Sinhgad Institutes

Sinhgad Institute of Technology & Science, Narhe Pune

LAB MANUAL

Lab Practice II

(Artificial Intelligence)

Semester-VI

Department of Computer Engineering

Faculty In-charge: Vidhya R Vasekar

Experiment No. 1

Aim: 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 study and implementation of searching techniques.

Outcome: Students will be understand the implementation of Depth First Search and Breadth First Search

Pre-requisites:

- 64-bit Open source Linux or its derivative
- Programming Languages: C++/JAVA/PYTHON/R

Theory:

GRAPH TRAVERSALS

Graph traversal means visiting every vertex and edge exactly once in a well-defined order. While using certain graph algorithms, you must ensure that each vertex of the graph is visited exactly once. The order in which the vertices are visited are important and may depend upon the algorithm or question that you are solving.

During a traversal, it is important that you track which vertices have been visited. The most common way of tracking vertices is to mark them.

BREADTH FIRST SEARCH (BFS)

Breadth-first search (BFS) is an algorithm used for tree traversal on graphs or tree data structures. BFS can be easily implemented using recursion and data structures like dictionaries and lists.

BFS 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.

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

1. Visited
2. Not Visited

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

The algorithm works as follows:

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

The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

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

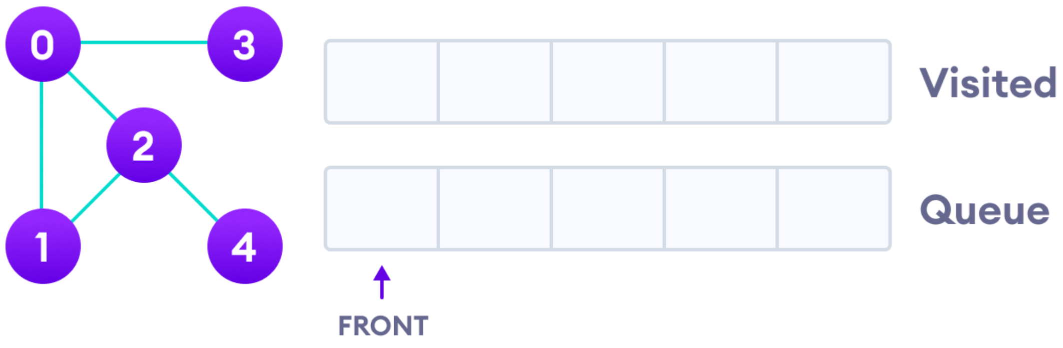


Figure 1 Undirected graph with 5 vertices

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

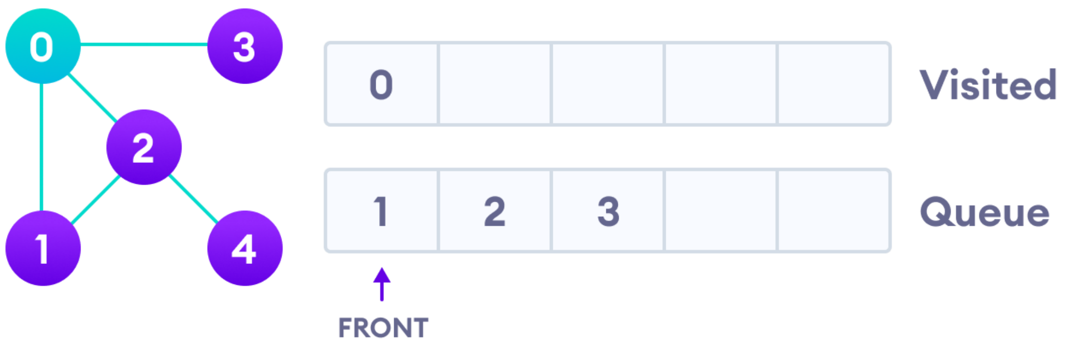


Figure 2 Visit start vertex and add its adjacent vertices to queue

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

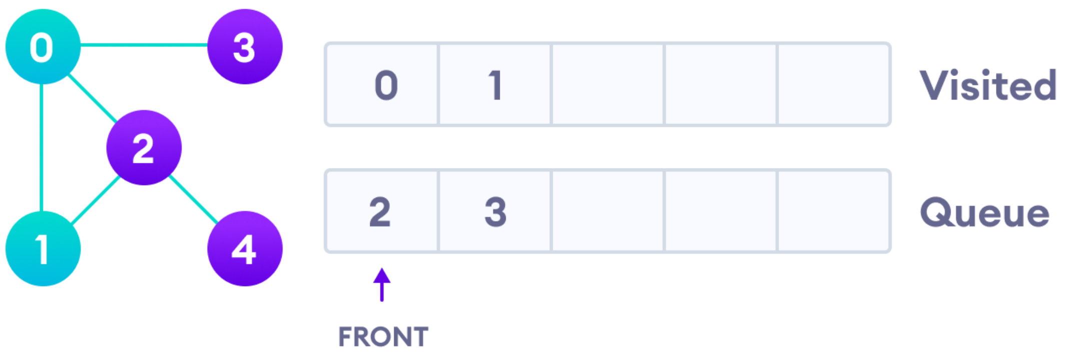


Figure 3 Visit the first neighbour of start node 0, which is 1

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

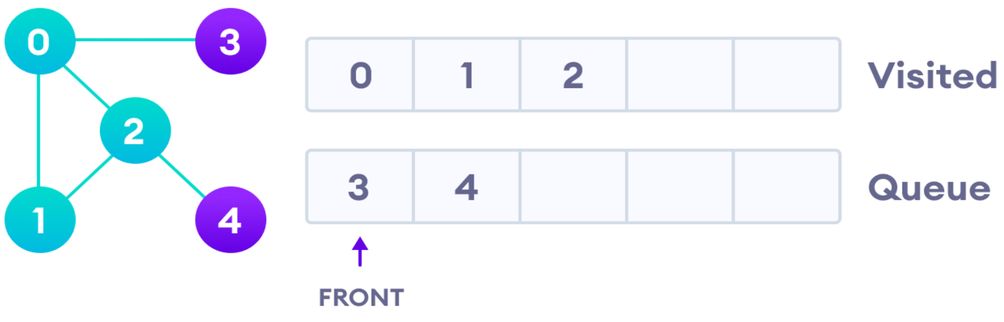


Figure 4 4 remains in the queue

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

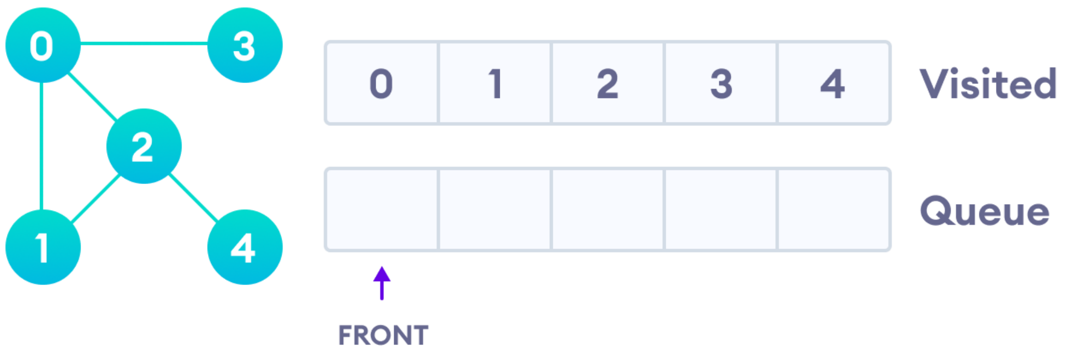


Figure 5 Visit last remaining item in the queue to check if it has unvisited neighbors

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

BFS pseudocode

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

BFS Algorithm Complexity

The time complexity of the BFS algorithm is represented in the form of O(V + E), where V is the number of nodes and E is the number of edges.The space complexity of the algorithm is O(V).

