SARASWATI Education Society's

SARASWATI College of Engineering

Learn Live Achieve and Contribute

Kharghar, Navi Mumbai - 410 210.

NAAC Accredited

Department of Computer Engineering

Name: Karan Kishor Buddhiwant

Roll No: 34

Subject: Artificial Intelligence (AI)

Class/Sem: T.E. / SEM-VI

SARASWATI Education Society's
SARASWATI College of Engineering
Learn Live Achieve and Contribute
Kharghar, Navi Mumbai - 410 210.
NAAC Accredited

Department of Computer Engineering

Subject: Artificial Intelligence **Class/Sem: TE/VI**

Name of Laboratory: Artificial Intelligence Lab **Year: 2021-2022**

LIST OF EXPERIMENTS

Prof. Neha Mahajan **Prof. Sujata Bhairnallykar**

Subject In-charge **HOD**

SARASWATI Education Society's

SARASWATI College of Engineering

Learn Live Achieve and Contribute

Kharghar, Navi Mumbai - 410 210.

NAAC Accredited

Department of Computer Engineering

Subject: Artificial Intelligence **Class/Sem: TE/VI**

Name of Laboratory: System Lab **Year: 2022-2023**

ASSIGNMENT INDEX

Prof. Neha Mahajan **Prof. Sujata Bhairnallykar**

Subject In-charge **H.O.D**

Experiment No. 1

Aim: A Case Study on Artificial Intelligence Application in Medical Diagnostics.

Abstract:

The main focus of Artificial Intelligence is to improvise the human cognitive capabilities. It brings about tremendous change in health-care, and the date that is being incessantly generated and the progress in the techniques to analyse them are the factors controlling the backbones of development in AI. Today, AI is playing an essential part in the advancement of the field of medicinal diagnostics. The review of AI applications in healthcare and medical diagnostics along with its future applications and effects The techniques are being applied to structure as well as unstructured medical data. Popular and effective AI systems incorporate machine techniques for structured data like neural network, classical support vector machine and deep learning, and NLP for unstructured data. Major AI techniques involve medical diagnostics including cancer, neurology and cardiology.

Keywords: Medical Diagnostics, Support Vector Machine, Deep Learning, NLP.

Introduction:

Medical Artificial Intelligence:

There is a strong likelihood that AI will help doctors make more reliable and error-free clinical decisions and in some cases give it more importance than human judgement in some important areas of medical care (eg. oncology). The four most important perspective of medical investigators[4]:

- The need of AI application in healthcare
- Data types to be analysed by AI systems
- Generating meaningful results using Artificial Intelligence through procedural mechanism
- Disease groups that can be dealt by AI systems

Need of Ai:

AI employs procedural algorithms to process large set of datasets in healthcare, and it further uses the results to come up to a final judgement to assist medical diagnostics. Learning and self-correcting capability can be added feature to improve its accuracy. Moreover, AI systems can provide updated medical information from journals, clinical practices with relatively less error.

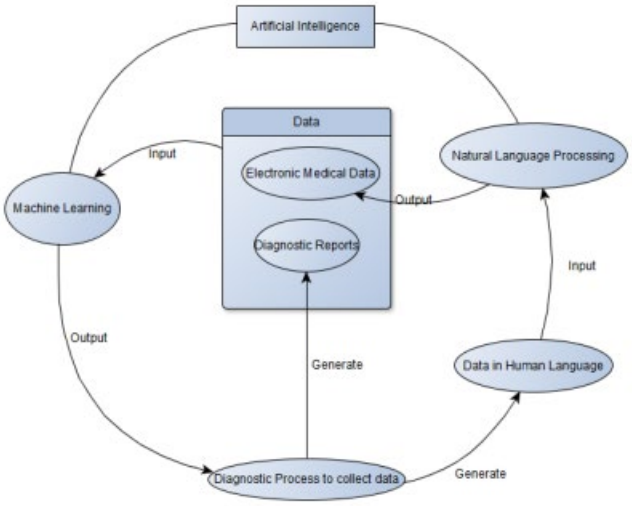
Dataset:

In the diagnosis stage of clinical dataset, a substantial proportion of data for the literature analysis of Artificial Intelligence is obtained from electro-diagnosis, diagnosis imaging and genetic testing. An example is, Ankita Sahu and Akhil Sahu encouraged radiologists into adopting Artificial Intelligence technology while analysing the huge data that is retrieved from diagnostic images. In order to diagnose gastric cancer Li et al tried to analyse the long noncoding RNAs and the abnormal genetic expression that is obtained from them. An

electrodiagnosis support system had been developed by Shin et al which is used for localising injury to the brain. There are two other major data sources which comprise of notes made from physical examination along with results obtained from running lab tests. It can be recognized through images, electrophysiological and genetic dataset since they carry a huge amount of clinical notes that can‟t be directly analysed. Hence the main focus of AI system is to convert these unstructured texts to system-understandable electronic medical data.

Ai Systems:

The AI system is divided into two major categories on the basis of dataset. The first category employs machine learning techniques to analyse structured data directly. On the other hand second category works on unstructured data using natural language processing methods to enhance the structured data. For instance in the medical application of AI, ML techniques works on patients‟ traits to cluster them or deduce the probability of any particular disease, whereas NLP strives at converting unstructured data to structured data that is further studied using Machine Learning techniques.



Current Applications of Ai in Medical Diagnostics

A considerable lot of the present machine learning analytic applications seems to fall under these classifications:

Chatbot:

Companies are utilizing AI-chatbots with speechrecognition feature that can recognize patterns through symptoms as stated by the patient to frame a potential conclusion, to avoid sickness and additionally suggest a suitable course of action.

Oncology: Scientists are trying to identify cancerous tissues using deep learning technique at a comparable level to any trained doctor, which can trigger fast diagnosis of cancer at its initial stages itself.

Pathology:

Pathology is an important area that if covered by AI systems can be of great help. It is the science or study of origin, nature and course of diseases analysed by lab tests of bodily fluids like sputum, blood and urine, and also through tissue analysis. The traditional methods of diagnosis involve use of microscope which can be timetaking and also at times can have error. The AI techniques like machine learning and machine vision techniques tend to enhance the traditional methods used by pathologists.

Future of Ai in Medical Diagnosis:

There is a of diversion of doctor from traditional method to the use of AI chatbots that is AI form of doctor and a UK based digital healthcare organisation is studying this collaboration of patients with AI doctors. Ai is becoming popular in medical services and is in the process of evolving the traditional method to a new way of diagnosing and treating ailments through the fast-growing field of machine learning and big data analytics.

Conclusion and Discussion:

An effective AI framework must have the ML component to handle structured dataset (genetic information, EP information, images) and NLP segment for extracting unstructured data's. Advanced algorithms at that point should be prepared using healthcare data so that it can help doctors with illness analysis and proposing methods for treatment.

In spite of the fact that the AI innovations are drawing in generous considerations in therapeutic research, implementation in real-life is yet confronting barriers. The very first problem originates from regulations. The safety needs to be surveyed and the adequacy of AI frameworks neds to be maintained which the current regulations lack. US FDA has made the primary endeavour to give regulations to evaluate AI systems in order to overcome the problems.

The second problem is information exchange. To function properly, AI frameworks should be prepared (ceaselessly) by information from clinical examinations. After initial raining when the AI framework is deployed, supply of information continuously turns into a critical concern for advance advancement and change in the framework. Incentives for sharing information related to framework are not given by current healthcare condition. By the by, a healthcare revolution is yet to approach in order to invigorate information that is being shared in the USA. While there is much chance, AI in therapeutic diagnostics is still a moderately new approach, with numerous clinicians still left to be persuaded of its unwavering quality, affectability and how it will be basically coordinated into clinical practice without undermining clinical expertise.

Experiment No. 2

AIM: Write a program to implement Breadth First Search.

RESOURCES REQUIRED: Visual Studio Code, Python.

THEORY:

Breadth First Search:

Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

ALGORITHM:

Step 1**:** SET STATUS = 1 (ready state) for each node in G

Step 2**:** Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3**:** Repeat Steps 4 and 5 until QUEUE is empty

Step 4**:** Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5**:** Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2(waiting state)

[END OF LOOP]

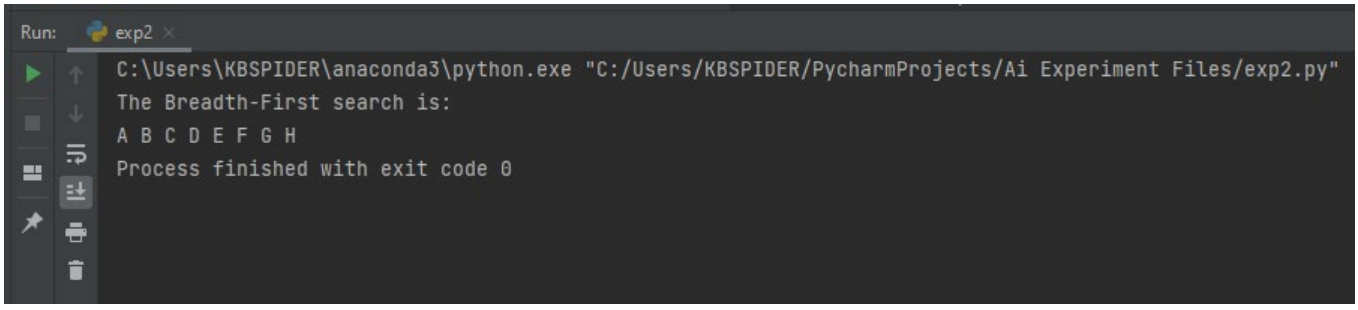
Step 6**:** EXIT

CODE:

```
graph = {
  'A':['B','C'],
  'B':['D','E'],
  'C':['F'],
  'D':[],
  'E':[] ,
  'F':['G','H'],
  'G':[],'H':[]   }
```

```
visited = []
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)

print("The Breadth-First search is: ")
bfs(visited, graph, 'A')
```

OUTPUT:



CONCLUSION: Hence we have successfully implemented the Breadth First Search Algorithm.

