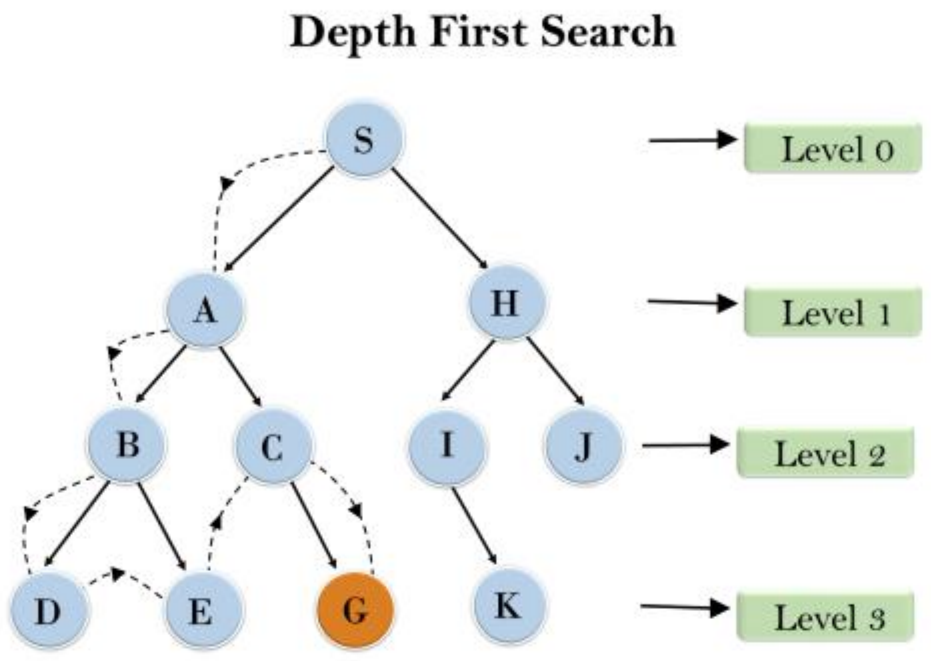
2. Depth-first Search:

- Depth-first search isa recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Example: In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



- **Completeness:** DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.
- **Time Complexity:** Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$$T(n)= 1+ n^2+ n^3 +.........+ n^m=O(n^m)$$

- **Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)**
- **Space Complexity:** DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.
- **Optimal:** DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

Algorithm:
1. We will start by putting any one of the graph's vertex on top of the stack. 2. After that take the top item of the stack and add it to the visited list of the vertex. 3. Next, create a list of that adjacent node of the vertex. Add the ones which aren't in the visited list of vertexes to the top of the stack. 4. Lastly, keep repeating steps 2 and 3 until the stack is empty.
Program:

```
# Using a Python dictionary to act as an adjacency list
graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
print("Following is the Path using Depth-First Search")
dfs(visited, graph, 'A')
```

Output:

```
Following is the Path using Depth-First Search
A
B
D
E
F
C
```

Conclusion: Thus we have studied BFS & DFS algorithm in detail and implemented using recursive function.

Prcatical No. 2: Manual Content		
	Guru Gobind Singh Foundation's **Guru Gobind Singh College of Engineering and Research Center, Nashik**	
Experiment No: 02		
Title of Experiment: Implement A star Algorithm for any game search problem.		

Student Name:			
Class:	**TE (Computer)**		
Div:	**-**	**Batch:**	**CO**
Roll No.:			
Date of Attendance (Performance):			
Date of Evaluation:			
Marks (Grade) Attainment of CO Marks out of 10			
CO Mapped	CO1. Design a system using different informed search / uninformed search or heuristic approaches		
Signature of Subject Teacher			

TITLE: Implement A* Algorithm for any game search problem

AIM: Aim of this practical is to apply algorithmic logic for any game

OBJECTIVES: Based on above main aim following are the objectives

1. To study A* algorithm
2. To study different game search problems
3. To determine performance A* algorithm in various games

MOTIVATION: To approximate the shortest path in real-life situations, like- in maps, games where there can be many hindrances.

INTRODUCTION

A* Search Algorithm:

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals. A* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms. This fact is cleared in detail in below sections. And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation). Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithm comes to the rescue. What A* Search Algorithm does is that at each step it picks the node according to a value-‘f’ which is a parameter equal to the sum of two other parameters – ‘g’ and ‘h’. At each step it picks the node/cell having the lowest ‘f’, and process that node/cell. We define ‘g’ and ‘h’ as simply as possible below

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.

h = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don’t know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.). There can be many ways to calculate this ‘h’ which are discussed in the later sections.

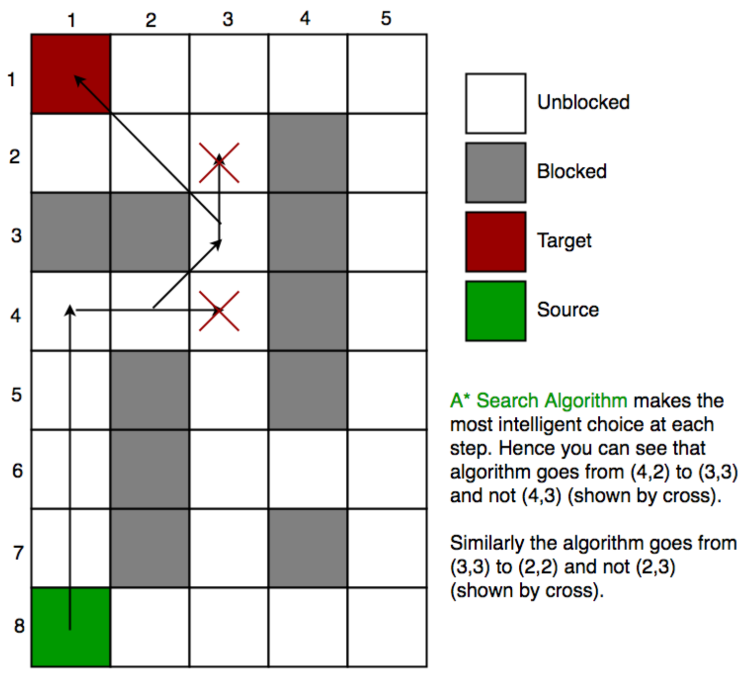
Algorithm

We create two lists – Open List and Closed List (just like Dijkstra Algorithm)

```
// 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
          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

        ii) if a node with the same position as
```

```
            successor is in the OPEN list which has a
            lower f than successor, skip this successor

        iii) 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)
```

So suppose as in the below figure if we want to reach the target cell from the source cell, then the A* Search algorithm would follow path as shown below. Note that the below figure is made by considering Euclidean Distance as a heuristics.