BFS Algorithm Applications

1. To build index by search index
2. For GPS navigation
3. Path finding algorithms
4. In Ford-Fulkerson algorithm to find maximum flow in a network
5. Cycle detection in an undirected graph

DEPTH FIRST SEARCH (DFS)

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

Depth First Search Algorithm

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

1. Visited
2. Not Visited

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

The DFS algorithm works as follows:

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

Depth First Search Example

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

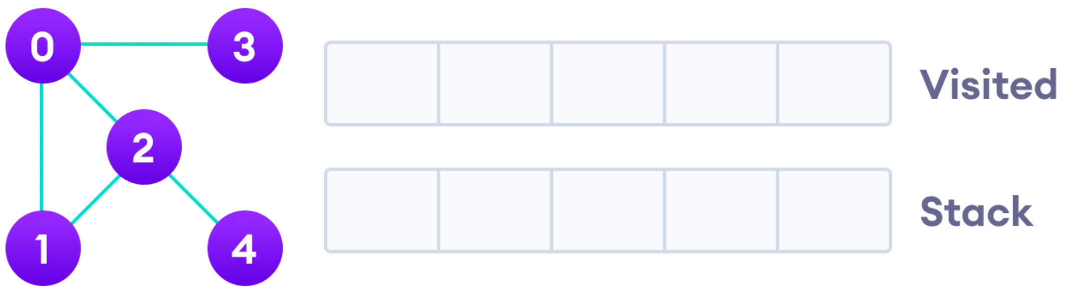


Figure 6 Undirected graph with 5 vertices

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

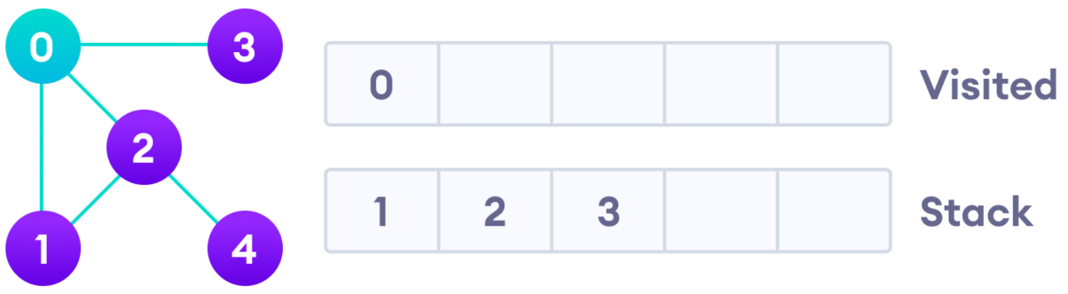


Figure 7 Visit the element and put it in the visited list

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

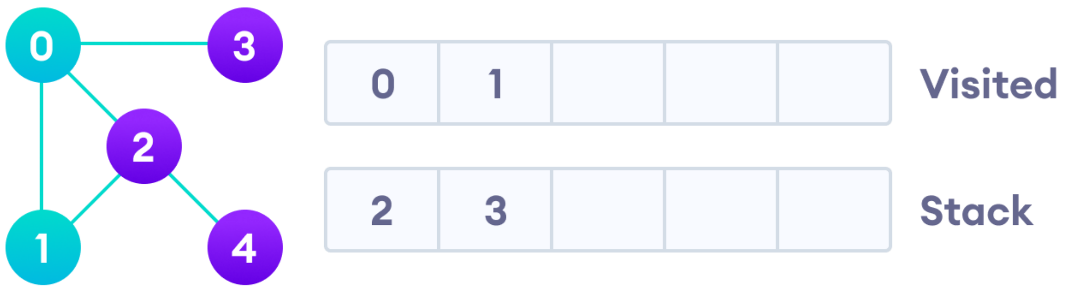


Figure 8 Visit the element at the top of stack

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

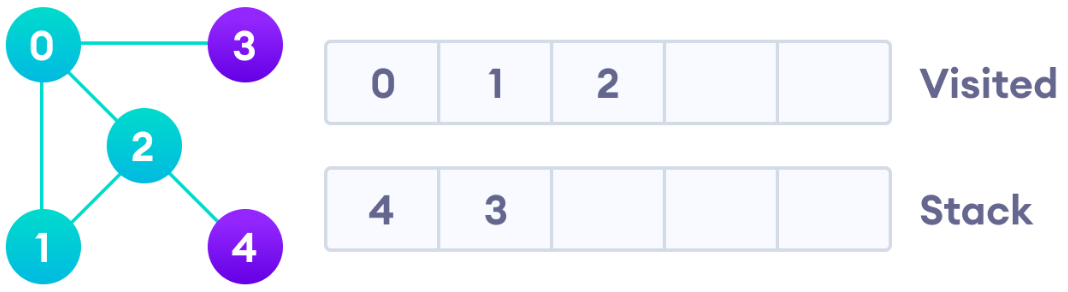


Figure 9 Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

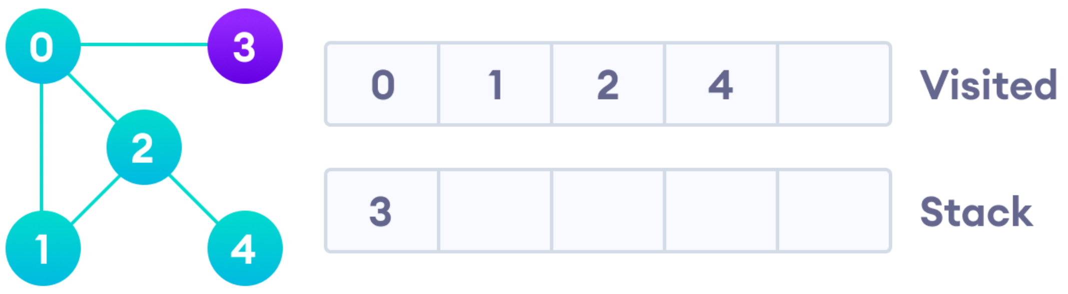


Figure 10Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

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

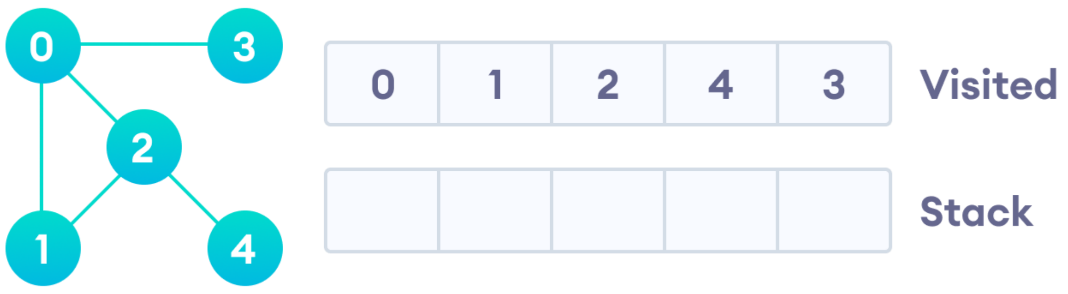


Figure 11 After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.

DFS Pseudocode (recursive implementation)

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

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

Complexity of Depth First Search

The time complexity of the DFS algorithm is represented in the form of O(V + E), where V is the number of nodes and E is the number of edges. The space complexity of the algorithm is O(V).

Application of DFS Algorithm

1. For finding the path
2. To test if the graph is bipartite
3. For finding the strongly connected components of a graph
4. For detecting cycles in a graph

Conclusion: We have implanted the BFS and DFS algorithm.

FAQ:

1. Explain DFS searching with example?
2. Explain BFS searching with example?
3. What is time complexity of DFS and BFS
4. What are applications of DFS and BFS

EXPERIMENT NO. 2