Experiment No. 3

AIM: Write a program to implement Depth First Search.

RESOURCES REQUIRED: Visual Studio Code, Python.

THEORY:

Depth First Search:

Depth-first search 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. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.

ALGORITHM:

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]

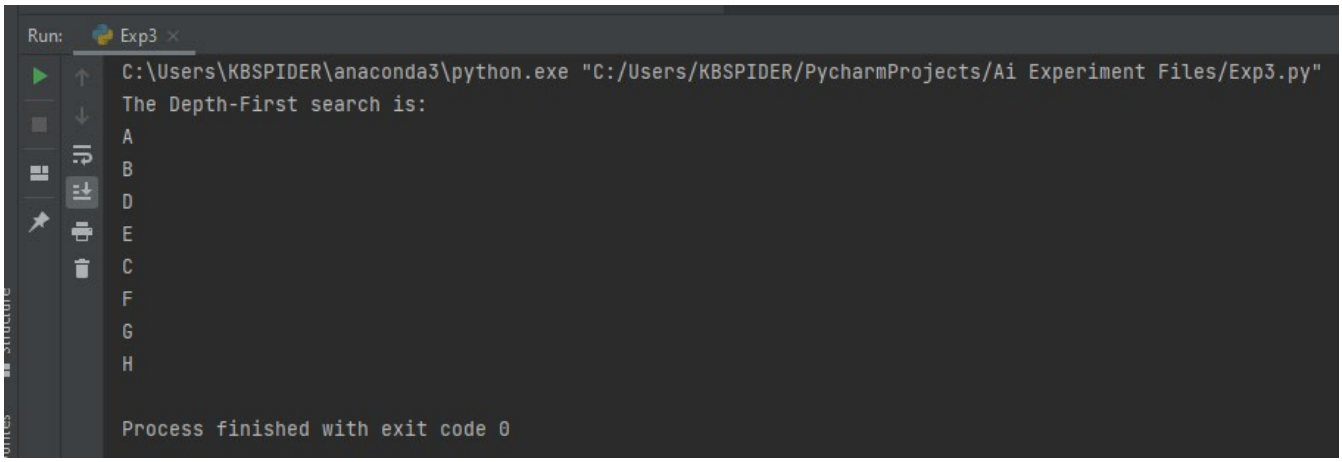
Step 6: EXIT

CODE:

```
graph = {
  'A':['B','C'],
  'B':['D','E'],
  'C':['F'],
  'D':[],
  'E':[] ,
  'F':['G','H'],
  'G':[],'H':[]   }
visited = []
def dfs(visited, graph, node):
```

```
    if node not in visited:
        print(node)
        visited.append(node)
        for neighbour in graph[node]:
            dfs(visited,graph,neighbour)


print("The Depth-First search is: ")
dfs(visited, graph, 'A')
```

OUTPUT:



CONCLUSION: Hence we have successfully implemented the Depth First Search Algorithm.

Experiment No. 4

AIM: Write a program to implement Uniform Cost Search.

RESOURCES REQUIRED: Visual Studio, Python.

THEORY:

Uniform Cost Search:

Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

Algorithm:

Insert the root into the queue

While the queue is not empty

Dequeue the maximum priority element from the queue

(If priorities are same, alphabetically smaller path is chosen)

If the path is ending in the goal state, print the path and exit

Else

Insert all the children of the dequeued element, with the cumulative costs as priority

Complexity: O(m^(1+floor(l/e)))
where,
m is the maximum number of neighbours a node has
l is the length of the shortest path to the goal state
e is the least cost of an edge

Advantages:

Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

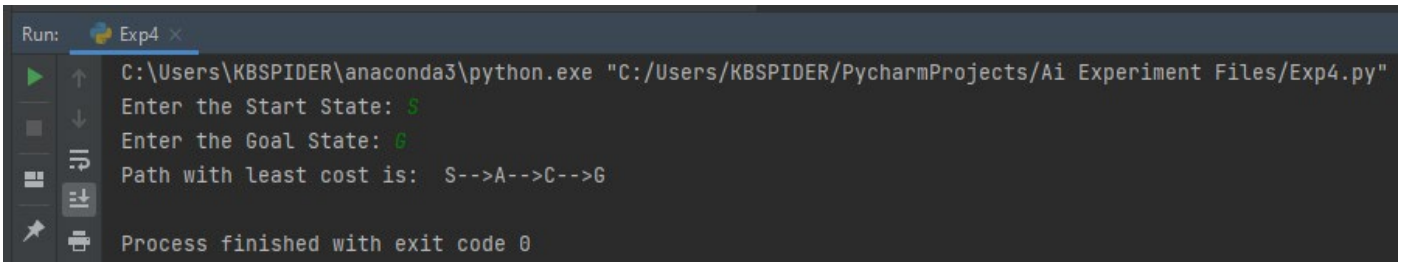
It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

CODE:

```
graph=[
    ['S','A',1],
    ['S','G',12],
    ['A','B',3],
    ['A','C',1],
    ['B','D',3],
    ['C','D',1],
    ['C','G',2],
    ['D','G',3]
]
temp=[]
temp1=[]
for i in graph:
    temp.append(i[0])
    temp1.append(i[1])
nodes=set(temp).union(set(temp1))
def UCS(graph,cost,open,closed,cur_node):
    if cur_node in open:
        open.remove(cur_node)
    closed.add(cur_node)
    for i in graph:
        if(i[0]==cur_node and costs[i[0]]+i[2]<costs[i[1]]):
            open.add(i[1])
            costs[i[1]]=costs[i[0]]+i[2]
            path[i[1]]=path[i[0]]+'-->'+i[1]
    costs[cur_node]=999999
    small=min(costs,key=costs.get)
    if small not in closed:
        UCS(graph,costs,open,closed,small)
costs=dict()
```

```
temp_cost=dict()
path=dict()
for i in nodes:
    costs[i]=999999
    path[i]=''
open=set()
closed=set()
start_node=input("Enter the Start State: ")
open.add(start_node)
path[start_node]=start_node
costs[start_node]=0
UCS(graph,costs,open,closed,start_node)
goal_node=input("Enter the Goal State: ")
print("Path with least cost is: ",path[goal_node])
```

OUTPUT:



CONCLUSION: Hence we have successfully completed implementing Uniform Cost Search Algorithm.

Experiment No. 5

AIM: Write a program to implement Greedy Best First Search.

RESOURCES REQUIRED: Visual Studio, Python.

THEORY:

Greedy Best First Search:

The informed search algorithm is also called heuristic search or directed search. In contrast to uninformed search algorithms, informed search algorithms require details such as distance to reach the goal, steps to reach the goal, cost of the paths which makes this algorithm more efficient. Here, the goal state can be achieved by using the heuristic function. The heuristic function is used to achieve the goal state with the lowest cost possible. This function estimates how close a state is to the goal.

Greedy best-first search uses the properties of both depth-first search and breadth-first search. Greedy best-first search traverses the node by selecting the path which appears best at the moment. The closest path is selected by using the heuristic function.

Algorithm:

Step 1: Place the starting node into the OPEN list.

Step 2: If the OPEN list is empty, Stop and return failure.

Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.

Step 4: Expand the node n, and generate the successors of node n.

Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.

Step 7: Return to Step 2.

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.
Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$.
Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

Advantages:

Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms. This algorithm is more efficient than BFS and DFS algorithms.

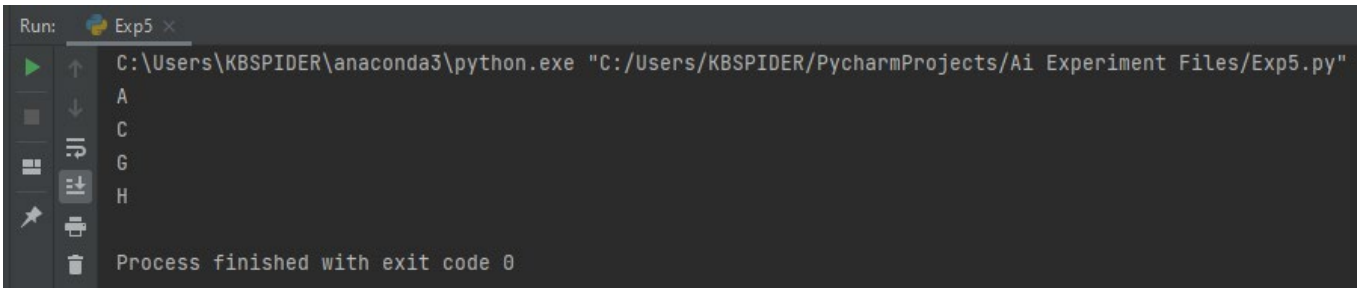
Disadvantages:

It can behave as an unguided depth-first search in the worstcase scenario. It can get stuck in a loop as DFS. This algorithm is not optimal.

CODE:

```
graph = {
'A':[('B',12), ('C',4)],
'B':[('D',7), ('E',3)],
'C':[('F',8), ('G',2)],
'D':[],
'E':[('H',0)],
'F':[('H',0)],
'G':[('H',0)]
}
def bfs(start, target, graph, queue=[], visited=[]):
    if start not in visited:
        print(start)
        visited.append(start)

    queue=queue+[x for x in graph[start] if x[0][0] not in visited]
    queue.sort(key=lambda x:x[1])
    if queue[0][0]==target:
        print(queue[0][0])
    else:
        processing=queue[0]
        queue.remove(processing)
        bfs(processing[0], target, graph, queue, visited)
bfs('A', 'H', graph)
```

OUTPUT:



CONCLUSION: Hence we have successfully completed implementing Greedy Best First Search Algorithm.