1
2
3
4
5
1
2
3
4
5
6
7
8
Unblocked
Blocked
Target
Source
A* Search Algorithm makes the most intelligent choice at each step. Hence you can see that algorithm goes from (4,2) to (3,3) and not (4,3) (shown by cross).
Similarly the algorithm goes from (3,3) to (2,2) and not (2,3) (shown by cross).

Heuristics:

We can calculate g but how to calculate h ?

A) Either calculate the exact value of h (which is certainly time consuming).

OR

B) Approximate the value of h using some heuristics (less time consuming).

A) Exact Heuristics –

We can find exact values of h, but that is generally very time consuming.

Below are some of the methods to calculate the exact value of h.

1) Pre-compute the distance between each pair of cells before running the A* Search Algorithm.
2) If there are no blocked cells/obstacles then we can just find the exact value of h without any pre-computation using the distance formula/Euclidean Distance

B) Approximation Heuristics –

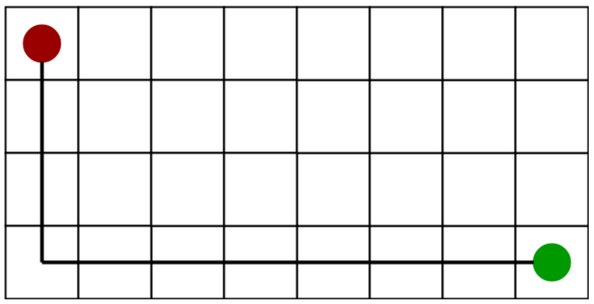
There are generally three approximation heuristics to calculate h –

1) **Manhattan Distance** – It is nothing but the sum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively, i.e.,

```
h = abs (current_cell.x – goal.x) +
abs (current_cell.y – goal.y)
```

When to use this heuristic? – When we are allowed to move only in four directions only (right, left, top, bottom)

The Manhattan Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).



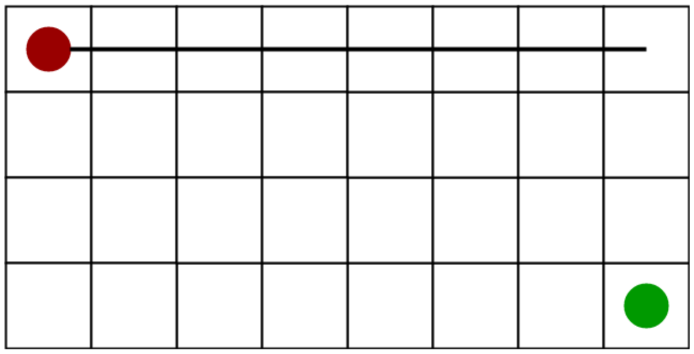
2) Diagonal Distance-

It is nothing but the maximum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively, i.e.,

```
dx = abs(current_cell.x – goal.x)
dy = abs(current_cell.y – goal.y)


h = D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)

where D is length of each node(usually = 1) and D2 is diagonal
distance between each node (usually = sqrt(2) ).
```

When to use this heuristic? – When we are allowed to move in eight directions only (similar to a move of a King in Chess)

The Diagonal Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).



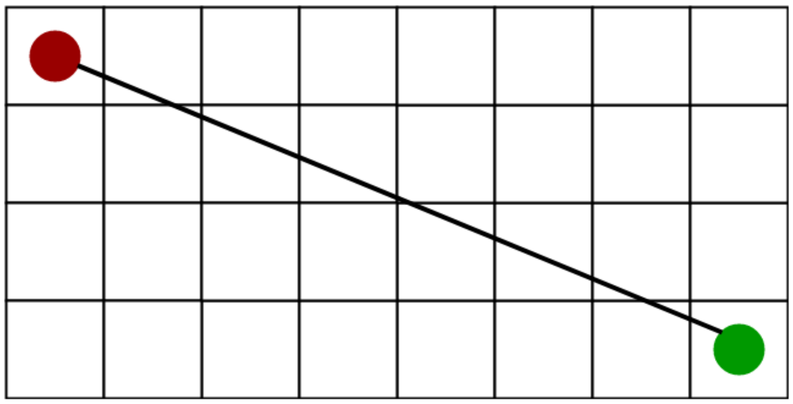
3) Euclidean Distance-

As it is clear from its name, it is nothing but the distance between the current cell and the goal cell using the distance formula

```
h = sqrt ( (current_cell.x – goal.x)2 +
           (current_cell.y – goal.y)2 )
```

When to use this heuristic? – When we are allowed to move in any directions.

The Euclidean Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).



Implementation:

We can use any data structure to implement open list and closed list but for best performance, we use a set data structure of C++ STL(implemented as Red-Black Tree) and a boolean hash table for a closed list.

The implementations are similar to Dijsktra's algorithm. If we use a Fibonacci heap to implement the open list instead of a binary heap/self-balancing tree, then the performance will become better (as Fibonacci heap takes O(1) average time to insert into open list and to decrease key)

CONCLUSIONS: Thus we have studied A* algorithm in detail and implemented it in game search problem

Assignment No: 3

1. Title of Assignment:

Implement Greedy search algorithm for Selection Sort.

2. Prerequisite:

Basic knowledge of Greedy algorithm and Sorting concept.

3. Objective:

In this experiment, we will be able to do the following:

- Study how selection sort works under greedy search algorithm.