Aim: Implement A star Algorithm for any game search problem.

Objective:

- To understand Heuristic search and functions
- To understand working of A* algorithm

Theory:

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals. It is one which is the process of finding a path between multiple points, called "nodes". It enjoys widespread use due to its performance and accuracy. However, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance, although other work has found A* to be superior to other approaches.

Pseudo code:

```
OPEN //the set of nodes to be evaluated
CLOSED //the set of nodes already evaluated
add the start node to OPEN

loop
   current = node in OPEN with the lowest f_cost
   remove current from OPEN
   add current to CLOSED

   if current is the target node //path has been found
      return

   foreach neighbour of the current node
      if neighbour is not traversable or neighbour is in CLOSED
         skip to the next neighbour

      if new path to neighbour is shorter OR neighbour is not in OPEN
         set f_cost of neighbour
         set parent of neighbour to current
         if neighbour is not in OPEN
            add neighbour to OPEN
```

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

$$f(n) = g(n) + h(n), \text{ where}$$

- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

Tic-Tac-Toe:

• Without considering symmetry the search space is 9!; using symmetry the search space is 12 * 7! • A simple heuristic is the number of solution paths still open when there are 8 total paths (3 rows, 3 columns, 2 diagonals)

• Here is the search space using this heuristic

• The total search space is now reduced to about 40, depending on the opponents play

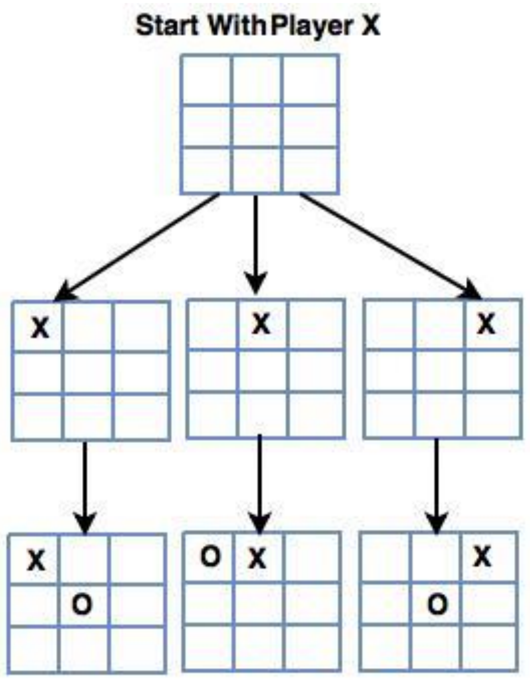
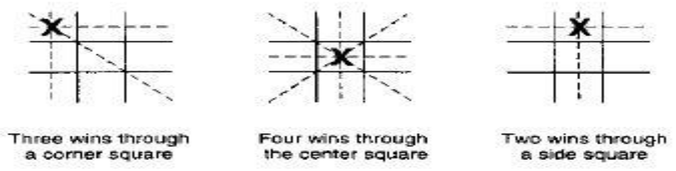
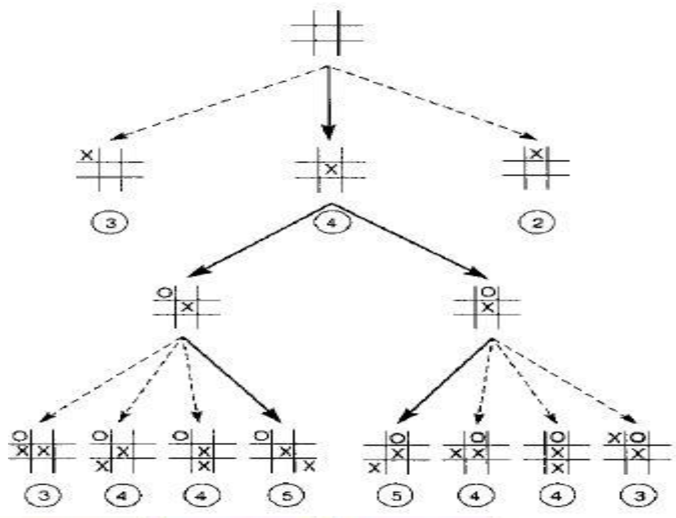


Fig (a): Initial Moves


Three wins through a corner square
Four wins through the center square
Two wins through a side square



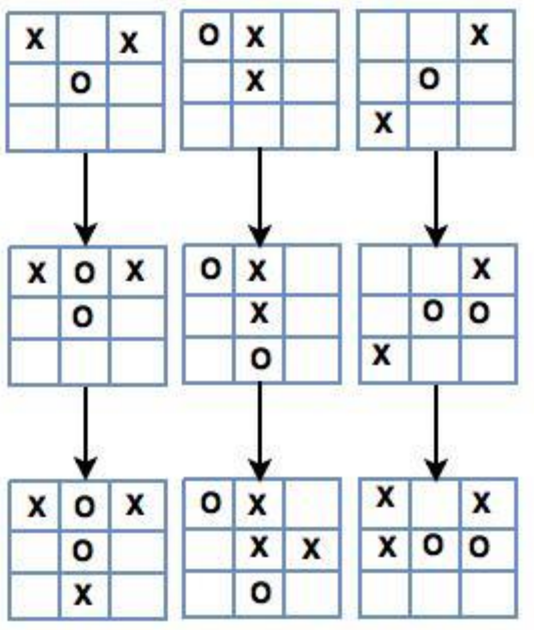


Fig (b): Intermediate Moves (Continue from Fig(a))

Tic-tac-toe seems dumb, but it actually requires you to *lookahead* one opponent's move to ensure that you will not loss. That is, you need to consider your opponent's move after your next move.