Experiment No. 6

AIM: Write a program to implement A* Algorithm.

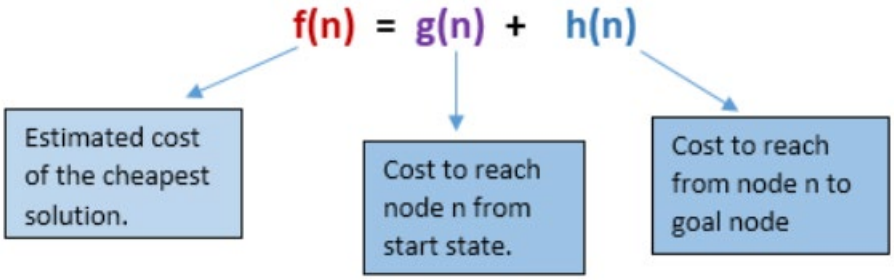
RESOURCES REQUIRED: Visual Studio, Python.

THEORY:

A* Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



Algorithm:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise.

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to Step 2**.**

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

Complete: A* algorithm is complete as long as: Branching factor is finite and Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

Admissible: the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.

Consistency: Second required condition is consistency for only A* graph-search.

Advantages:

A* search algorithm is the best algorithm than other search algorithms. A* search algorithm is optimal and complete. This algorithm can solve very complex problems.

Disadvantages:

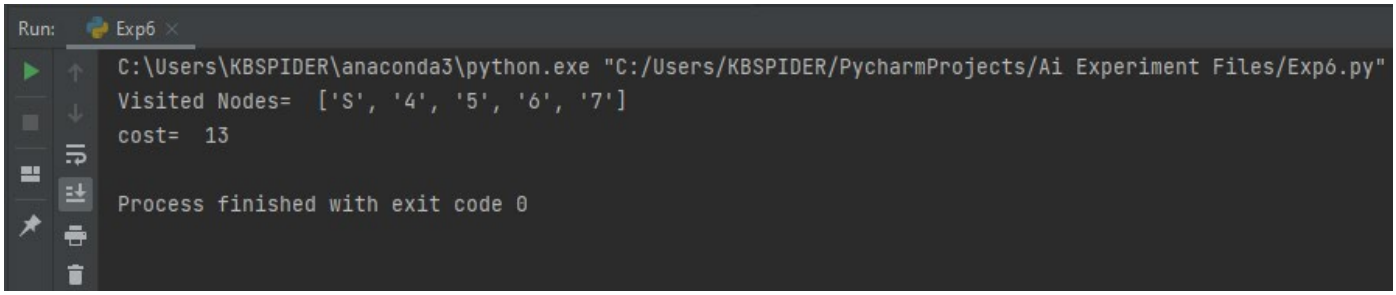
It does not always produce the shortest path as it mostly based on heuristics and approximation. A* search algorithm has some complexity issues. The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

CODE:

```
graph={
 'S':[('1',3,24),('4',4, 22)],
 '1':[('2',1,20),('4',5,22)],
 '2':[('3',4,18),('5',5,20)],
 '3':[],
 '4':[('1',5,24),('5',2,20)],
 '5':[('2',5,20),('6',4,20)],
 '6':[('7',3,0)],
 '7':[]
}
def astar(start,stop,openList=[],closedList=[], costs=0):
    if start not in closedList:
        closedList.append(start)
```

```
    for neighbor in graph[start]:
        if neighbor[0] not in closedList:
            openList.append(neighbor)
            openList.sort(key=lambda neighbor:neighbor[1]+neighbor[2])

    if openList[0][0]==stop:
        costs+=openList[0][1]+openList[0][2]
        closedList.append(openList[0][0])
        print("Visited Nodes= ",closedList,"\ncost= ",costs)
    else:
        value=openList[0]
        closedList.append(value[0])
        costs+=value[1]
        openList.remove(value)
        astar(value[0],stop,openList,closedList,costs)
astar('S','7')
```

OUTPUT:



CONCLUSION: Hence we have successfully completed implementing A* Algorithm.

Experiment No. 7

AIM: To learn the basics of Prolog and installation of SWI Prolog.

RESOURCES REQUIRED: SWI Prolog.

THEORY:

Basics of Prolog:

Prolog as the name itself suggests, is the short form of Logical Programming. It is a logical and declarative programming language. Before diving deep into the concepts of Prolog, let us first understand what exactly logical programming is.

Logic Programming is one of the Computer Programming Paradigm, in which the program statements express the facts and rules about different problems within a system of formal logic. Here, the rules are written in the form of logical clauses, where head and body are present. For example, H is head and B1, B2, B3 are the elements of the body. Now if we state that "H is true, when B1, B2, B3 all are true", this is a rule. On the other hand, facts are like the rules, but without any body. So, an example of fact is "H is true".

Some logic programming languages like Datalog or ASP (Answer Set Programming) are known as purely declarative languages. These languages allow statements about what the program should accomplish. There is no such step-by-step instruction on how to perform the task. However, other languages like Prolog, have declarative and also imperative properties. This may also include procedural statements like "To solve the problem H, perform B1, B2 and B3".

Prolog or Programming in Logics is a logical and declarative programming language. It is one major example of the fourth generation language that supports the declarative programming paradigm. This is particularly suitable for programs that involve symbolic or non-numeric computation. This is the main reason to use Prolog as the programming language in Artificial Intelligence, where symbol manipulation and inference manipulation are the fundamental tasks. In Prolog, we need not mention the way how one problem can be solved, we just need to mention what the problem is, so that Prolog automatically solves it. However, in Prolog we are supposed to give clues as the solution method. Prolog language basically has three different elements:

Facts: The fact is predicate that is true, for example, if we say, "Tom is the son of Jack", then this is a fact.

Rules: Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met. For example, if we define a rule as:

grandfather(X, Y) :- father(X, Z), parent(Z, Y) This implies that for X to be the grandfather of Y, Z should be a parent of Y and X should be father of Z.

Facts: We can define fact as an explicit relationship between objects, and properties these objects might have. So facts are unconditionally true in nature. Suppose we have some facts as given below:

- Tom is a cat
- Kunal loves to eat Pasta
- Hair is black
- Nawaz loves to play games
- Pratyusha is lazy.

So these are some facts, that are unconditionally true. These are actually statements, that we have to consider as true. Following are some guidelines to write facts:

- Names of properties/relationships begin with lower case letters.
- The relationship name appears as the first term.
- Objects appear as comma-separated arguments within parentheses.
- A period "." must end a fact.
- Objects also begin with lower case letters. They also can begin with digits (like 1234), and can be strings of characters enclosed in quotes e.g. color('pink', 'red').
- phoneno(agnibha, 1122334455). is also called a predicate or clause.

Syntax: The syntax for facts is as follows:

relation(object1,object2...)

Example: Following is an example of the above concept

cat(tom).

loves_to_eat(kunal,pasta).

of_color(hair,black).

loves_to_play_games(nawaz).

lazy(pratyusha).

Rules: We can define rule as an implicit relationship between objects. So facts are conditionally true. So when one associated condition is true, then the predicate is also true. Suppose we have some rules as given below:

- Lili is happy if she dances.
- Tom is hungry if he is searching for food.
- Jack and Bili are friends if both of them love to play cricket.
- will go to play if school is closed, and he is free.

So these are some rules that are conditionally true, so when the right hand side is true, then the left hand side is also true.

Here the symbol (:-) will be pronounced as “If”, or “is implied by”. This is also known as neck symbol, the LHS of this symbol is called the Head, and right hand side is called Body. Here we can use comma (,) which is known as conjunction, and we can also use semicolon, that is known as disjunction.

Syntax:

rule_name(object1, object2, ...) :- fact/rule(object1, object2, ...)

Suppose a clause is like :

P :- Q;R.

This can also be written as

P :- Q.

P :- R.

If one clause is like :

P :- Q,R;S,T,U.

Is understood as

P :- (Q,R);(S,T,U).

Or can also be written as:

P :- Q,R.

P :- S,T,U.

Example:

happy(lili) :- dances(lili).

hungry(tom) :- search_for_food(tom).

friends(jack, bili) :- lovesCricket(jack), lovesCricket(bili).

goToPlay(ryan) :- isClosed(school), free(ryan).

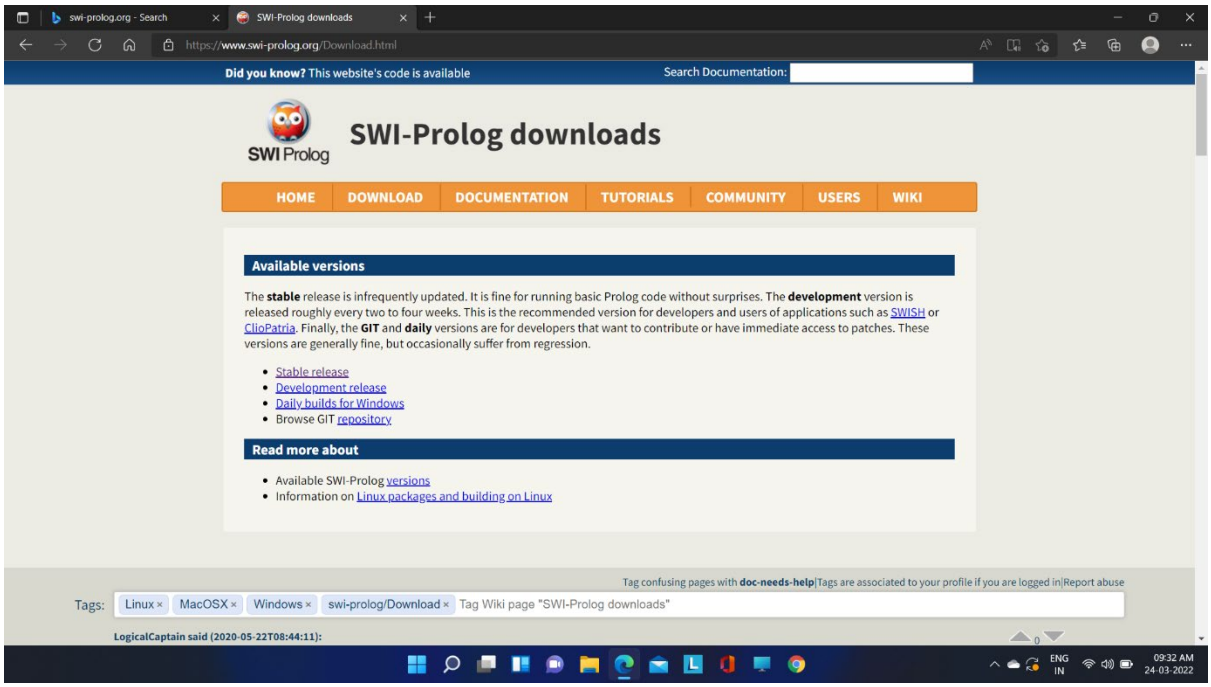
Queries: Queries are some questions on the relationships between objects and object properties. So question can be anything, as given below:

- Is tom a cat?
- Does Kunal love to eat pasta?
- Is Lili happy?
- Will Ryan go to play?