4. Outcome: Successfully able to sort , unsorted list of numbers.

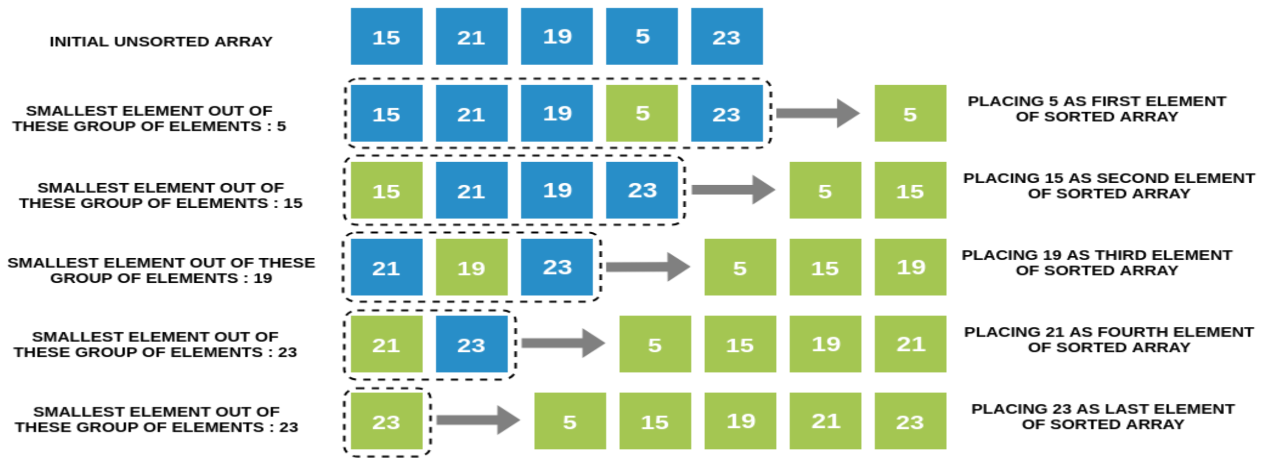
5. Software and Hardware Requirement:

Open Source C++ Programming tool like G++/GCC, python,java and Ubuntu.

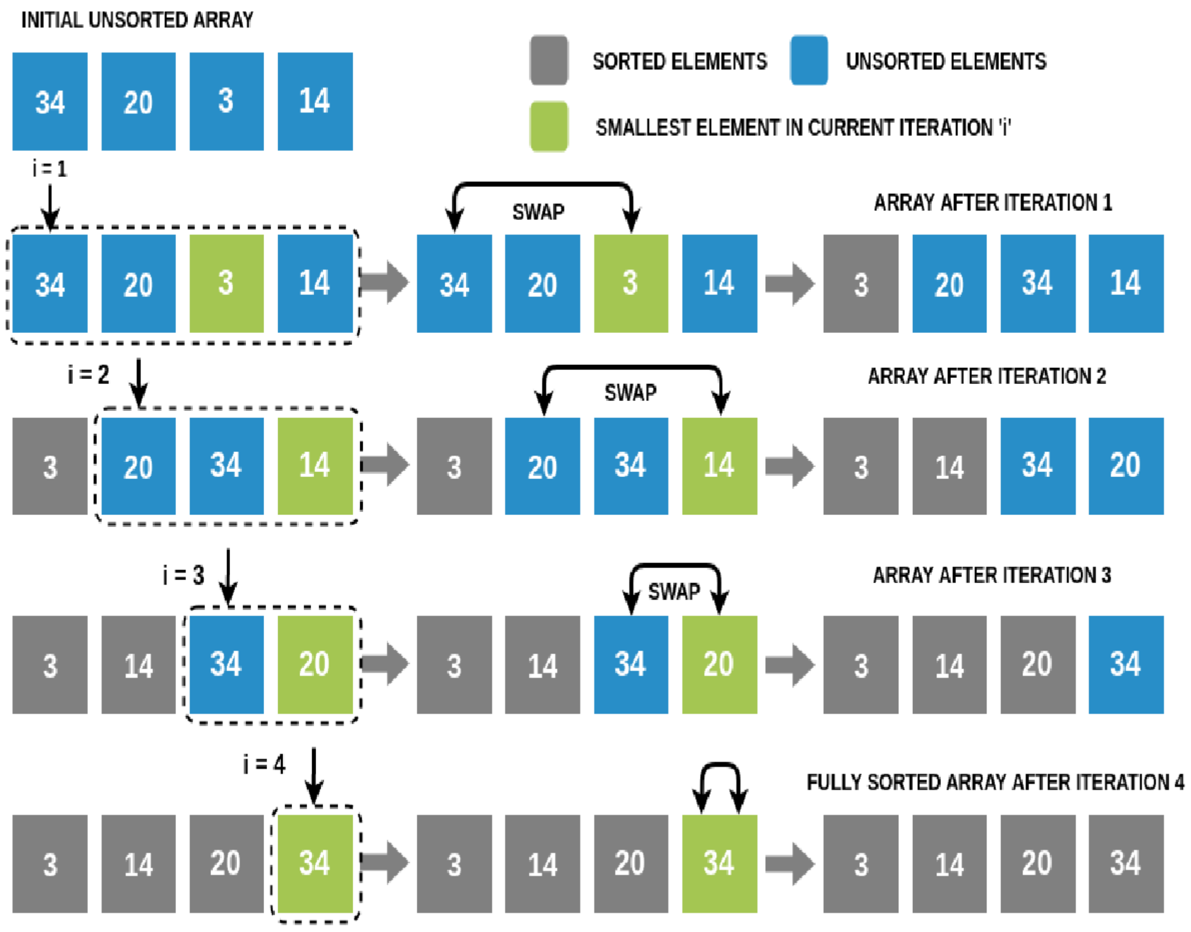
6. Relevant Theory / Literature Survey:

In Selection Sort, we take the simplest, most intuitive approach to sort an array. Choose the smallest number, place it in the first position. Then choose the next smallest number out of the remaining elements, and place it in the second position and so on till the end.

Intuition Behind the Algorithm


INITIAL UNSORTED ARRAY
15 21 19 5 23
SMALLEST ELEMENT OUT OF THESE GROUP OF ELEMENTS : 5
PLACING 5 AS FIRST ELEMENT OF SORTED ARRAY
SMALLEST ELEMENT OUT OF THESE GROUP OF ELEMENTS : 15
PLACING 15 AS SECOND ELEMENT OF SORTED ARRAY
SMALLEST ELEMENT OUT OF THESE GROUP OF ELEMENTS : 19
PLACING 19 AS THIRD ELEMENT OF SORTED ARRAY
SMALLEST ELEMENT OUT OF THESE GROUP OF ELEMENTS : 23
PLACING 21 AS FOURTH ELEMENT OF SORTED ARRAY
SMALLEST ELEMENT OUT OF THESE GROUP OF ELEMENTS : 23
PLACING 23 AS LAST ELEMENT OF SORTED ARRAY

Selection Sort Algorithm



We will perform N-1 iterations on the array (N is the number of elements in the array). In iteration i (1≤i≤N-1):

- We will traverse the array from the ith index to the end and find the smallest number among these elements. Note that if there are two smallest elements of the same value, we choose the one with the lower index.
- We will swap this smallest element with the ith element.
- Hence at the end of the ith iteration, we have found the ith smallest number, and placed it at the ith position in the array.

In the (N-1)th iteration, we will place the (N-1)th smallest element, which is the 2nd largest element in the array, at the second last position. This will leave us with one element which would already be at its correct place.This completes our sorting! Selection Sort Algorithm.