For example, suppose that the computer uses 'O'. At (D), the computer did not consider the opponent's next move and place at the corner (which is preferred over the side). At (E), the opponent was able to block the computer's winning move and create a fork.

```
 X |  |      X |  |      X |  |      X |  |      X |  | X
-----------   -----------   -----------   -----------   -----------
  |  |       | O |       | O |       | O |       | O |
-----------   -----------   -----------   -----------   -----------
  |  |       |  |       |  | X    O |  | X    O |  | X
   (A)         (B)         (C)         (D)         (E)
```

INPUT:

- Option to start the game i.e. Computer or a player
- Different positions of the cross or right marks of the players on the board.

OUTPUT:

- One of the player wins or computer wins or a play ends in draw.

CONCLUSION:

Hence we have successfully studied and implemented Tic Tac Toe using A* algorithm

FAQ:

1. How to calculate optimal path using A* algorithm?
2. What is mean by underestimation and overestimation in A*?

EXPERIMENT NO. 3

Aim: Implement Greedy Search Algorithm for any of the following application: Prim's Minimal Spanning Tree Algorithm

Objective:

- To study and implement greedy search algorithm: Prim's minimum spanning tree

Theory:

A greedy algorithm is an algorithmic strategy that makes the best optimal choice at each small stage with the goal of this eventually leading to a globally optimum solution. This means that the algorithm picks the best solution at the moment without regard for consequences.

Prim'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

Working of PRIM's Algorithm:

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 one vertex and keep adding edges with the lowest weight until we reach our goal.

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

1. Initialize the minimum spanning tree with a vertex chosen at random.
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
3. Keep repeating step 2 until we get a minimum spanning tree

Example of Prim's algorithm:

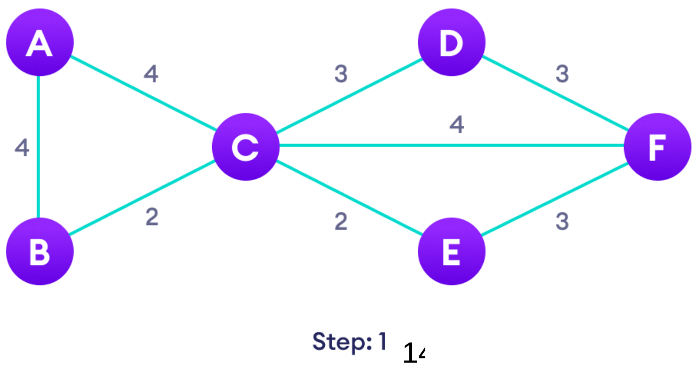


Figure 12 Start with a weighted graph

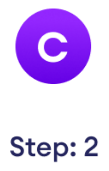


Figure 13 Choose a vertex

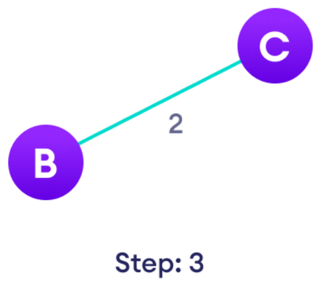


Figure 14 Choose the shortest edge from this vertex and add it

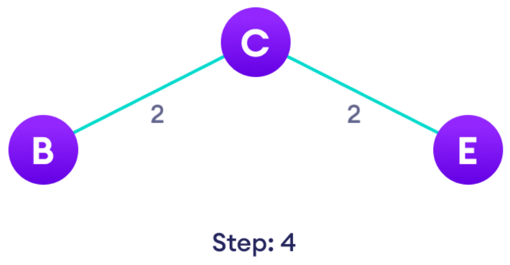


Figure 15 Choose the nearest vertex not yet in the solution

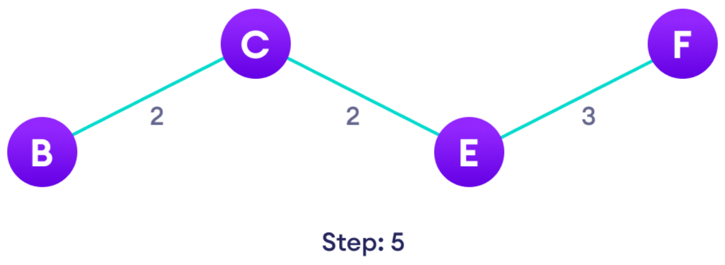


Figure 16 Choose the nearest edge not yet in the solution, if there are multiple choices, choose one at random

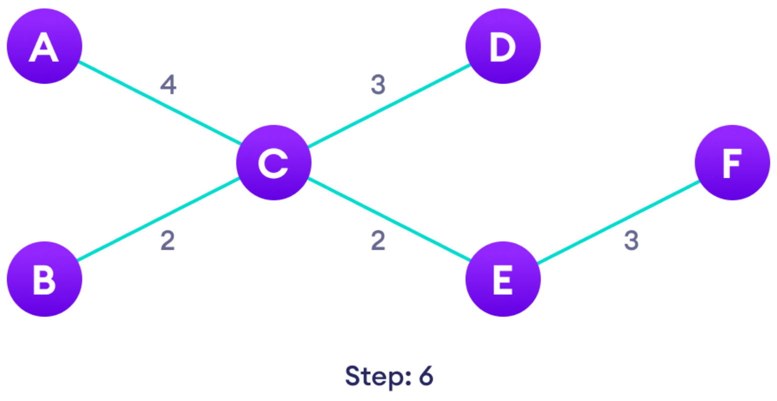


Figure 17 Repeat until you have a spanning tree

Prim's Algorithm Pseudocode:

```
T = ∅;
U = { 1 };
while (U ≠ V)
   let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
   T = T ∪ {(u, v)}
   U = U ∪ {v}
```

Prim's vs Kruskal's Algorithm

Kruskal's algorithm is another popular minimum spanning tree algorithm that uses a different logic to find the MST of a graph. Instead of starting from a vertex, Kruskal's algorithm sorts all the edges from low weight to high and keeps adding the lowest edges, ignoring those edges that create a cycle.

Prim's Algorithm Complexity

The time complexity of Prim's algorithm is O(E log V).

Prim's Algorithm Application

- Laying cables of electrical wiring
- In network designed
- To make protocols in network cycles

Conclusion: We have implemented Prim’s algorithm in python

FAQ:

1. What is mean by Greedy search algorithm?
2. Explain Prim’s algorithm with example
3. What is time complexity of Prim’s algorithm?
4. What are the applications of Prim’s algorithm?