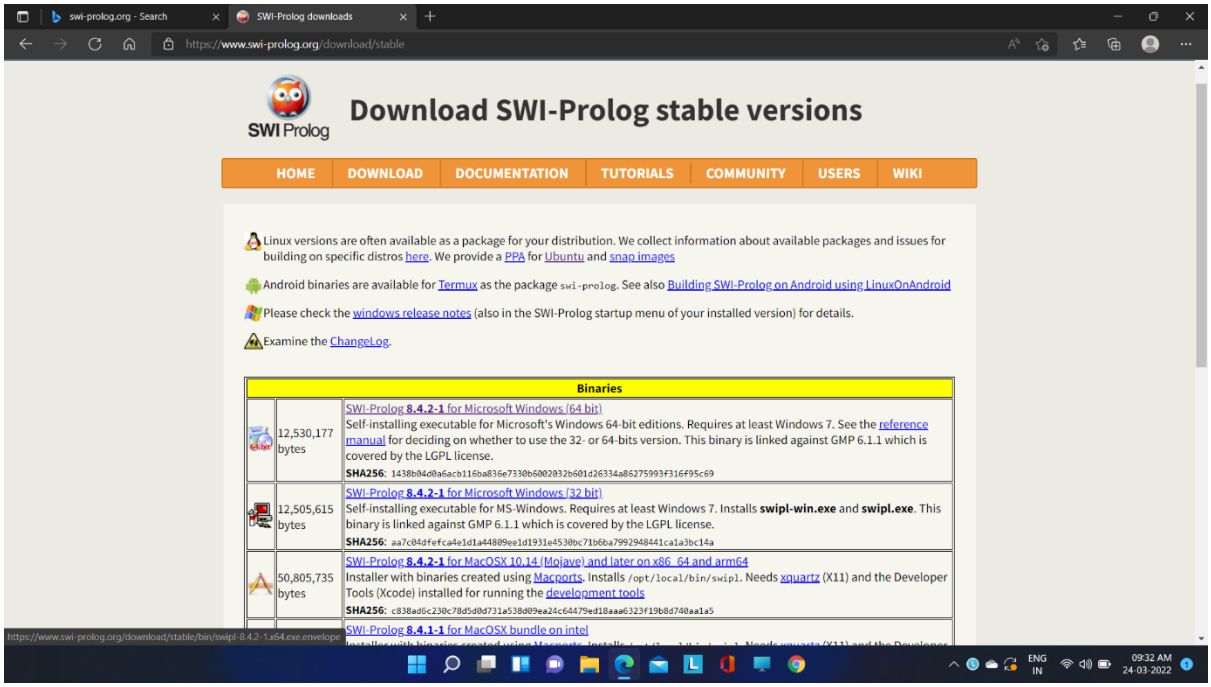
So according to these queries, Logic programming language can find the answer and return them.

IMPLEMENTATION:

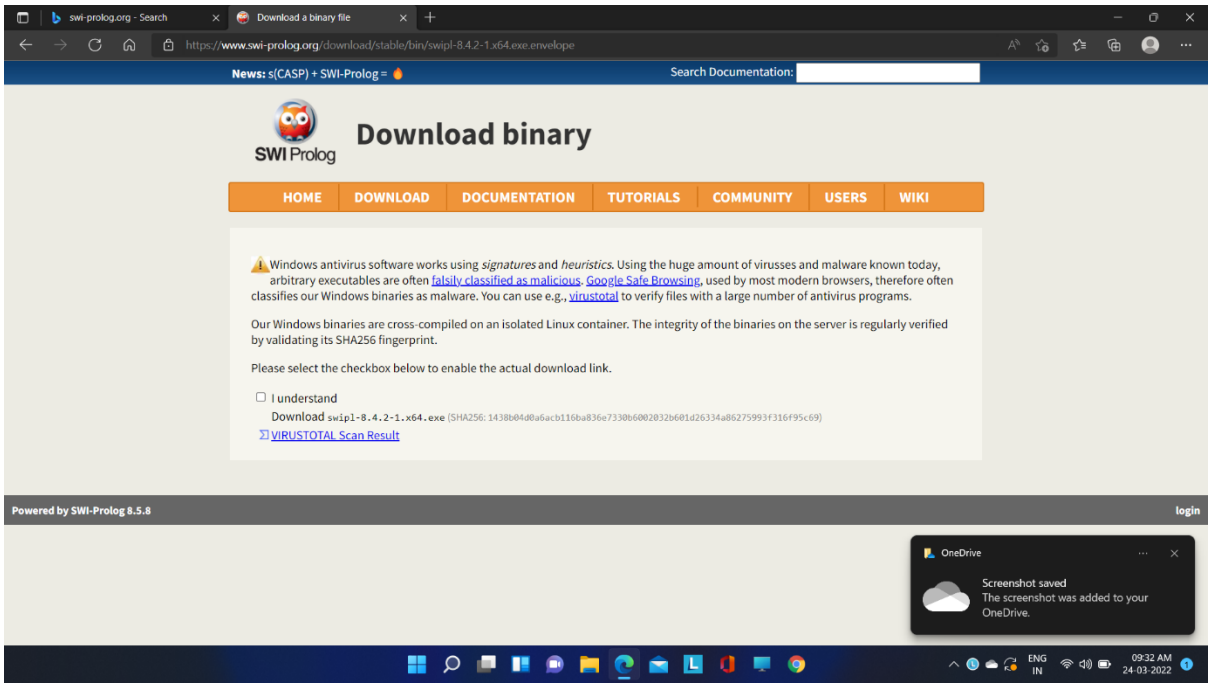
Visit the website https://www.swi-prolog.org/ and download the SWI Prolog. Download the Stable Version.



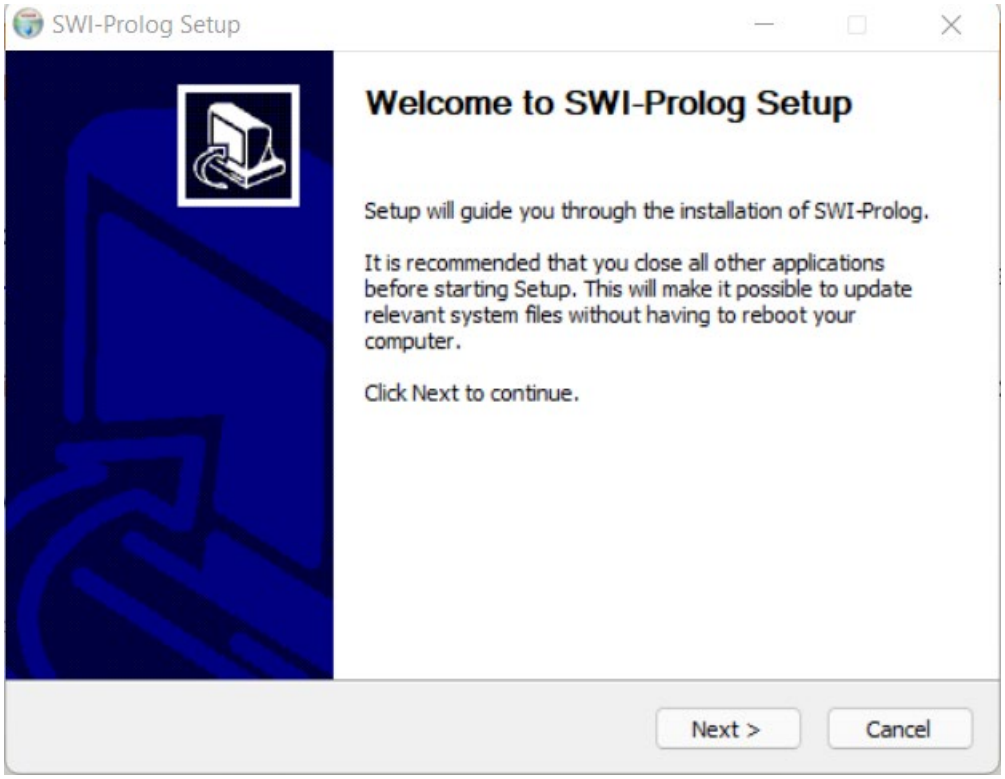
Install the SWI Prolog for Microsoft Windows of 64 bit.

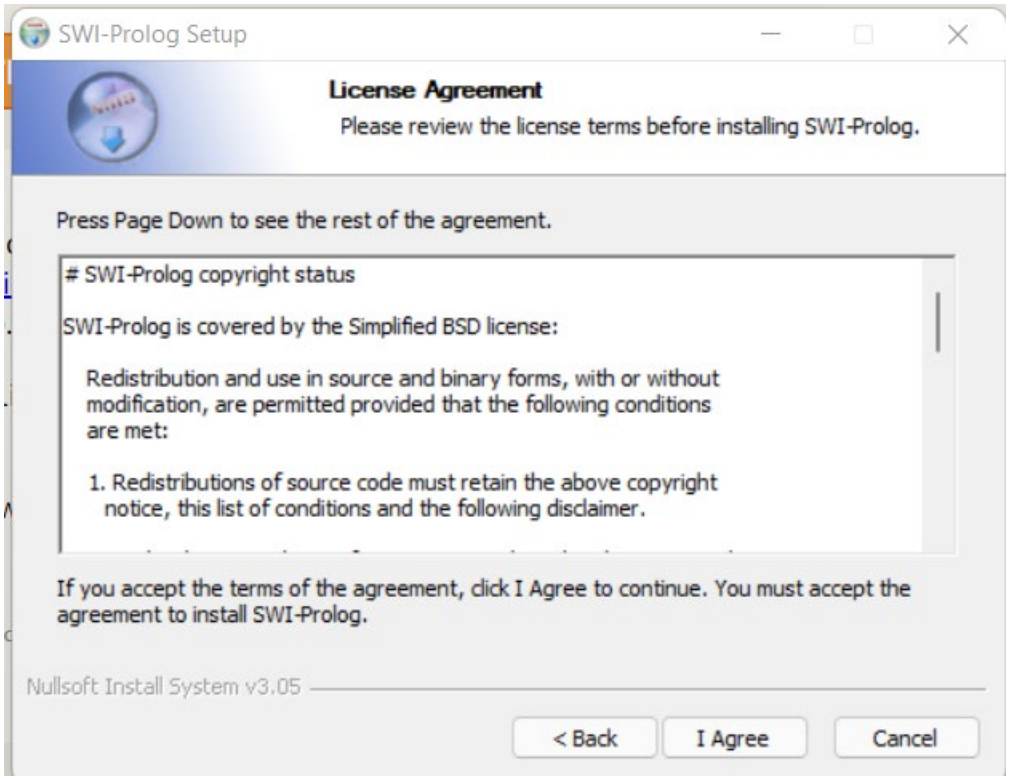


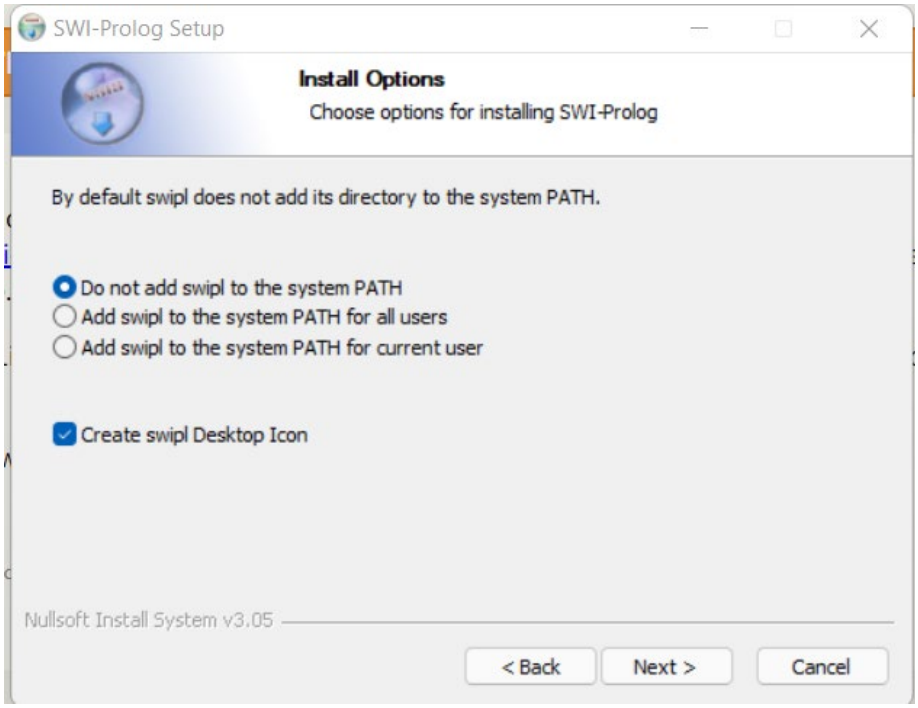
Now accept the agreement and install the exe file.

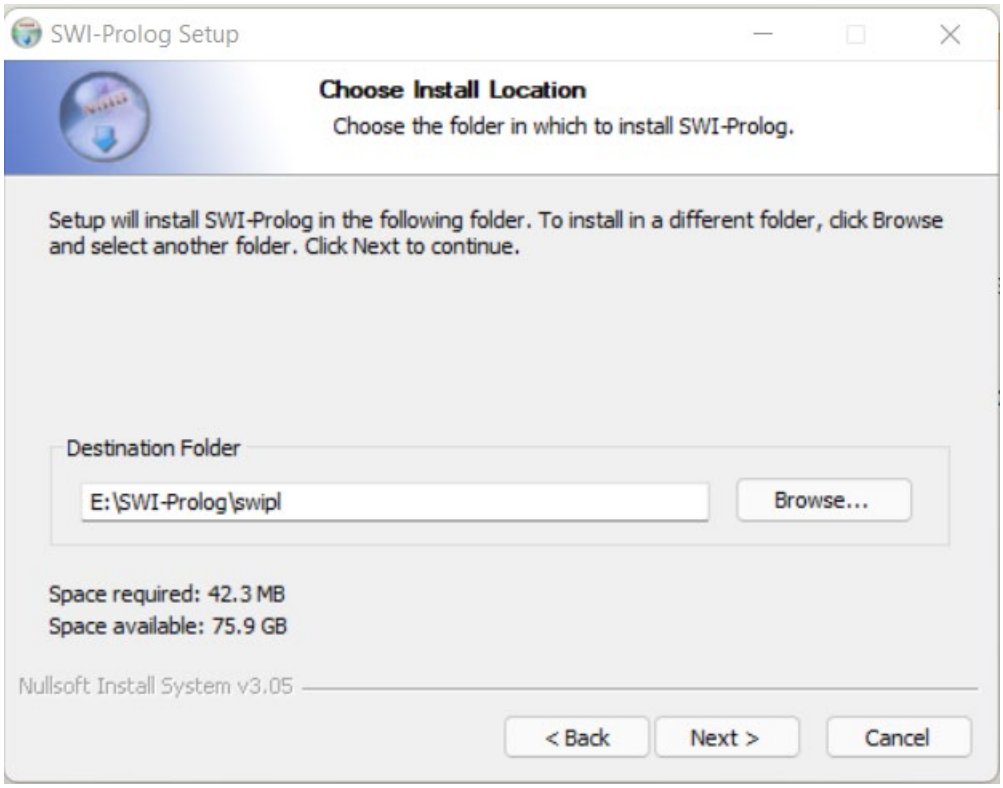


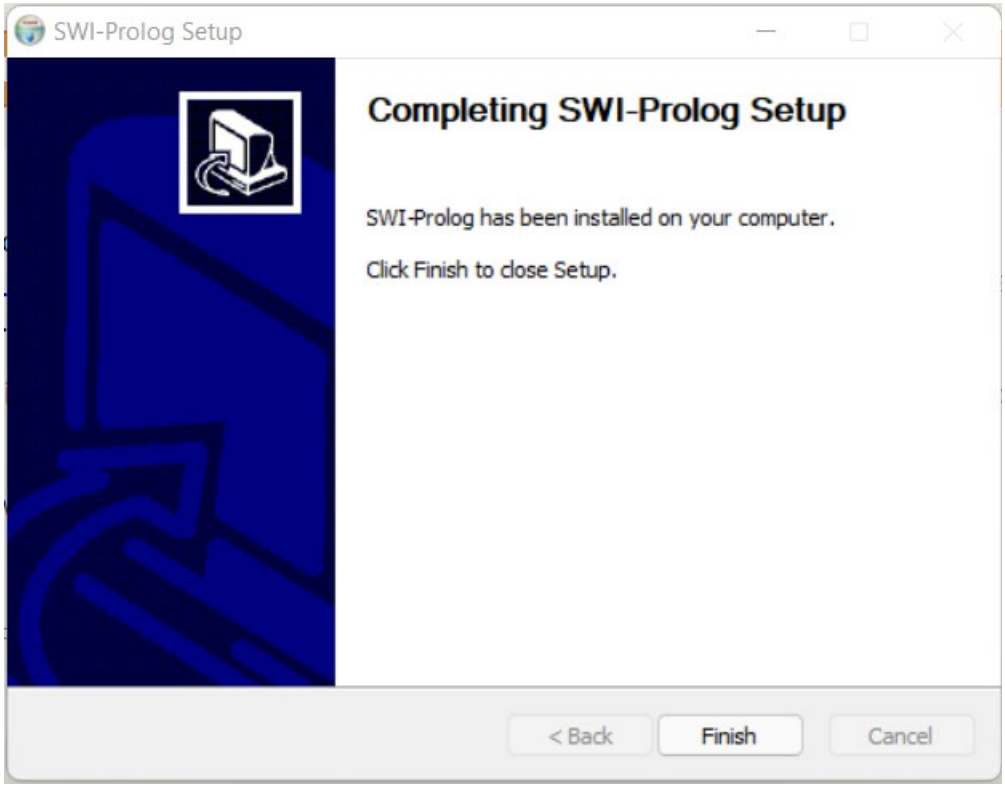
Once the exe file is downloaded install the setup required for Prolog.




SWI-Prolog Setup
License Agreement
Please review the license terms before installing SWI-Prolog.
Press Page Down to see the rest of the agreement.
SWI-Prolog copyright status
SWI-Prolog is covered by the Simplified BSD license:
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install SWI-Prolog.
Nullsoft Install System v3.05
< Back
I Agree
Cancel


SWI-Prolog Setup
Install Options
Choose options for installing SWI-Prolog
By default swipl does not add its directory to the system PATH.
Do not add swipl to the system PATH
Add swipl to the system PATH for all users
Add swipl to the system PATH for current user
Create swipl Desktop Icon
Nullsoft Install System v3.05
< Back
Next >
Cancel





CONCLUSION: Hence we have successfully installed SWI prolog.

Experiment No. 8

AIM: Write a program to implement Min Max Algorithm.

RESOURCES REQUIRED: SWI Prolog.

THEORY:

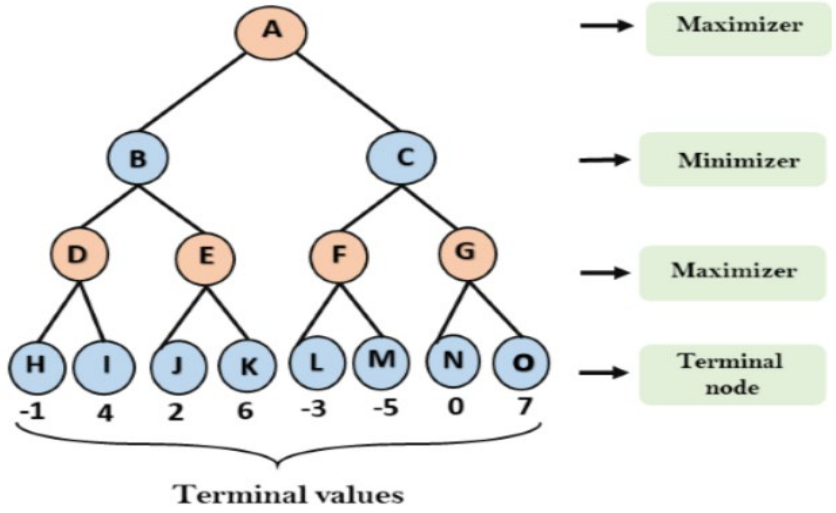
Min-max algorithm:

Min-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally. Min-Max algorithm uses recursion to search through the game-tree. Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state. In this algorithm two players play the game, one is called MAX and other is called MIN. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit. Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value. The minmax algorithm performs a depth-first search algorithm for the exploration of the complete game tree. The minmax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Working of Min-Max Algorithm:

In this example, there are two players one is called Maximizer and other is called Minimizer. Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score. This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes. At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step1:



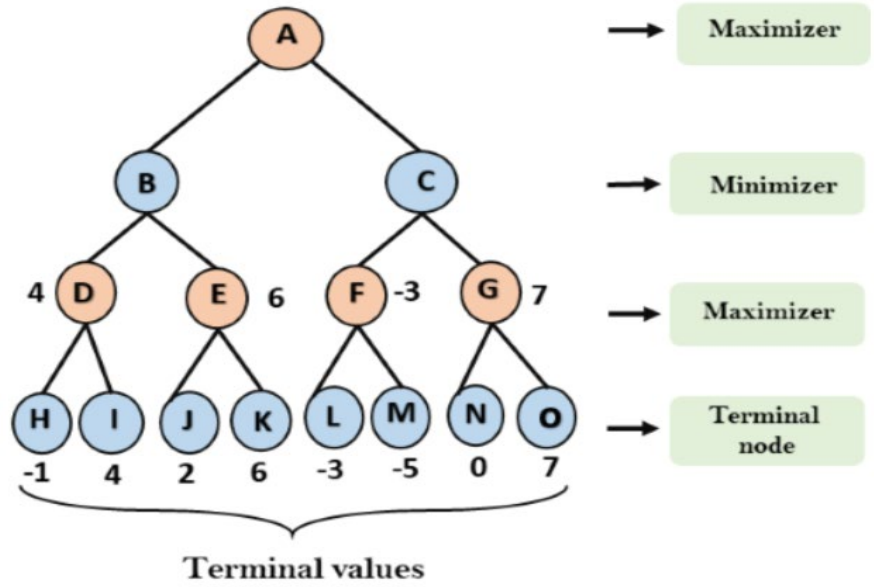
Step2:

For node D max(-1,- -∞) => max(-1,4)= 4

For Node E max(2, -∞) => max(2, 6)= 6

For Node F max(-3, -∞) => max(-3,-5) = -3

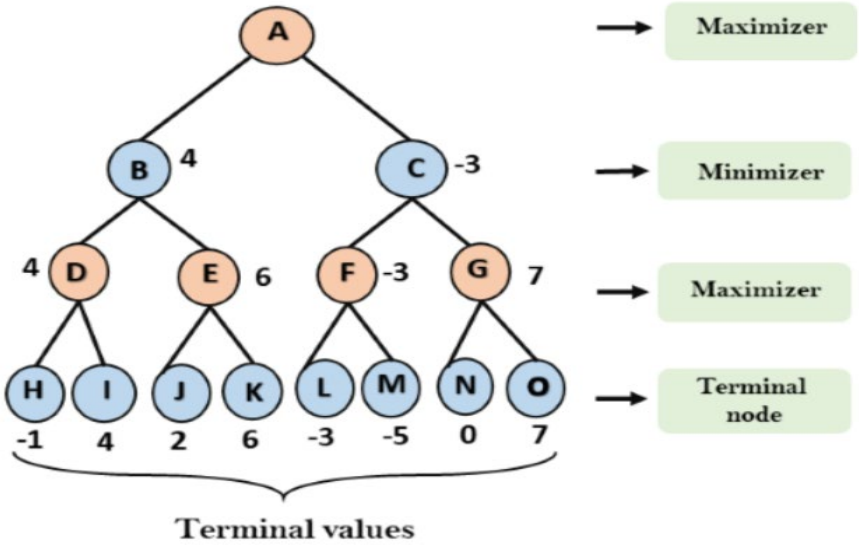
For node G max(0, -∞) = max(0, 7) = 7



Step3:

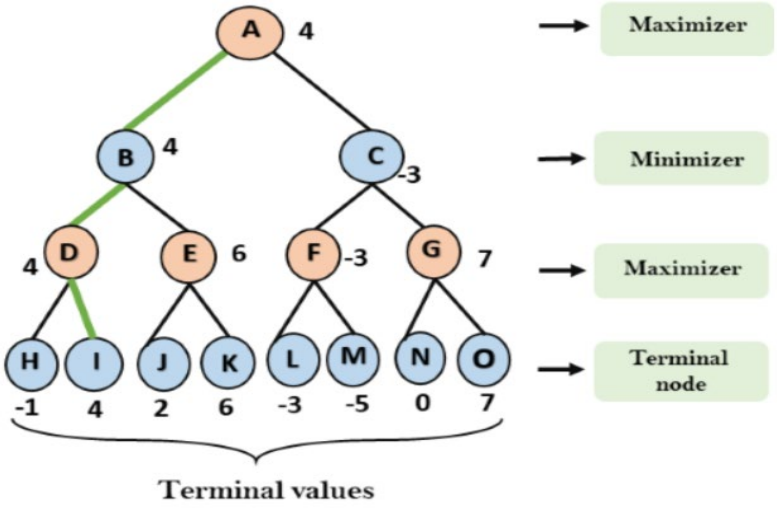
For node B= min(4,6) = 4

For node C= min (-3, 7) = -3



Step4:

For node A max(4, -3)= 4



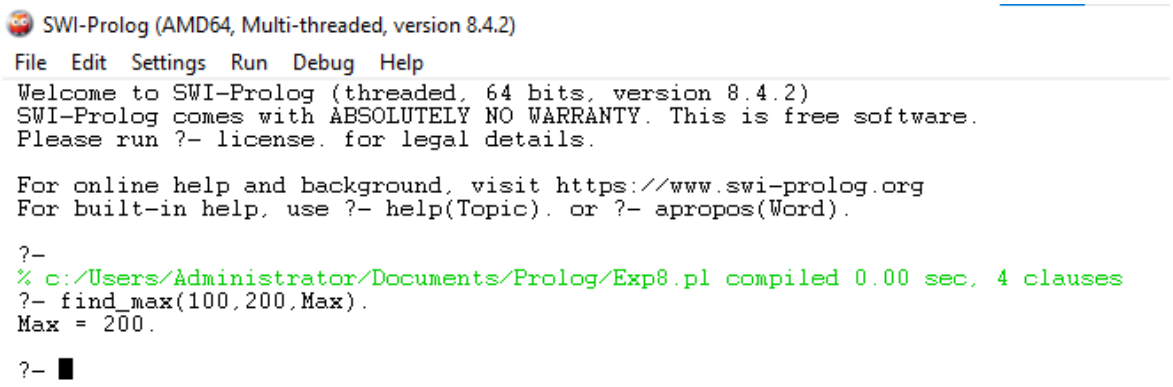
CODE:

```
find_max(X, Y, X) :- X >= Y, !.

find_max(X, Y, Y) :- X < Y.

find_min(X, Y, X) :- X =< Y, !.

find_min(X, Y, Y) :- X > Y.
```

Queries:

```
find_max(100,200,Max).
```

OUTPUT:



CONCLUSION: Hence we have successfully implemented the MinMax algorithm using SWI Prolog.

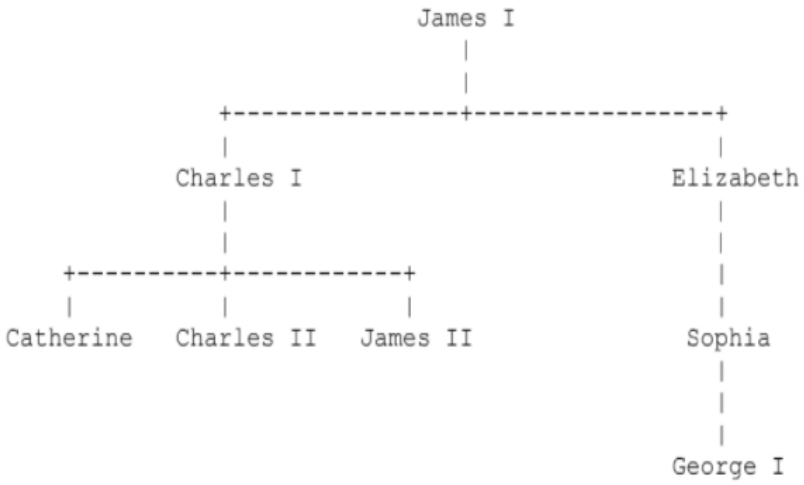
Experiment No. 9

AIM: To deduce information from family tree.

RESOURCES REQUIRED: SWI Prolog.

THEORY:

Family Tree:



CODE:

```
male(james1).
male(charles1).
male(charles2).
male(james2).
male(george1).
female(catherine).
female(elizabeth).
female(sophia).
parent(charles1, james1).
parent(elizabeth, james1).
parent(charles2, charles1).
parent(catherine, charles1).
```

parent(james2, charles1).

parent(sophia, elizabeth).

parent(george1, sophia).

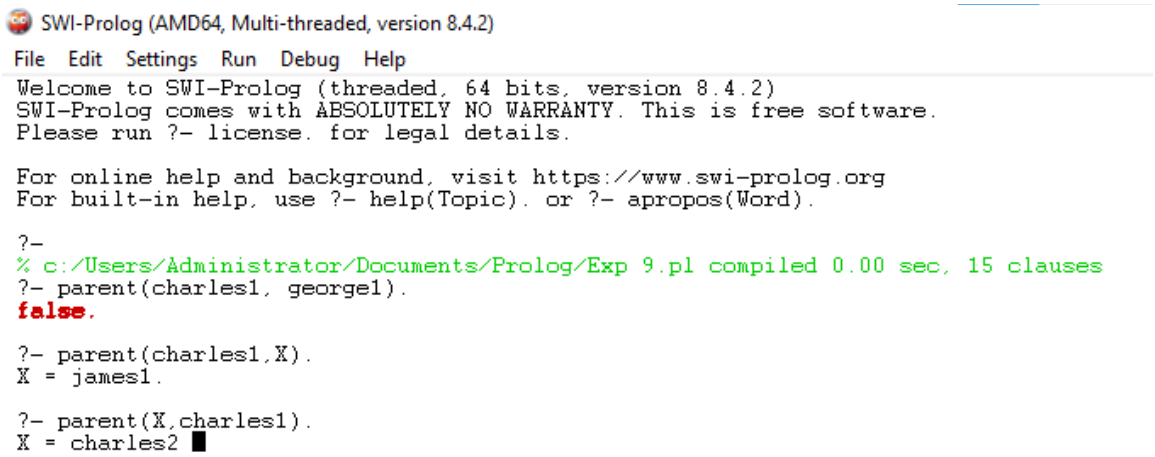
Queries:

parent(charles1, george1).

parent(charles1,X).

parent(X,charles1).

OUTPUT:



CONCLUSION: Hence we have successfully deduced the information from family tree using SWI Prolog.

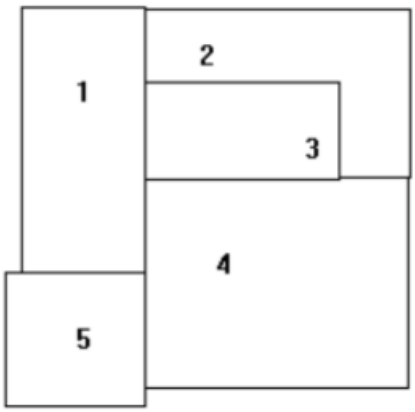
Experiment No. 10

AIM: Write a program to implement Graph Coloring Algorithm.

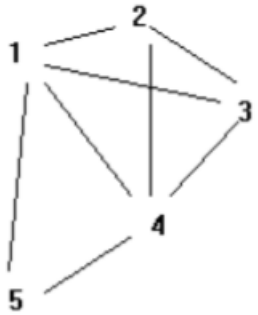
RESOURCES REQUIRED: SWI Prolog.

THEORY:

A famous problem in mathematics concerns coloring adjacent planar regions. Like cartographic maps, it is required that, whatever colors are actually used, no two adjacent regions may not have the same color. Two regions are considered adjacent provided they share some boundary line segment. Consider the following map.



We have given numerical names to the regions. To represent which regions are adjacent, consider also the following graph.



Here we have erased the original boundaries and have instead drawn an arc between the names of two regions, provided they were adjacent in the original drawing. In fact, the adjacency graph will convey all of the original adjacency information.

CODE:

A Prolog representation for the adjacency information could be represented by the following unit clauses, or facts.

adjacent(1,2). adjacent(2,1).

adjacent(1,3). adjacent(3,1).

adjacent(1,4). adjacent(4,1).

adjacent(1,5). adjacent(5,1).

adjacent(2,3). adjacent(3,2).

adjacent(2,4). adjacent(4,2).

adjacent(3,4). adjacent(4,3).

adjacent(4,5). adjacent(5,4).

Queries

adjacent(2,3).

adjacent(5,3).

adjacent(3,R).

Declare colorings for the regions in Prolog using unit clauses.

color(1,red,a). color(1,red,b).

color(2,blue,a). color(2,blue,b).

color(3,green,a). color(3,green,b).

color(4,yellow,a). color(4,blue,b).

color(5,blue,a). color(5,green,b).

To write a Prolog definition of a conflictive coloring, meaning that two adjacent regions have the same color. For example, here is a Prolog clause, or rule to that effect.

conflict(Coloring) :-

adjacent(X,Y),

color(X,Color,Coloring),

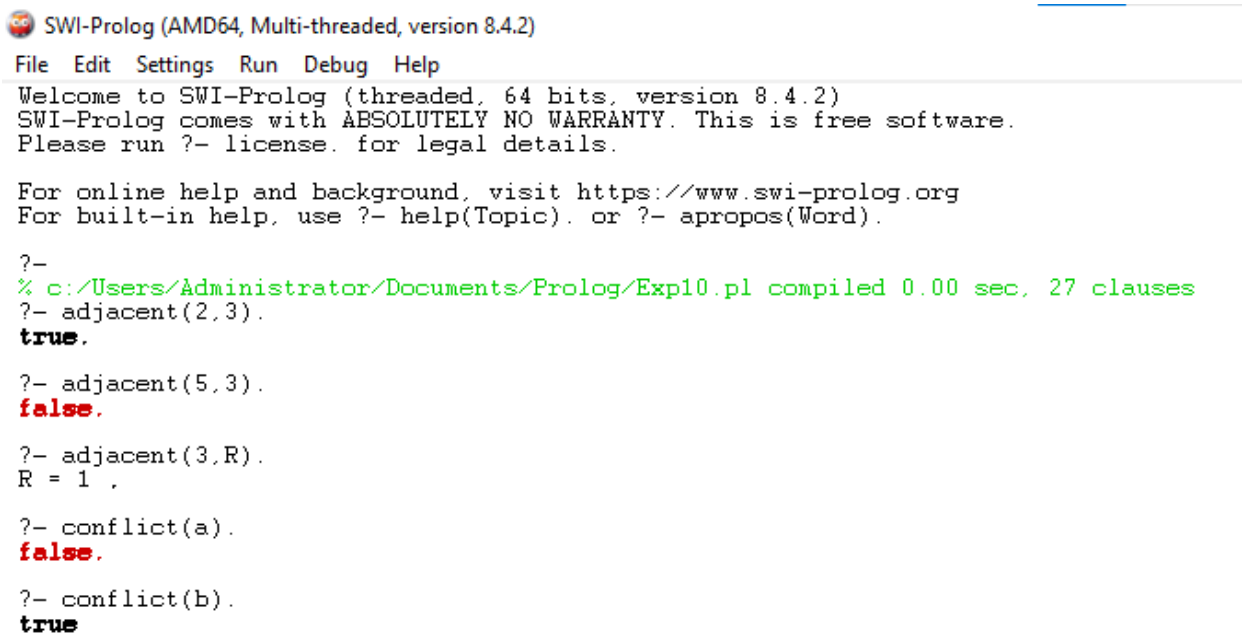
color(Y,Color,Coloring).

Queries:

conflict(a).

conflict(b).

OUTPUT:



CONCLUSION: Hence we have successfully implemented Graph Coloring Algorithm using SWI Prolog.

Experiment No. 11

AIM: Write a program to find factorial of a number using SWI Prolog.

RESOURCES REQUIRED: SWI Prolog.

THEORY:

Factorial:

The factorial (denoted or represented as n!) for a positive number or integer (which is denoted by n) is the product of all the positive numbers preceding or equivalent to n (the positive integer). The factorial function can be found in various areas of mathematics, including algebra, mathematical analysis, and combinatorics.

In Mathematics, factorial is an important function, which is used to find how many ways things can be arranged or the ordered set of numbers. The well known interpolating function of the factorial function was discovered by Daniel Bernoulli. The factorial concept is used in many mathematical concepts such as probability, permutations and combinations, sequences and series, etc. In short, a factorial is a function that multiplies a number by every number below it till 1. For example, the factorial of 3 represents the multiplication of numbers 3, 2, 1, i.e. 3! = 3 × 2 × 1 and is equal to 6. In this article, you will learn the mathematical definition of the factorial, its notation, formula, examples and so on in detail.

The function of a factorial is defined by the product of all the positive integers before and/or equal to *n,* that is:

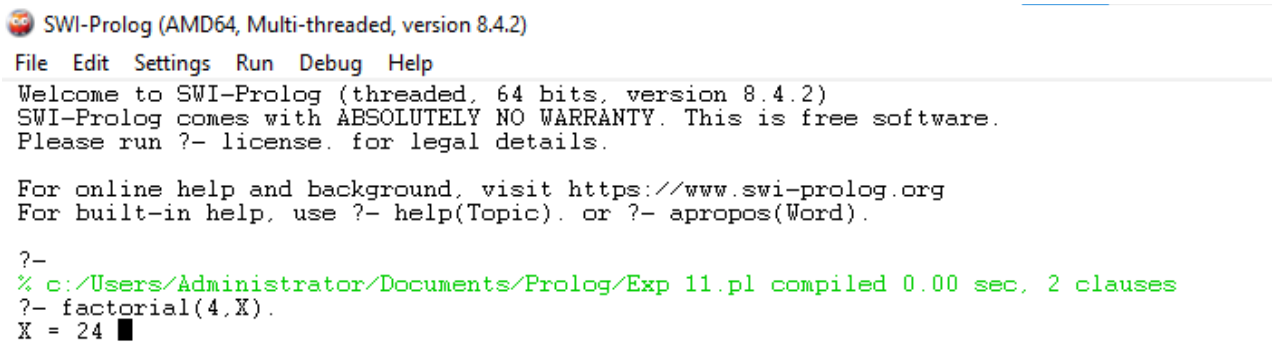
$$n! = 1 \cdot 2 \cdot 3 \cdots (n-2) \cdot (n-1) \cdot n,$$

CODE:

```
factorial(0,1).

factorial(N,M) :-

N>0,

N1 is N-1,

factorial(N1, M1),

M is N*M1.
```

Queries: factorial(4,X).

OUTPUT:



CONCLUSION: Hence we have successfully implemented the program to find a factorial of a number using SWI Prolog.

Assignment No. 1

Pg no. 1

34[illegible]_KARAN BUDDHIWANT

AI - Assignment 1

☆ AI Applications.

Q.1] NLP (Natural Language Processing)

Ans→ Natural Language Processing (NLP) is a subfield of linguistics, computer science and artificial intelligence concerned with the interactions between computers and human languages, in particular how to program computers to process and analyze large amounts of 'Natural Language' data.

✳ Common NLP Tasks :-

① Text and speech processing :-

→ Optical Characters Recognition :- Given an image representing printed texts, determine the corresponding texts.

→ Speech recognition :- Given a sound clip of a person or people speaking, determine the ~~textual~~ textual representation of the speech.

→ Speech ~~Segmentation~~ Segmentation :- Separate the words at the given sound clip of a person speaking.

→ Text - to Speech :- Given a text, transformation those unit are produced a spoken representation.

Pg no 2.

② Morphological analysis :-

→ Lemmatization :- The task of removing inflectional endings only and to return the base dictionary form of a word which is also known as a lemma. Lemmatization is another technique for reducing words to their normalized form.

→ Morphological segmentation :- Separate words into individual morphemes and classify the class of the morphemes. The complexity of the process depends upon the morphology (the structure of the word).

→ Part of speech tagging :- Given a sentence, determine the part of speech for each word. Many words are those which can act as different parts of speech according to the sentence.

→ Stemming :- The process of reducing inflected words to a base form. Stemming yields similar results as lemmatization but does so on grounds of rules, not a dictionary.

③ Syntactic analysis :-

→ Grammar induction :- Generate a formal grammar that describes a language's syntax.

→ Sentence breaking :- Given a chunk of text, find the sentence boundaries.

Pg no. 3

→ Passing :- Determine the passe tree of the given sentence. The grammar for natural language is ambiguous and typical sentences have multiple possible analyses.

Q.2] Robotics : Application of AI.

Ans→ Robotics is a domain in Astificial Intelligence that deals with the study of creating intelligence and efficient robots.

→ Robots are astificial agents acting in real life envisonment.

→ Robotics is a branch of AI, which is composed of Electrical Engineering, Mechanical Engineering, and Computer Engineering for designing construction and applications of robots.

→ Aspects of Robotics :-

① The robots have mechanical construction, form or shape designed to accomplish a particular task.

② They have electric components which power and control the machinery.

③ The contain some level of computer program that determines what, when and how a robot will perform actions.

Pg no 4

→ Robot locomotion :-

Locomotion is the mechanism that makes a robot capable of moving in its environment. There are various types of locomotions.

① Legged Locomotion:- This type of locomotion consumes more power while demonstrating walk, jump, trot, hop, climb up or down, crouch, etc.

② Wheeled locomotion :- It requires fewer no. of motors to accomplish a movement. It requires is little ease to implements as there are less stability issues in case of more no. of wheels. There are following types
→ Standard wheel
→ Castes wheel
→ Swedish 45° & swedish 90° wheel.
→ Ball or spherical wheel.

③ Slip / Slide locomotion :- In this type, the vehicles use tracks as in a tank. The robot is steered by moving the tracks with diffent speed in the same or opposite direction. It offers stability because of large contact area of track and ground.

Pg No. 5.

→ Components of a robot :-

① Power supply :- The robots are powered by batteries, solar power, hydraulic, or pneumatic power sources.

② Actuators :- They convert energy into movement.

③ Electric motors :- They are required for rotational movement.

④ Pneumatic Air Muscles :- They contract themselves when air is sucked in them.

⑤ Muscle wires :- They contract about 5% when electric current is passed through them.

⑥ Piezo Motors and Ultrasonic Motors :- Mainly used and best suitable for industrial robots.

⑦ Sensors :- They provide knowledge at real time information on the task environment. Robots are equipped with vision sensors to be to compute the depth in the environment. A tactile sensor imitates the mechanical properties of touch receptors of human fingertips.

Assignment No. 2

Pg. no. 1.

34_ KARAN BUDDHIWANT

AI - Assignment - 2

Q.] AI Appications in ~~health~~ healthcase, retail & banking.

Ans:→ a] AI applications in healthcase :-

① Managing medical records and other data :- Since the first step in health ~~care~~ case is compiling and analysing information, data management is the most widely used application of artificial intelligence and digital automation. Robots ~~called~~ collect, store reformat, and trace data to provide faster, more consistent access.

② Doing Repeatative Jobs :- Analysis test, X-rays, CT Scans, data entry, and other mundane tasks can all be done faster and more accurately by robots. Cardiology and radiology are ~~too~~ two disciplines where the amount of data to analyse can be huge and time consuming. Cardiologists and radiologists in future should only take most complicated cases where human supervision is needed.

③ Treatment Design :- Artificial Intelligence systems have been created to analysis data - notes and reports from a patients file, external research, and clinical expertise - to help select the correct, individually customized treatment plan.

④ Vistual Nusses :- The stastup sense has developed 'Molly' a digital nusse to help people monitor patients conditions and follow up with small treatments, between doctors visits. The program uses machine learning to support patients, specializing in chsonic illness.

⑤ Health monitoring :- Wearable health trackers like of from smartwatches, monitor heart rate, and activities like sports, sleep, etc. They can send alearts to the uses to get more exesize and can shase this information to doctors for additional examination on patients on their health track in daily routine.

b) AI Applications in Retail :-

① Caries Free Stores :- By cutting down employee compensation and inesreasing efficiency and seducing wait time, automation in billing process, stores had significant savings on opesational expenses. Amazon Stores in U.S and Canada ase one of those example whese these's no cashout system, go to the stose, take us needed ~~or~~ items and leave all the products taken will be automatically listed ~~for~~ for billing on the amazon app.

Pg:no 3.

② Chatbot - Based Solutions :- With the introduction of AI chatbots, its much more convinent for businesses to target customers needs and solve their problems in an efficient and a transpesant manner. AI chatbots provide a higher level of customers service, send notifications about new collections etc.

③ Price Regulation strategy :- AI technology for retail stores has helped business set ~~prices~~ prices for their products more competitively to thrive in the market. It visualizes the likely outcomes of multiple pricing ~~str~~ strategies. It reads data within seconds from business's earlier transaction and from the market for the products actual cost, ~~past~~ promotional activities, and sales figures in market and keeps its past or generating profiles within the same threshhold with higher efficiency.

④ Virtual Trial Rooms :- A new trend and a new way of shopping especially at a time of covid which gave this technique a big boost. These virtual trial rooms help customers save time, save money from travel expenses, and from the convinience at your home, the customers find pefect outfit with all elements pefectly matched.

⑤ Feedback and Predictions :- The electronic devices which have been installed are likely to ask about the customers experience after they are done with sales and ~~page~~ payments.

Pg: no. 4

C] AI Applications in Banking

① Cybersecurity and Fraud ~~detection~~ detection :- Due to inceasing online transaction, bill payements, cash withdrawal these is increase in need for banking sector to ramp up its cyber-security and fraud detection efforts. This when AI is used to help bonues to improve the security of ~~online~~ online finance, track loopholes in their systems, and minimize risks. AI along with ML can easily identity fraudulent activities.

② ~~Loan~~ Loan and credit decisions :- Banks have started incorporating AI-based systems to make more profitable, informed and safer loan and credit decisions. AI based systems will look into the behaviours and patterns of customers will limited credit history to determine their credit worthiness.

③ Tracking Market Trends :- AI in financial servies helps banks to ~~pro~~ process large volumes of data and predict the latest market trends, cussencies ~~charge~~ charges, and stocks. Advance machine learning techniques help evaluate market sentiments and suggest investment options.

④ Risks Management :- External global factors such as cussency fluctuations, natural disasters, or political assets have serious impact on banking and financial ~~indus~~ industries. AI also helps find risky applications by evaluating the probablity of a client failing to pay back a loan. Its predicts this future behaviour by analyzing past behavioural patterns and smartphone data.