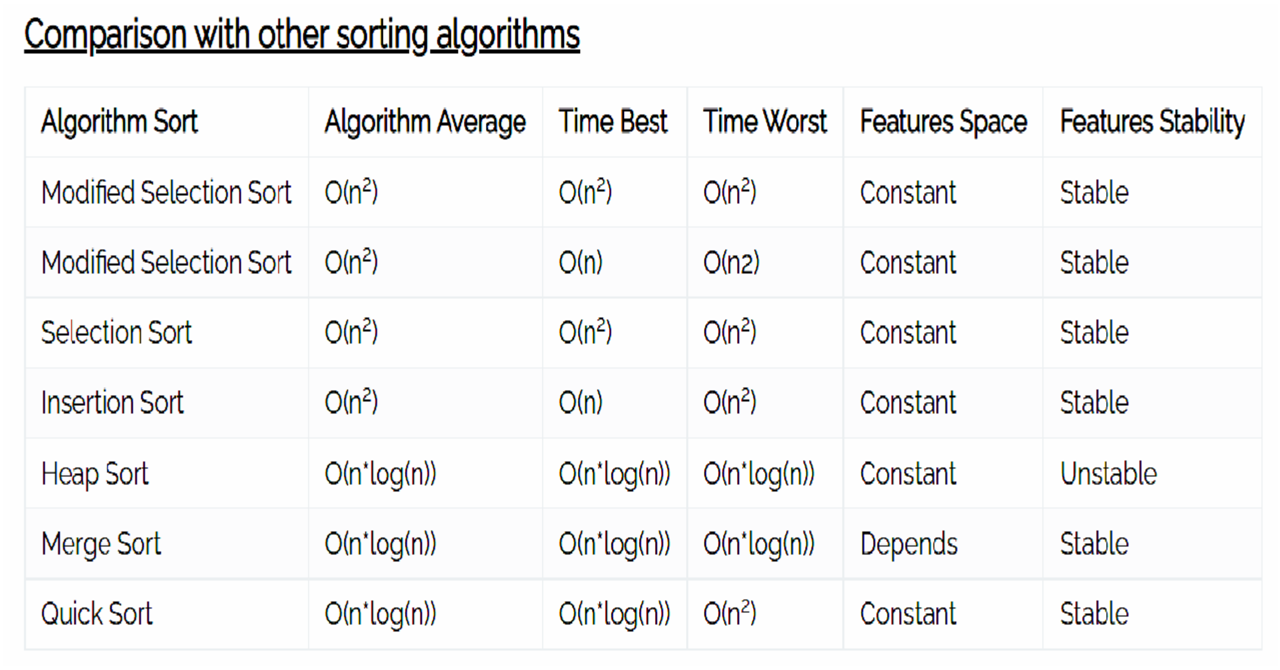
Running Time of Selection Sort

Let's assume that we are sorting N elements of a given array using Selection Sort.

- To complete one iteration, we traverse a part of the array (from index i to the end) exactly once (while keeping track of the smallest element encountered so far). Since the longest length we ever traverse in any given iteration is N (in the first iteration when i=1 -> from first to last element), time complexity of completing one iteration is O(N).
- In Selection Sort, we run N iterations, each of which takes O(N) time. Hence overall time complexity becomes O(N*N).
- Note that even if the array is fully sorted initially, Selection Sort will take O(N^2) time to complete, just as it will take for a reverse sorted or randomly sorted array.

Space Complexity of Selection Sort

While swapping two elements, we need some extra space to store temporary values. Other than that, the sorting can be done in-place. Hence space complexity is O(1) or constant space.

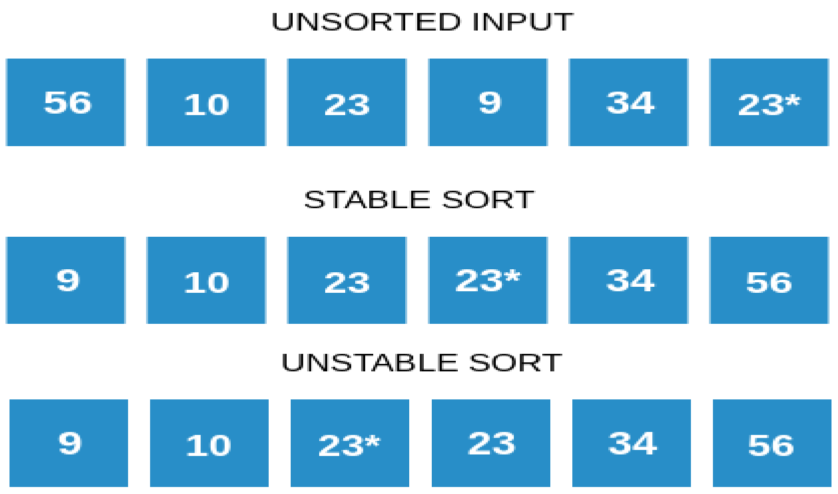


Comparison with other sorting algorithms

Algorithm Sort	Algorithm Average	Time Best	Time Worst	Features Space	Features Stability
Modified Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Constant	Stable
Modified Selection Sort	$O(n^2)$	O(n)	O(n2)	Constant	Stable
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Constant	Stable
Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$	Constant	Stable
Heap Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Constant	Unstable
Merge Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Depends	Stable
Quick Sort	O(n*log(n))	O(n*log(n))	$O(n^2)$	Constant	Stable

What is a Stable Sort Algorithm?

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array.



Is Selection Sort Stable?

Yes, Selection Sort is a stable sorting algorithm. When looking for the smallest element, we choose the element with lower index in case there are two or more equal elements that are the smallest elements in the array. This makes sure that we preserve the relative ordering between equal elements.

7. Questions:

Q 1: What is the time and space complexity of selection sort?

Q 2: If an array is [6,1,9,10] , sort the list by selection sort, step wise.

Q 3: What is the maximum number of comparisons in one iteration for an array of size N?

Q 4: Draw Comparison chart with other sorting techniques with respect to time and space complexity.

Q 5: What is a Stable Sort Algorithm? whether selection sort is a stable algorithm?

8. Conclusion:

In This way we have studied how to sort , unsorted list of numbers using selection sort.