EXPERIMENT NO. 4

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

Objective:

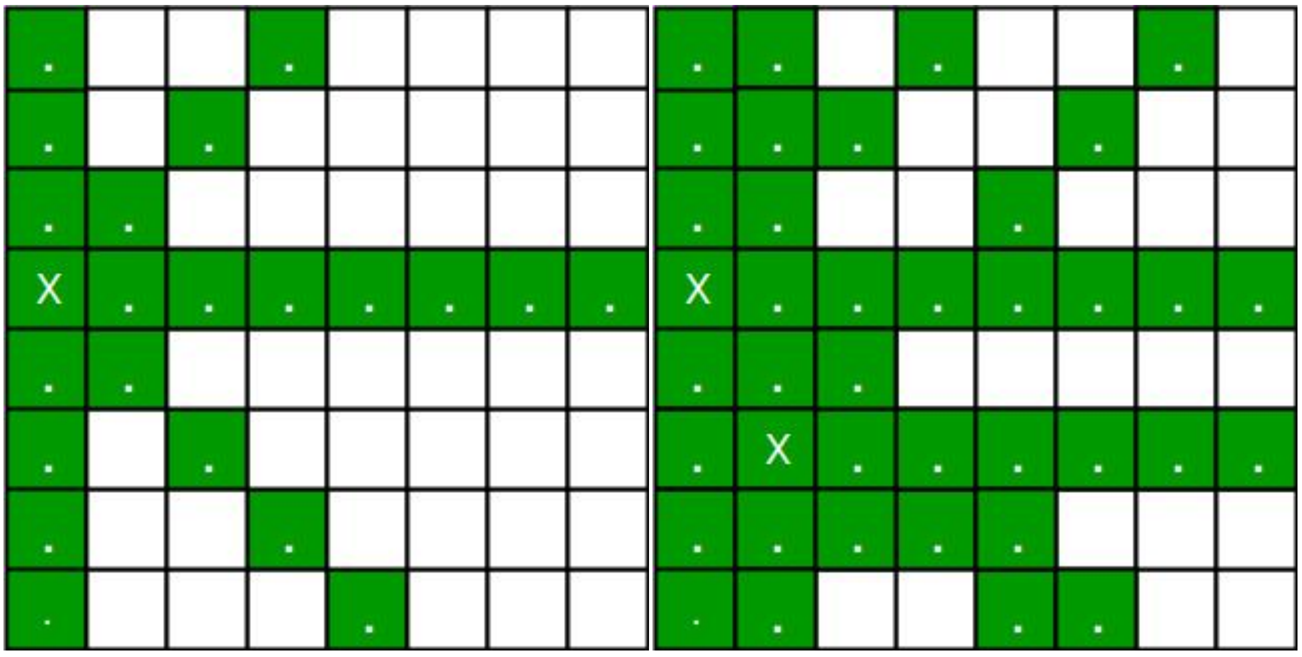
- To study and implement n-queens problem using Branch and Bound and backtracking

Theory:

The N queens puzzle is the problem of placing N chess queens on an N×N chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

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.

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



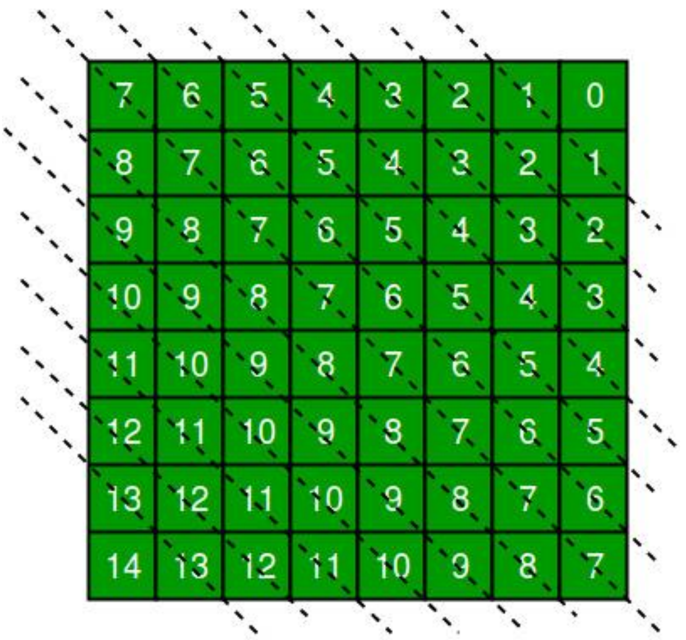
1. For the 1st Queen, there are total 8 possibilities as we can place 1st Queen in any row of first column. Let's place Queen 1 on row 3.
2. After placing 1st Queen, there are 7 possibilities left for the 2nd Queen. But wait, we don't really have 7 possibilities. We cannot place Queen 2 on rows 2, 3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

We need to figure out an efficient way of keeping track of which cells are under attack. In previous solution we kept an 8--by--8 Boolean matrix and update it each time we placed a queen, but that required linear time to update as we need to check for safe cells.
Basically, we have to ensure 4 things:
1. No two queens share a column.
2. No two queens share a row.
3. No two queens share a top-right to left-bottom diagonal.
4. No two queens share a top-left to bottom-right diagonal.

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

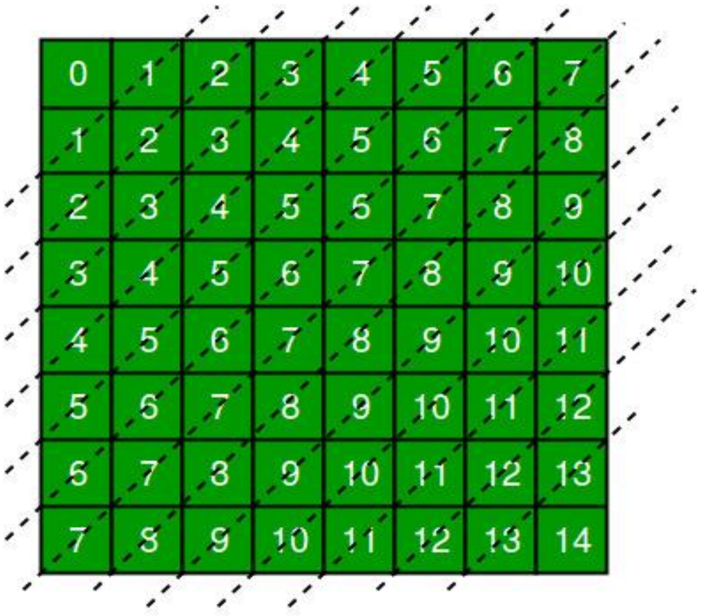
Lets do some pre-processing first. Let's create two N x N matrix one for / diagonal and other one for \ diagonal. Let's call them slashCode and backslashCode respectively. The trick is to fill them in such a way that two queens sharing a same /-diagonal will have the same value in matrix slashCode, and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



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

r -c + 7



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

r + c

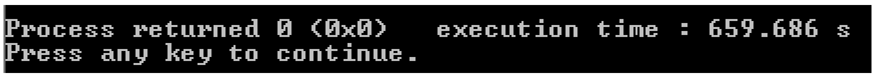
The 'N – 1' in the backslash code is there to ensure that the codes are never negative because we will be using the codes as indices in an array.

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

Performance:

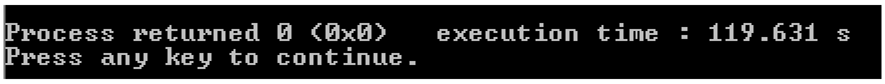
When run on local machines for N = 32, the backtracking solution took 659.68 seconds while above branch and bound solution took only 119.63 seconds. The difference will be even huge for larger values of N

Backtracking Algorithm



```
Process returned 0 (0x0)   execution time : 659.686 s
Press any key to continue.
```

Branch and Bound Algorithm



```
Process returned 0 (0x0)   execution time : 119.631 s
Press any key to continue.
```

<u>Conclusion</u>: We have implemented n-queen problem using branch and bound and backtracking

<u>FAQ</u>:

1. What is the difference between branch and bound and Backtracking?
2. Which algorithm provides faster output?
3. Elaborate n-queen problem with example?

EXPERIMENT NO. 5

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

Objectives:

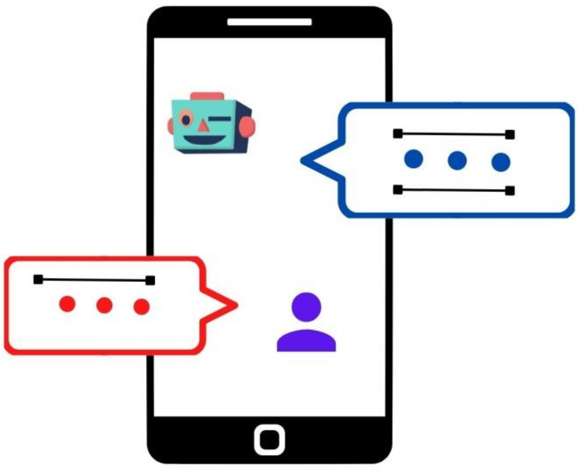
- To study and implement elementary chatbot.

Theory:

Are you fed up with waiting in long lines to speak with a customer support representative? Can you recall the last time you interacted with customer service? There's a chance you were contacted by a bot rather than human customer support professional. We will here discuss how to build a Chatbot using Python.

Bots are specially built software that interacts with internet users automatically. Bots are made up of algorithms that assist them in completing jobs. By auto-designed, we mean that they run on their own, following instructions, and therefore begin the conservation process without the need for human intervention.

Bots are responsible for the majority of internet traffic. For e-commerce sites, traffic can be significantly higher, accounting for up to 90% of total traffic. They can communicate with people and on social media accounts, as well as on websites.



Type of Bots

1. ChatBot — An Artificial Intelligence (AI) programme that communicates with users through app, message, or phone. It is most commonly utilised by Twitter.

2. Social Media Bot- Created for social media sites to answer automatically all at once.

3. Google Bot is commonly used for indexing and crawling. Spider Bots—Developed for retrieving data from websites, the Google Bot is widely used for indexing and crawling.

4. Spam Bots are programmed that automatically send spam emails to a list of addresses.

5. Transnational Bots are bots that are designed to be used in transactions.

6. Monitoring Bots – Creating bots to keep track of the system's or website's health.

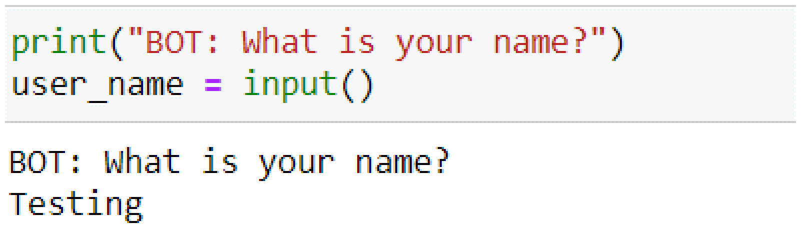
Build libraries should be avoided if you want to have a thorough understanding of how a chatbot operates in Python. In 1994, Michael Mauldin was the first to coin the term "chatterbot" as Julia.

Simple ChatBot build by using Python

A ChatBot is merely software by which humans can interact with each other. Examples include Apple's Siri, Amazon's Alexa, Google Assistant, and Microsoft's Cortana.

A. Interaction of User for asking the name

First, we will ask Username
print("BOT: What is your name?")
user_name = input()



```python
print("BOT: What is your name?")
user_name = input()
```

```
BOT: What is your name?
Testing
```

image source Jupyter Notebook

B. Response of ChatBot

To begin with, we are assigning questions and answers ChatBot must ask us, for example, we have assigned three variables with different answers, most important point to note it can be used in code and can be also updated automatically by just changing in values of variables. The format is used to pass the values easily.

```
name = "Bot Number 286"
monsoon = "rainy"
mood = "Smiley"
resp = {
"what's your name?": [
"They call me {0}".format(name),
"I usually go by {0}".format(name),
"My name is the {0}".format(name) ],
"what's today's weather?": [
"The weather is {0}".format(monsoon),
"It's {0} today".format(monsoon)],
"how are you?": [
"I am feeling {0}".format(mood),
"{0}! How about you?".format(mood),
"I am {0}! How about yourself?".format(mood), ],
"": [
"Hey! Are you there?",
```

C. Creating a Function Response

Library random imported to choose the response. It will select the answer by bot randomly instead of the same act.

```
import random
def res(message):
if message in resp:
        bot286_message =
random.choice(resp[message])
else:
        bot286_message =
random.choice(resp["default"])
return bot286_message
```

D. Another Function

Sometimes the questions added are not related to available questions, and sometimes some letters are forgotten to write in the chat. At that time, the bot will not answer any questions, but another function is forward.

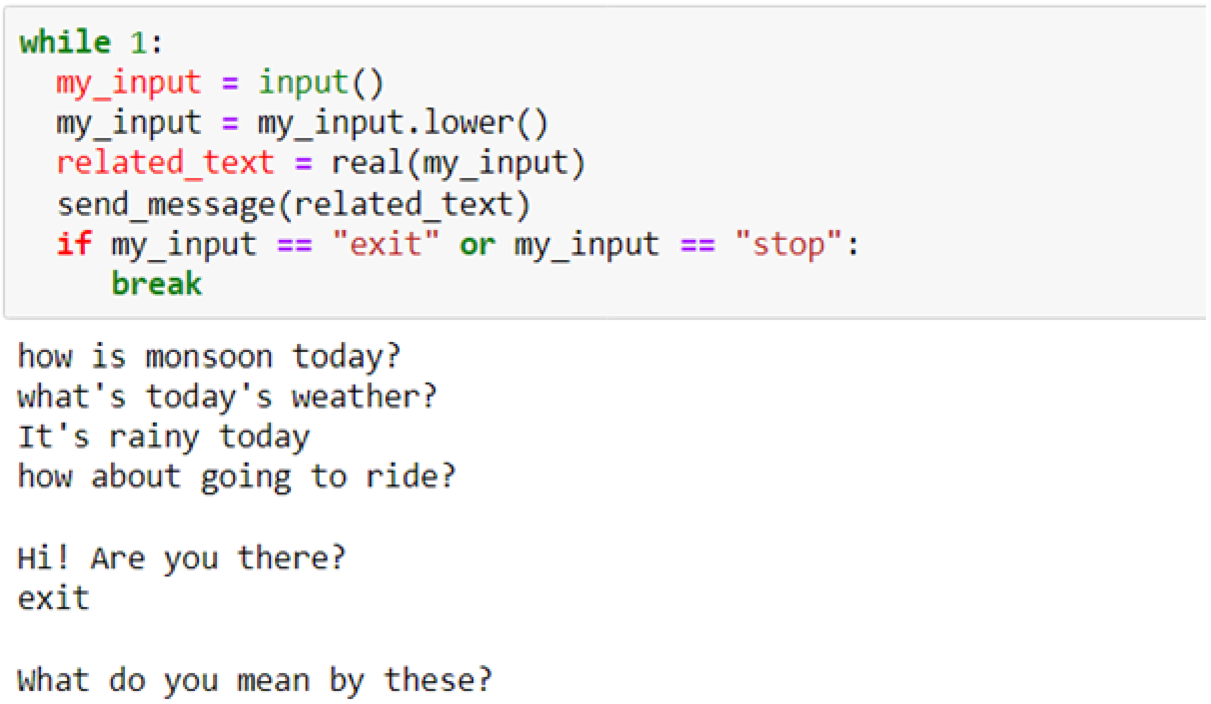
```
def real(xtext):
  if "name" in xtext:
        ytext = "what's your name?"
elif "monsoon" in xtext:
        ytext = "what's today's weather?"
elif "how are" in xtext:
        ytext = "how are you?"
else:
```

E. Sending back the message function

```
def
send_message(message):
print((message))
response = res(message)
print((response))
```

F. Final Step to break the loop

```
while 1:
my_input = input()
my_input = my_input.lower()
related_text = real(my_input)
send_message(related_text)
if my_input == "exit" or my_input == "stop":
break
```



```python
while 1:
  my_input = input()
  my_input = my_input.lower()
  related_text = real(my_input)
  send_message(related_text)
  if my_input == "exit" or my_input == "stop":
     break
```

```
how is monsoon today?
what's today's weather?
It's rainy today
how about going to ride?

Hi! Are you there?
exit

What do you mean by these?
```

image source Jupyter Notebook

Another example is by installing library chatterbot

Chatterbot:

Natural language Processing (NLP) is a necessary part of artificial intelligence that employs natural language to facilitate human-machine interaction.Python uses many libraries such as NLTK, spacy, etc. A chatbot is a computer program that can communicate with humans in a natural language. They frequently rely on machine learning, particularly natural language processing (NLP).

Use the following commands in Terminal in Anaconda prompt.

pip install chatterbot

pip install chatterbot_corpus

You can also install chatterbot from its
source: https://github.com/gunthercox/ChatterBot/archive/master.zip

After installing unzip, the file and open Terminal, and type in: cd

chatter_bot_master_directory. Finally, type python setup.py install.

For installing Spacy

```
pip install -U spacy
```

For downloading model “en_core_web_sm”

```
import spacy
from spacy.cli.download import download
download(model="en_core_web_sm")
```

or

pip install https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.2.0/en_core_web_sm-2.2.0.tar.gz

Testing installation

```
import spacy
nlp = spacy.blank("en")
doc = nlp("This is a sentence.")
```

You will need further two classes ChatBot and ListTrainers

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

First Example

```
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
# Create a new chatbot named Charlie
chatbot = ChatBot('name')
trainer = ListTrainer(chatbot)
trainer.train([
    "Hi, can I help you?",
    "Sure, I'd like to book a flight to Iceland.",
    "Your flight has been booked."
])
# Get a response to the input text 'I would like to book a flight.'
response = chatbot.get_response('I would like to book a flight.')
print(response)
```

Second example by ChatBot

```
Bot286 = ChatBot(name='PyBot')
talk = ['hi buddy!',
            'hi!',
            'how do you do?',
            'how are you?',
            'i'm fine.',
            'fine, you?',
            'always smiling.']
list_trainer = ListTrainer(Bot286)for item in (talk):
    list_trainer.train(item)
>>>print(my_bot.get_response("hi"))
 how do you do?
from chatterbot.trainers import ChatterBotCorpusTrainercorpus_trainer =
ChatterBotCorpusTrainer(Bot286)
 corpus_trainer.train('chatterbot.corpus.english')
```

Benefits of Bots –

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

Conclusion: We have implemented a chatbot using python.

FAQ:

1. what is chatterbot?
2. What are the applications of chatbot?

EXPERIMENT NO. 6

Aim: 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

Objective:

- To study and implement expert system using Prolog.

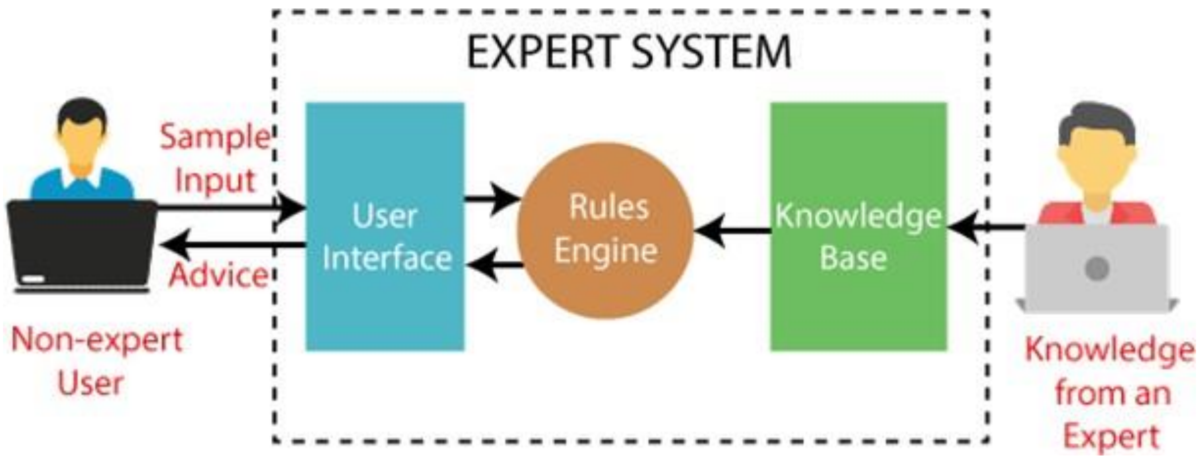
Theory:

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 compsex 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

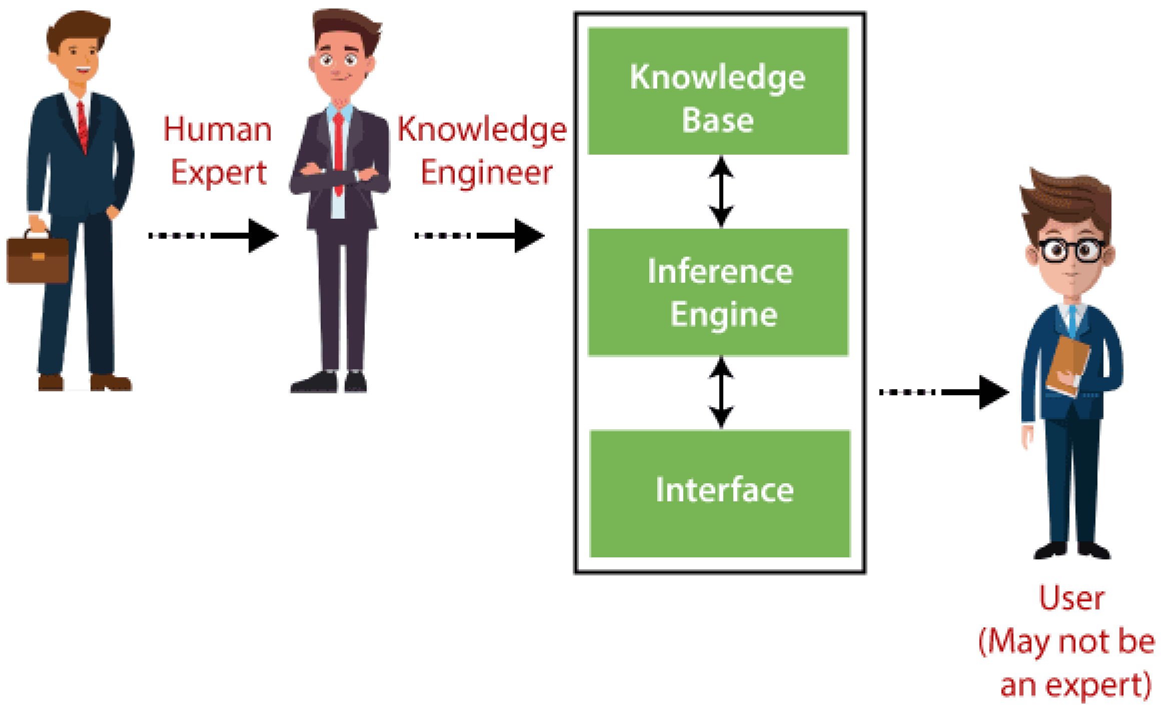
- **High Performance:** The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.
- **Understandable:** It responds in a way that can be easily understandable by the user. It can take input in human language and provides 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**



Participants in the development of Expert System

There are three primary participants in the building of Expert System:

1. **Expert:** The success of an ES much depends on the knowledge provided by human experts. These experts are those persons who are specialized in that specific domain.

2. **Knowledge Engineer:** Knowledge engineer is the person who gathers the knowledge from the domain experts and then codifies that knowledge to the system according to the formalism.
3. **End-User:** This is a particular person or a group of people who may not be experts, and working on the expert system needs the solution or advice for his queries, which are complex.

Advantages of Expert System

- These systems are highly reproducible.
- They can be used for risky places where the human presence is not safe.
- Error possibilities are less if the KB contains correct knowledge.
- The performance of these systems remains steady as it is not affected by emotions, tension, or fatigue.
- They provide a very high speed to respond to a particular query.

Limitations of Expert System

- The response of the expert system may get wrong if the knowledge base contains the wrong information.
- Like a human being, it cannot produce a creative output for different scenarios.
- Its maintenance and development costs are very high.
- Knowledge acquisition for designing is much difficult.
- For each domain, we require a specific ES, which is one of the big limitations.
- It cannot learn from itself and hence requires manual updates.

Applications of Expert System

- **In designing and manufacturing domain**

  It can be broadly used for designing and manufacturing physical devices such as camera lenses and automobiles.

- **In the knowledge domain**

  These systems are primarily used for publishing the relevant knowledge to the users. The two popular ES used for this domain is an advisor and a tax advisor.

- **In the finance domain**

  In the finance industries, it is used to detect any type of possible fraud, suspicious activity, and advise bankers that if they should provide loans for business or not.

- **In the diagnosis and troubleshooting of devices**

  In medical diagnosis, the ES system is used, and it was the first area where these systems were used.

- **Planning and Scheduling**

  The expert systems can also be used for planning and scheduling some particular tasks for achieving the goal of that task.

Expert Systems in Prolog

An expert system emulates the decision-making ability of a human *expert*.

Prolog is very well suited for *implementing* expert systems due to several reasons:

- Prolog itself can be regarded as a simple *inference engine* or theorem prover that derives conclusions from known rules. Very simple expert systems can be implemented by relying on Prolog's built-in search and backtracking mechanisms.
- Prolog data structures let us flexibly and conveniently *represent* rule-based systems that need additional functionality such as probabilistic reasoning.
- We can easily write meta-interpreters in Prolog to implement custom evaluation strategies of rules.

Example: Animal identification

Our aim is to write an expert system that helps us *identify animals*.

Suppose we have already obtained the following knowledge about animals, which are rules of inference:

- If it has a *fur* and says *woof*, then the animal is a dog.
- If it has a *fur* and says *meow*, then the animal is a cat.
- If it has *feathers* and says *quack*, then the animal is a duck.

These rules are not exhaustive, but they serve as a running example to illustrate a few points about expert systems.

The key idea of an expert system is to derive useful new information based on user-provided input.

Direct Prolog implementation

We now consider an implementation that uses **Prolog rules** *directly* to implement the mentioned inference rules.

This is straight-forward, using is_true/1 to emit a question and only proceeding with the current clause if the user input is the atom yes:

```
animal(dog)  :- is_true("has fur"), is_true("says woof").
animal(cat)  :- is_true("has fur"), is_true("says meow").
animal(duck) :- is_true("has feathers"), is_true("says quack").
```

```
is_true(Q) :-
        format("~s?\n", [Q]),
        read(yes).
```

There is a clear drawback of this approach, which is shown in the following sample interaction:

```
?- animal(A).
has fur?
|: yes.
says woof?
|: no.
has fur?
|: yes.
says meow?
|: yes.

A = cat .
```

The system has asked a question *redundantly*: Ideally, the fact that the animal *does* have a fur would have to be stated at most *once* by the user

Conclusion: We have implemented the Expert system using Prolog

FAQ:

1. What is an Expert system?
2. What are the components of expert system?
3. What is inference engine?
4. What is knowledge base?