Prcatical No. 4: Manual Content			
	Guru Gobind Singh Foundation's **Guru Gobind Singh College of Engineering and Research Center, Nashik**		
Experiment No: 04			
Title of Experiment: Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.			
Student Name:			
Class:	**TE (Computer)**		
Div:	**-**	**Batch:**	**CO**
Roll No.:			
Date of Attendance (Performance):			
Date of Evaluation:			
Marks (Grade) Attainment of CO Marks out of 10	see table below		
CO Mapped	CO2: Apply basic principles of AI in solutions that require problem solving, inference, perception, knowledge representation, and learning		
Signature of Subject Teacher			

A	P	W	T	Total

<u>TITLE:</u> Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem

<u>AIM:</u> To use Branch & Bound and Backtracking to solve CSP & n-queens or graph coloring problem.

<u>OBJECTIVES:</u> Based on above main aim following are the objectives

1. To study Branch & Bound and Backtracking methods.
2. To study Constraint Satisfaction Problem
3. To study n-queen or graph coloring problem

<u>Theory:</u>

Constraint Satisfaction 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. 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.

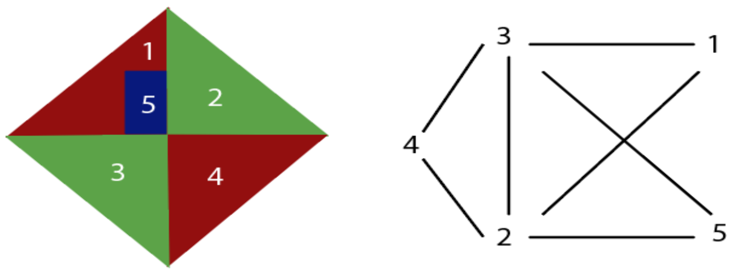
C: It is a set of constraints which are followed by the set of variables.

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.

CSP Problems:

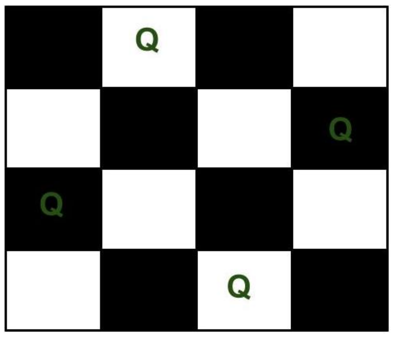
Constraint satisfaction includes those problems which contains some constraints while solving the problem. CSP includes the following problems:

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



Graph Coloring

n-queen problem: In n-queen problem, the constraint is that no queen should be placed either diagonally, in the same row or column. The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.



Backtracking Algorithm to solve n-queen problem:

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

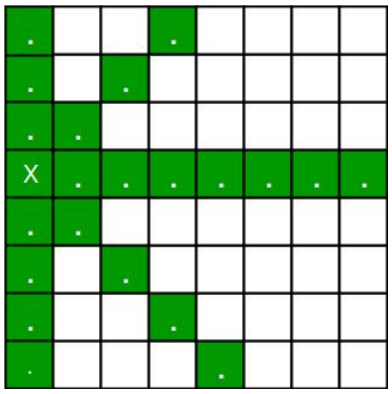
```
1) Start in the leftmost column
```

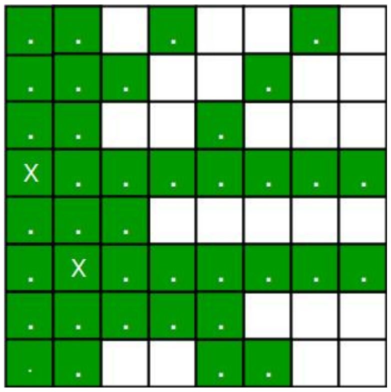
```
2) If all queens are placed
    return true
3) Try all rows in the current column.
   Do following for every tried row.
    a) If the queen can be placed safely in this row
       then mark this [row, column] as part of the
       solution and recursively check if placing
       queen here leads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

N Queen Problem using Branch and Bound:

In backtracking solution we backtrack when we hit a dead end. In Branch and Bound solution, after building a partial solution, we figure out that there is no point going any deeper as we are going to hit a dead end.

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





1. For the 1st Queen, there are total 8 possibilities as we can place 1st Queen in any row of first column. Let's place Queen 1 on row 3.
2. After placing 1st Queen, there are 7 possibilities left for the 2nd Queen. But wait, we don't really have 7 possibilities. We cannot place Queen 2 on rows 2, 3 or 4 as those cells are under attack from Queen 1. So, Queen 2 has only 8 – 3 = 5 valid positions left.
3. After picking a position for Queen 2, Queen 3 has even fewer options as most of the cells in its column are under attack from the first 2 Queens.

We need to figure out an efficient way of keeping track of which cells are under attack. In previous solution we kept an 8--by--8 Boolean matrix and update it each time we placed a queen, but that required linear time to update as we need to check for safe cells.

Basically, we have to ensure 4 things:

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

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

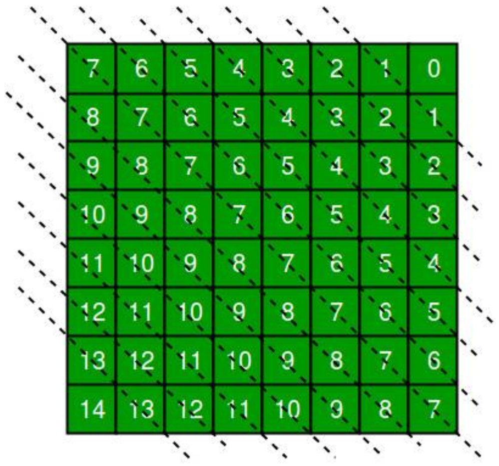
Let's do some pre-processing first. Let's create two N x N matrix one for / diagonal and other one for \ diagonal. Let's call them slashCode and backslashCode respectively. The trick is to fill them in such a way that two queens sharing a same /-diagonal will have the same value in matrix slashCode,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

For an N x N matrix, fill slashCode and backslashCode matrix using below formula –

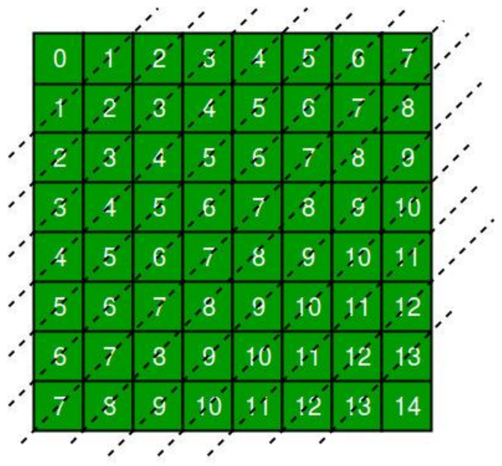
slashCode[row][col] = row + col

backslashCode[row][col] = row – col + (N-1)

Using above formula will result in below matrices



r -c + 7



r + c

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

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

Conclusion: Thus we have studied how to solve n-queen (Constraint Satisfaction Problem) using Backtracking and Branch & Bound.

Prcatical No. 5: Manual Content

	Guru Gobind Singh Foundation's **Guru Gobind Singh College of Engineering and Research Center, Nashik**	

Experiment No: 05

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

Student Name:			
Class:	**TE (Computer)**		
Div:	**-**	**Batch:**	**CO**
Roll No.:			
Date of Attendance (Performance):			
Date of Evaluation:			

Marks (Grade) Attainment of CO Marks out of 10

A	**P**	**W**	**T**	**Total**

CO Mapped	CO3: Design and develop an interactive AI application
Signature of Subject Teacher	

TITLE: Develop an elementary chatbot for any suitable customer interaction application

AIM: To create Chatbot for customer interaction

OBJECTIVES: Based on above main aim following are the objectives

1. To study theory of Chatbot
2. To create Chatbot

Theory:

Chatbots:

A chatbot can be defined as a computer program that converses with users over the internet as any human would with natural language and sentiments. It is a program or piece of software that automatically responds to messages sent over a website chatbox, email, social media messaging app, or text.

For a deeper understanding of Chatbot, we can define it as a computer program that impersonates human conversations in its natural format, which may include text (since the advent of bots) or spoken language using artificial intelligence (AI) techniques such as Natural Language Processing (NLP) and audio analysis. One of the primary aspects of an AI-based bot is that it is dynamic.

A chatbot is a smart application that reduces human work and helps an organization to solve basic queries of the customer. Today most of the companies, business from different sector makes use of chatbot in a different way to reply their customer as fast as possible. chatbots also help in increasing traffic of site which is top reason of business to use chatbots.

Chatbot asks for basic information of customers like name, email address, and the query. If a query is simple like product fault, booking mistake, need some information then without any human connection it can solve it automatically and If some problem is high then It passes the details to the human head and helps customer to connect with organization manager easily. And most of the customers like to deal and talk with a chatbot.

How do the Chatbots function?

The main technology that lies behind chatbots is NLP and Machine Learning.

When a question is presented to a chatbot, a series or complex algorithms process the received input, understand what the user is asking, and based on that, determines the answer suitable to the question.

Chatbots have to rely on the ability of the algorithms to detect the complexity of both text and spoken words. Some chatbots perform very well to the point it becomes difficult to differentiate whether the user is a machine or a human.

However, handling complex conversations is a huge challenge; where there is a usage of various figures of speech, it may be difficult for machines to understand.

Why we need Chatbots?

Cost and Time Effective: Humans cannot be active on-site 24/7 but chatbots can and the replying power of chatbots is much fast than humans.

Cheap Development cost: with the advancement in technology many tools are developed that help easy development and integration of chatbots with little investment.

Human Resource: Today Chatbots can also talk with text o speech technology so it gives the feel as a human is talking on another side.

Business Branding: Businesses are changing with technology and chatbot is one out of them. Chatbot also helps in advertising, branding of organization product and services and give daily updates to users.

Types of Chatbots:

1. **Rule-based chatbots:** Chatbots follow a set of established rules or flows to respond to questions posted by a user. All your simple applications contain rule-based chatbots, which respond to queries based on the rules they are trained on. For instance, a weather application, where you ask for weather forecast and it fetches the data from different sources and responds with the information.

   Rule-based chatbots may not be able to hold complex conversations. It can only accomplish the tasks it is programmed to perform unless more improvements are made by the developer.

2. **Self-learning chatbots: S**elf-learning bots are highly efficient because they are capable to grab and identify the user's intent on their own. They are build using advanced tools and techniques of Machine Learning, Deep Learning, and NLP. Self-learning bots are further divided into 2 subcategories.

   a. **Retrieval-based chatbots**:- Retrieval-based it is somewhat the same as Rule-based where predefined input patterns and responses are embedded.

b. **Generative-Based chatbots**:- It is based on the same phenomenon as Machine Translation build using sequence 2 sequences neural network.

Most of the organization uses self-learning chatbot along with embedding some rules like Hybrid version of both methods which makes chatbot powerful to handle each situation during a conversation with a customer.

Chatbot Development Platforms:

IBM Watson: Watson is one of the most preferred platforms when it comes to building AI chatbots. The advantage of Watson is its capability to serve different verticals and manage complex interactions with ease.

Microsoft Azure Bot Service: The Azure bot service provides the developer with SDK and portal, along with a bot connector service that will allow the developer to connect to any social media platform. The SDK also helps with debugging your bot and provides a large selection of sample bots that can be used as building blocks for your bot. This Cloud-based service is accessible from almost anywhere and provides multiple language support.

QnA Maker: This is another bot from Microsoft, which is exactly as the name suggests. It can be of great help to any business that is asked frequent questions from their customers regarding their products. QnA Maker allows you to develop and train your bots for answering simple questions, based on your FAQ URLs, any structured documents, and manuals for the product within a matter of minutes.

Chatbot Deployment Platforms:

Once chatbots are developed, they need to be deployed to a deployment platform. You will have to choose a deployment platform based on your customer base. However, the use of chatbots revolves mostly around social media platforms or virtual assistant features in various devices. Let us look at some of the emerging bot platform ecosystems.

Facebook Messenger: With over 1 billion users, there is no denying that Facebook has a wide reach around the world. For developers who are developing bots, this is a great platform to reach out to a bigger audience. Facebook has been investing in bot development and has provided tools for users to create bots for their specific needs without writing a single line of code. Fast food joints like Burger King has leveraged the use of bots to serve their customers by taking their order via Facebook. Many businesses have used Facebook to their advantage and improved ways of serving their customer base.

Skype for Business: This is another popular instant messaging platform utilized by many businesses around the world for their internal or external communication. Bots like Skyscanner allow you to make travel arrangements right in your Skype window. In addition, it helps you to find the most affordable travel options. Bots like Bing Image Preview and Getty Images allow you to search for images right from your Skype search bar.

Kik: It is an instant messaging platform, used for internal communication in businesses. One of the most popular bots on this platform is The Weather Channel. It forecasts the weather for you and lets you know if there is going to be any change in the weather. This is great for traveling professionals as they can plan their schedules accordingly.

There are other messaging platforms with interesting bots that have been used to make business operations smoother and easier.

How to Make a Chatbot in Python?

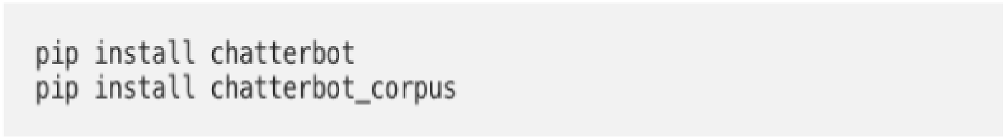
To build a chatbot in Python, you have to import all the necessary packages and initialize the variables you want to use in your chatbot project. Also, remember that when working with text data, you need to perform data preprocessing on your dataset before designing an ML model.

This is where tokenizing helps with text data – it helps fragment the large text dataset into smaller, readable chunks (like words). Once that is done, you can also go for lemmatization that transforms a word into its lemma form. Then it creates a pickle file to store the python objects that are used for predicting the responses of the bot.

Another vital part of the chatbot development process is creating the training and testing datasets.

1. **Prepare the Dependencies**

   The first step in creating a chatbot in Python with the ChatterBot library is to install the library in your system. It is best if you create and use a new Python virtual environment for the installation. To do so, you have to write and execute this command in your Python terminal:



```
pip install chatterbot
pip install chatterbot_corpus
```

You can also install ChatterBot's latest development version directly from GitHub. For this, you will have to write and execute the following command:

pip install git+git://github.com/gunthercox/ChatterBot.git@master

If you wish to upgrade the command, you can do so as well:

```
pip install --upgrade chatterbot_corpus
pip install --upgrade chatterbot
```

Now that your setup is ready, we can move on to the next step to create chatbot using python.

2. **Import Classes:** Importing classes is the second step in the Python chatbot creation process. All you need to do is import two classes – ChatBot from chatterbot and ListTrainer from chatterbot.trainers. To do this, you can execute the following command:

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

3. **Create and Train the Chatbot:** This is the third step on creating chatbot in python. The chatbot you are creating will be an instance of the class "ChatBot." After creating a new ChatterBot instance, you can train the bot to improve its performance. Training ensures that the bot has enough knowledge to get started with specific responses to specific inputs. You have to execute the following command now:

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

Here, the argument (that corresponds to the parameter name) represents the name of your Python chatbot. If you wish to disable the bot's ability to learn after the training, you can include the "read_only=True" command. The command "logic_adapters" denotes the list of adapters used to train the chatbot.

While the "chatterbot.logic.MathematicalEvaluation" helps the bot to solve math problems, the "chatterbot.logic.BestMatch" helps it to choose the best match from the list of responses already provided.

Since you have to provide a list of responses, you can do it by specifying the lists of strings that can be later used to train your Python chatbot, and find the best match for each query. Here's an example of responses you can train your chatbot using python to learn:

```
small_talk = ['hi there!',
              'hi!',
              'how do you do?',
              'how are you?',
              'i\'m cool.',
              'fine, you?',
              'always cool.',
              'i\'m ok',
              'glad to hear that.',
              'i\'m fine',
              'glad to hear that.',
              'i feel awesome',
              'excellent, glad to hear that.',
              'not so good',
              'sorry to hear that.',
              'what\'s your name?',
              'i\'m pybot. ask me a math question, please.']

math_talk_1 = ['pythagorean theorem',
               'a squared plus b squared equals c squared.']

math_talk_2 = ['law of cosines',
               'c**2 = a**2 + b**2 - 2 * a * b * cos(gamma)']
```

You can also create and train the bot by writing an instance of "ListTrainer" and supplying it with a list of strings like so:

```
list_trainer = ListTrainer(my_bot)

for item in (small_talk, math_talk_1, math_talk_2):
    list_trainer.train(item)
```

Now, your Python chatbot is ready to communicate.

4. **Communicate with the Python Chatbot:** To interact with your Python chatbot, you can use the .get_response() function. This is how it should look while communicating:

```
>>> print(my_bot.get_response("hi"))
how do you do?


>>> print(my_bot.get_response("i feel awesome today"))
excellent, glad to hear that.


 >>> print(my_bot.get_response("what's your name?"))
i'm pybot. ask me a math question, please.


>>> print(my_bot.get_response("show me the pythagorean
theorem"))
a squared plus b squared equals c squared.


>>> print(my_bot.get_response("do you know the law of
cosines?"))
c**2 = a**2 + b**2 - 2 * a * b * cos(gamma)
```

However, it is essential to understand that the chatbot using python might not know how to answer all your questions. Since its knowledge and training is still very limited, you have to give it time and provide more training data to train it further.

5. **Train your Python Chatbot with a Corpus of Data:**

   In this last step of how to make a chatbot in Python, for training your python chatbot even further, you can use an existing corpus of data. Here's an example of how to train your Python chatbot with a corpus of data provided by the bot itself:

```
from chatterbot.trainers import ChatterBotCorpusTrainer


corpus_trainer = ChatterBotCorpusTrainer(my_bot)
corpus_trainer.train('chatterbot.corpus.english')
```

The good thing is that ChatterBot offers this functionality in many different languages. So, you can also specify a subset of a corpus in a language you would prefer.

Conclusion: Thus we have successfully implemented elementary Chatbot for Customer interaction application.

Prcatical No. 6: Manual Content		
	Guru Gobind Singh Foundation's **Guru Gobind Singh College of Engineering and Research Center, Nashik**	

Experiment No: 06

Implement any one of the following Expert System

I. **Information management**
II. **Hospitals and medical facilities**
III. **Help desks management**
IV. **Employee performance evaluation**
V. **Stock market trading**
VI. **Airline scheduling and cargo schedules**

Student Name:			
Class:	**TE (Computer)**		
Div:	**-**	**Batch:**	**CO**
Roll No.:			
Date of Attendance (Performance):			
Date of Evaluation:			

Marks (Grade) Attainment of CO Marks out of 10	**A**	**P**	**W**	**T**	**Total**

CO Mapped	CO3: Design and develop an interactive AI application
Signature of Subject Teacher	

<u>TITLE:</u> Implement any one of the following Expert System

I. Information management

II. Hospitals and medical facilities

III. Help desks management

IV. Employee performance evaluation

V. Stock market trading

VI. Airline scheduling and cargo schedules

<u>AIM:</u> To create Expert System for any of above chosen topic.

<u>OBJECTIVES:</u> Based on above main aim following are the objectives

1. To study theory of Expert System
2. To create Expert System

<u>Theory:</u>

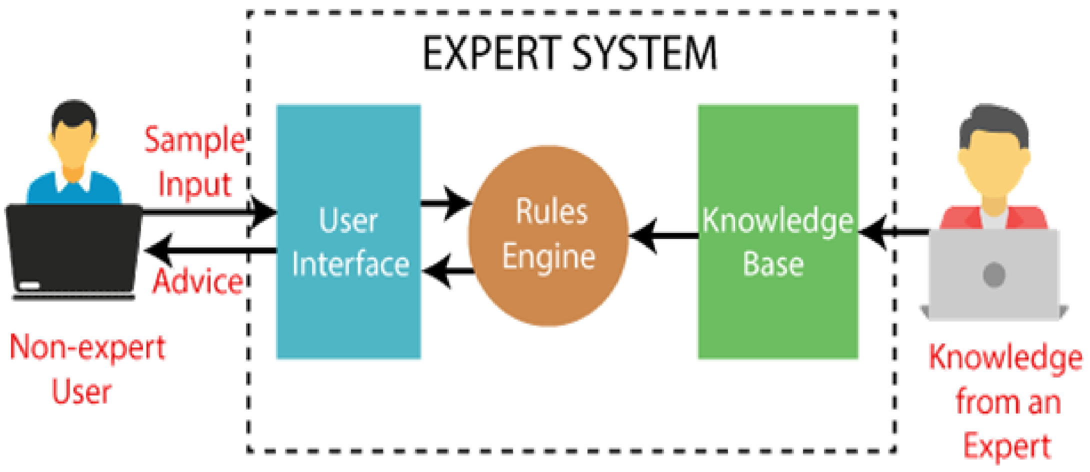
Expert System:

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence. It solves the most complex issue as an expert by extracting the knowledge stored in its knowledge base. The system helps in decision making for complex problems using both facts and heuristics like a human expert. It is called so because it contains the expert knowledge of a specific domain and can solve any complex problem of that particular domain. These systems are designed for a specific domain, such as medicine, science, etc.

The performance of an expert system is based on the expert's knowledge stored in its knowledge base. The more knowledge stored in the KB, the more that system improves its performance. One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.

Below is the block diagram that represents the working of an expert system:



Below are some popular examples of the Expert System:

DENDRAL: It was an artificial intelligence project that was made as a chemical analysis expert system. It was used in organic chemistry to detect unknown organic molecules with the help of their mass spectra and knowledge base of chemistry.

MYCIN: It was one of the earliest backward chaining expert systems that was designed to find the bacteria causing infections like bacteraemia and meningitis. It was also used for the recommendation of antibiotics and the diagnosis of blood clotting diseases.

PXDES: It is an expert system that is used to determine the type and level of lung cancer. To determine the disease, it takes a picture from the upper body, which looks like the shadow. This shadow identifies the type and degree of harm.

CaDeT: The CaDet expert system is a diagnostic support system that can detect cancer at early stages.

Characteristics of Expert System:

High Performance: The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.

Understandable: It responds in a way that can be easily understandable by the user. It can take input in human language and provides the output in the same way.

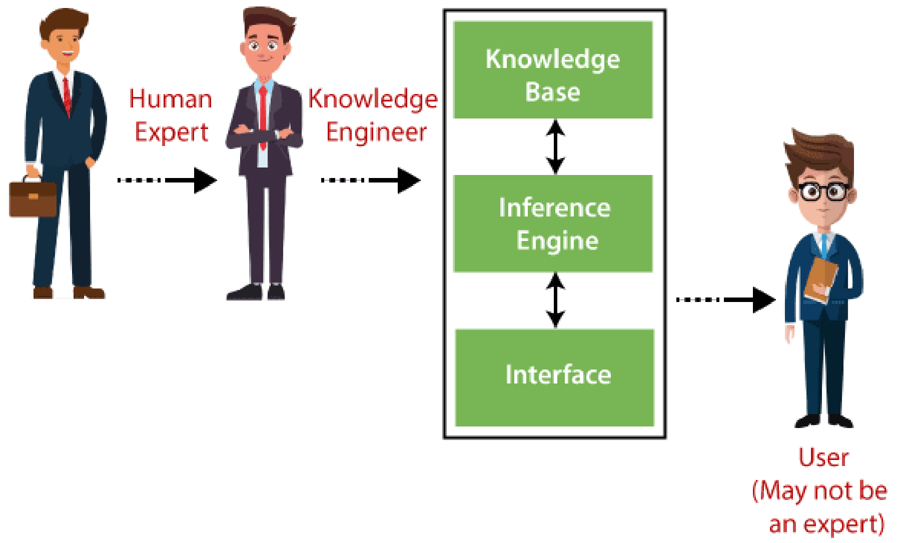
Reliable: It is much reliable for generating an efficient and accurate output.

Highly responsive: ES provides the result for any complex query within a very short period of time.

Components of Expert System

An expert system mainly consists of three components:

- User Interface
- Inference Engine
- Knowledge Base



1. User Interface

With the help of a user interface, the expert system interacts with the user, takes queries as an input in a readable format, and passes it to the inference engine. After getting the response from the inference engine, it displays the output to the user. In other words, it is an interface that helps a non-expert user to communicate with the expert system to find a solution.

2. Inference Engine (Rules of Engine)

- The inference engine is known as the brain of the expert system as it is the main processing unit of the system. It applies inference rules to the knowledge base to derive a conclusion or deduce new information. It helps in deriving an error-free solution of queries asked by the user.
- With the help of an inference engine, the system extracts the knowledge from the knowledge base.
- There are two types of inference engine:

- **Deterministic Inference engine:** The conclusions drawn from this type of inference engine are assumed to be true. It is based on facts and rules.
- **Probabilistic Inference engine:** This type of inference engine contains uncertainty in conclusions, and based on the probability.

Inference engine uses the below modes to derive the solutions:

- **Forward Chaining:** It starts from the known facts and rules, and applies the inference rules to add their conclusion to the known facts.
- **Backward Chaining:** It is a backward reasoning method that starts from the goal and works backward to prove the known facts.

3. Knowledge Base

The knowledgebase is a type of storage that stores knowledge acquired from the different experts of the particular domain. It is considered as big storage of knowledge. The more the knowledge base, the more precise will be the Expert System.

It is similar to a database that contains information and rules of a particular domain or subject. One can also view the knowledge base as collections of objects and their attributes. Such as a Lion is an object and its attributes are it is a mammal, it is not a domestic animal, etc.

Explain your implemented Expert System:

Conclusion: