Smt. Indira Gandhi College of Engineering
Ghansoli – Navi Mumbai
Computer Engineering Department
Academic Year 2022-23 (Even Sem)

Student Name: Shivam Narkhede **Roll No:** 29 **Class:** TE **Sem:** VI

Course Name: ~~CSL604~~ AI

Course Code: CSL604,

Experiment No. 2

Experiment Title: Select a problem statement in AI & write PEAS description and properties of Task environment & Problem formulation. for Mars Rover

Date of Performance	Date of Submission	Marks (10)					Sign / Remark
		A	B	C	D	E	
		2	3	2	2	1	
		Total Marks					

A: On Time Submission **B:** Understanding **C:** Analytical Skill

D: Critical Thinking **E:** Presentation

Date:_____

* Experiment No -2 *

Aim → Select a problem statement relevant to A.I

- Identify the problem
- PEAS description and properties of Task Environment.
- Problem formulation

Theory →

Identify the Problem → The A.I powered rover that can efficiently navigate the Martian Terrian, collect and analyze data, make decisions Based on that data and communicate Effectively with earth. The goal is to enable the rover to perform the complex task autonomously reducing the need for Human intervention and making it easier to explore the Planet's surface.

* PEAS Description →

i) Performance Measure →

- Data carried by Mars Rover
- Efficiency of data Transfer.
- Collect quality and quantity of data over an extended period of time
- Ensure safety of operation

Date:_____

ii) Environment →
- Rugged and harshed Terrain
- Extreme Temprecturs, ranging from -195 to 20 °C
- Rocky and sandy Terrain
- Dust storm.

iii) Actuators →
- Six wheels with individual steering motors
- Robotic arms. with various tools.
- Remote sensing instruments ex→ camera.
- Solar panels or Nuclear power soure

iv) sensors →
- Multiple camera for image and video Recording
- Spectrometer for analysing rocks
- Laser induced Breakdown spectroscopy
- Instrument for Magnetometer

* Problem formulation →

i) Define the goals and objective of the mission, including scientific Research obj. and technolyical demo. obj.
2) Identify the constraints & limitiations of martian env. such as extreme temp. and Radiation levels.
3) Delermine the capabilities and specification of the Rover Hardware / software sys.

Date:_____

4) Establish Risk Assesement and management strategies that metigate potential Hazards.

* Actions for Mars Rover →

1) Drive
2) Use Robotic Arm
3) Deploy Drill
4) Take images or videos.
5) Analyse composition.
6) Adjust position
7) Perform Hazard Avoidance

* Successor function for Mars Rover →

1) If Rover drives forward, it's position and orientation will change.
2) If Robotic Arm is used to Retrieve Rock sample or use an instrument the Rovers onboard payload will change
3) If drill is deployed to obtain Sub surface sample, the amount and type of soil will change
4) If solar panels are adjusted the Rovers power supply will change

Date:_____

* Goal state → The goal state of mars Rover is to succesfully complete its mission objective, which depends on specific goal of each mars mission, and can vary from one mission to another.

* Path cost → The path cost for the mars Rover prefers to the cost or effort Required for the Rover to Traverse a specific path or sequence of actions. The path cost would be calculated Based on various factors such as

- Energy consumption
- Time Taken
- Risk encountered
- scientific value gained.

Conclusion → With the help of PEAS description, we can explain the working of mars Rover with perfection.

Name → Shivam Markhede Roll → 29

Batch → IInd Sub → A.I Date: Markhede.

* Experiment NO - 3 *

Aim → Implementing Uninformed search Algorithm (Depth Limited search) for Problem ~~search~~ Solving in AI.

Theory → Depth limited search algorithm finds out the Best depth limit and does it by gradually increasing the limit until a goal is found.

It terminates the following in two cases.

1) When the Goal Node is found.
2) The Goal /Node does not exist in the Graph /Tree.

The Depth limited search algorithm is useful uninformed search when search space is large and depth of goal Node is unknown.

Algorithm →

Input : START and GOAL State

Output : LOCAL VARIABLE : found.

Initialize d = 0 and found D = false

while (found = false and d <= max depth) d.

Perform Dfs from start to depth D.

if goal state is obtained

found = true

Else

Roll → 29

Date:

(

the search of depth d

d = d + 1

}

End While

If FOUND = true

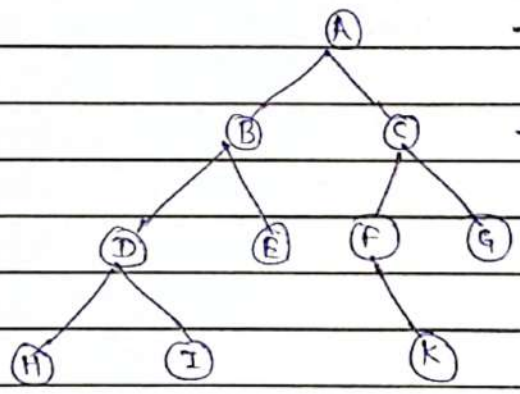
return the Depth

Else

Return the element is not present

STOP

Ex →



1st iteration → A

2nd iteration → A, B, C

3rd iteration → A, B, C, D, E, f, G, *

4th iteration → A, B, D, H, I, E, C, F, K, G.

In the fourth iteration, the algorithm will go for the final Node.

^ Goal.

Roll → 29

Date:

Conclusion →

The properties of Depth limited search algorithm are

* Time Complexity - $O(b^d)$
* Space Complexity - $O(b^d)$
* Completeness - The Algorithm is complete if the Branching factor is finite.
* Optimal → This Algorithm is optimal if path cost is non-decreasing function of the depth of the Node.

Name → Shivam Narkhede
Roll → 29 Sub → A.I
Batch → III

Date:

* Experiment No - 4 *

Aim → Implementing A* search - Informed Search Algorithm for Problem Solving in A.I..

Theory →

* A* Algorithm is one of the best and popular Techniques used for path finding and graph traversals.
* A lot of games and Web-Based Maps used this Algorithm for finding the shortest path efficiently.
* It is essentially a Best first search Algorithm.
  - A* Select the path that maximizes
  - $f(n) = g(n) + h(n)$
    where n is the next node on path
  - $g(n)$ is the cost of path from start node to n.,
  - $h(n)$ is the heuristic function that estimates the cost of the cheapest gpath from n to the goal.

* Algorithm →

- OPEN consists on Nodes that have been visited but not expanded. This is the list of pending Task.
- CLOSE Consists on the Nodes that have Been visited or expanded.

The Algorithm is as follows →

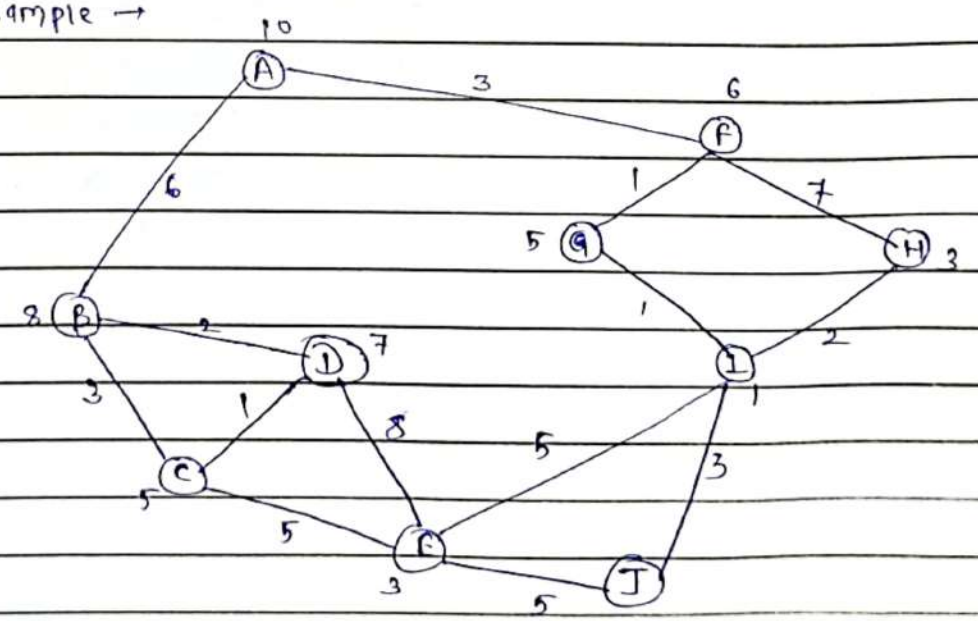
i) Initially, OPEN consists solely in single Node, the start Node S.

Roll → 29

Date:

2) If the list is empty, return failure and exit.
3) Remove Node n with the smallest value of f(n) from OPEN and move it to list CLOSED If node n is a goal state ; return success and exit.
4) Expand Node n.
5) If any successor to the n is the Goal Node, return success and the solution by Tracing the path from Goal Node to S.
6) for Each successor Node ,
   - Appy the evaluation function f to the Node
   - If the Node has not been in either list, add it to OPEN.
7) GO back to step (2).

Example →



Roll → 29

Date: Tuesday

⇒ Solution →

Step ① →

- Start with Node A
- Node B and Node f can be reached from Node A

A* Algorithm calculates f(B) & f(f)

- f(B) = 6+8 = 14
- f(f) = 3+6 = 9

Since f(f) < f(B), so it decides to go to Node f

Path → A → f

Step ② →

Node G & H can be reached from Node f.

∴ f(G) & f(H)

- f(G) = (3+1) + 5 = 9
- f(H) = (3+7) + 3 = 13

f(G) < f(H), so it will go to Node G.

Path → A → f → G

Step ③ →

Node I can be reached from G.

∴ f(I) = (3+1+3) + 1 = 8

So it will go to Node I

Path → A → f → G → I

Roll → 29

Date: Starknode

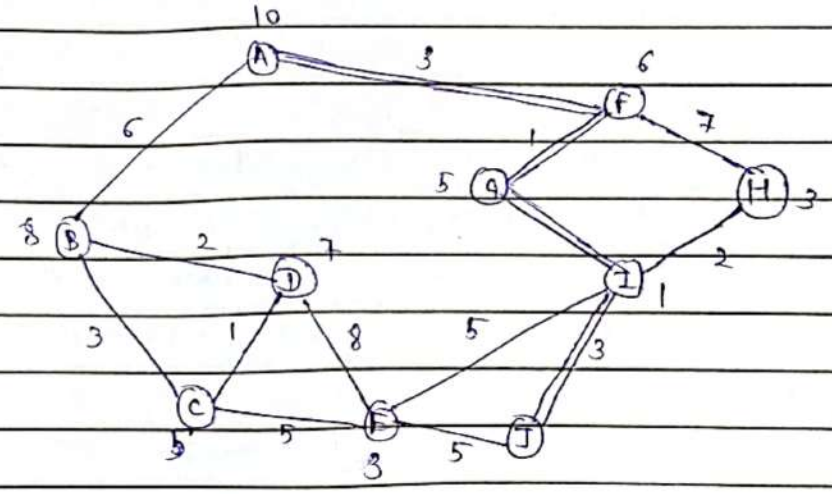
Step ④ → Node E, H & J can be reached from Node I

∴ $f(E) = (3+1+3+5) + 3 = 15$

∴ $f(H) = (3+1+3+2) + 3 = 12$

• $f(J) = (3+1+3+3) + 0 = 10$.

Path → A → f → G → I → J



Conclusion → A* Algorithm has following properties

- Completeness → Guranteed even on infinite Graphs.
- Admissibility → Guronteed if given an admissible heurestics (h(n))
- Time complexity → Depends on the Accuracy of Heurestic function
- space Complexity → O (B to the power of d)

Roll → 29

Name → Shivam Narkhede

Batch → IInd Sub → A.I. Date: Narkhede

* Experiment No - 5 *

Aim → To implement Hill Climbing Algorithm Simple and steepest variation for 8 Problem Solving in AI

Theory → Hill Climb Algorithm is a local Search Algorithm where the path to the solution is irrelevant, It considers only the final configuration.

The Hill Climb search Algorithm is simply a LOOP that continuously moves in a direction of increasing value. This is Uphill. It terminates when it reaches the "peak" where no neighbour has higher value.

- Simple Hill Climb Algorithm
- Steepest Hill Climb Algorithm.

Disadvantages →

- Local Maxima → A Local maxima is a peak that is higher than each of its neighbouring states but lower than global maximum.

- Ridges → Result in sequence of Local Maxima, that is very difficult for greedy Algorithms to Navigate

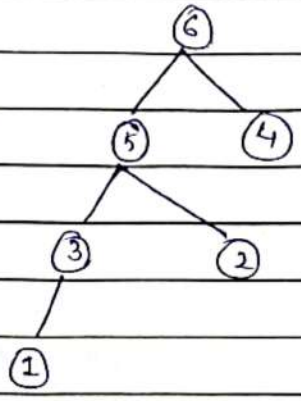
- Plateu → It is a flat area of the state scape landscape it can be flat local maximum from which not uphill exists.

Roll → 129

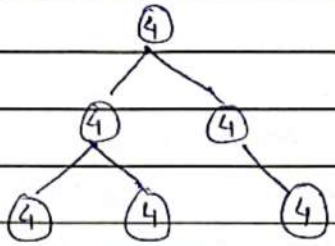
Harshada.

Date:

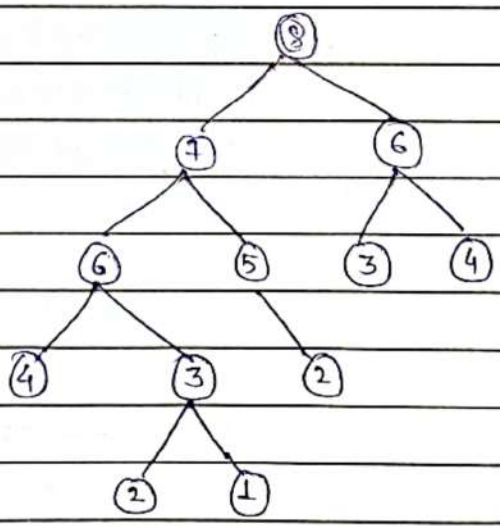
* Local Maxima →



* Plateau →



Ridge →



Roll → 29

Date: Starkhole

* Algorithm →

A Simple Hill Climbing Algorithm

1) Evaluate the initial state. If it is a Goal state then return and quit., otherwise make it a current state and go to step ②.

2) Loop until a solution is found or there are New operators left to be applied, i.e no new children left to be explored a Select and Apply a new operator.

3) a) If it is a goal then return and quit.
b) If it is better than current state then make it current new state.
c) If it is not better than current state then loop, go back to step ②.

* Conclusion → It is an iterative algorithm that starts with an Arbitrary solution to make a problem, then attempts to find a better ~~problem~~ solution by making an incremental change to the solution.

Name → Shivam Markhede
Batch → IInd Roll → 29
Subject → A.I

Markhede

Date:_____

* Experiment NO - 6 *

Aim → To implement Basic Programs in prolog

Theory → Prolog is the short form of logical programming. It is a logical and declarative programming language A stack scheduling policy is adopted. It maintains the Resolvent as a stack. It pops the top subgoal for reduction and pushes the derived goal on the resolvent stack.

Components of Prolog →

i) Constants → No. & symbols
ii) String → String of characters
iii) function & Predicate Name → Start with Alphabet and formed by using lower case letters neumericals, and underscores.
iv) Variables → Similar to functions but it must start with Upper case letters.
v) Clause → Terminated by full stop (.)
vi) Goal to prolog problem → given after Symbols ?- & terminated by Full Stop. (.)

Date:

$A \Rightarrow B_1, B_2, \ldots B_n$

Here A is consequent and $B_1, B_2, \ldots B_n$ are antecedants. A is true if $B_1, B_2, \ldots B_n$ are true simultaneously. Clause with No antecedants is a fact which is always true

Prolog control strategy →

* Prolog contains 3 Basic strategies.

i) Forward Moment →

Eg → $G_1 \Rightarrow B_{11}, B_{12}$

$G_2 \Rightarrow B_{21}, B_{22}, B_{23}, B_{24}$

$B_{11} \Rightarrow B_{111}, B_{112}$

To solve query ? → G_1, G_2

? − B_{11}, B_{12}, G_2

? − $B_{111}, B_{112}, B_{12}, G_2$.

ii) unification → It is a process of matching or finding the most general unifier. Constant matches with same constant. Variable matches with any constant and becomes instantiated. A variable matches with another variable.

Turbo

Date:_____

3) Backtracking → It takes place in 2 situations

a) When subgoal fails.

b) When last subgoal succeed and find alternative solution.

Ground query → Computation of ground query G(with no variable) generate a Proof Tree. If a query succeed then answer is Yes. Otherwise it is No.

Computation of a Non Ground query G, generates search tree and all finds all possible solutions of G, with Respect to a Program P.

Conclusion → Therefore, Basic ~~Prolog~~ programs in Prolog are implemented.

1. Basic

Equals.

```
equal(X,Y):-
   X=Y.
```

```
| ?- [prog1].
compiling C:/Users/aryal/Desktop/New folder/prog1.pl for byte code...
C:/Users/aryal/Desktop/New folder/prog1.pl compiled, 2 lines read - 271 bytes written, 7 ms

yes
| ?- equal(1,1)
.

yes
| ?- equal(1,11)
.

no
| ?- equal(1,Y).

Y = 1

yes
| ?- equal(X,99).

X = 99

yes
| ?-
```

Square.

```
square(X,Y):-
   Y is X*X.
```

```
| ?- [prog1].
compiling C:/Users/aryal/Desktop/New folder/prog1.pl for byte code...
C:/Users/aryal/Desktop/New folder/prog1.pl compiled, 2 lines read - 399 bytes written, 7 ms

yes
| ?- square(2,4).

yes
| ?- square(2,Y).

Y = 4

yes
| ?- square(3,Y).

Y = 9

yes
| ?- square(X,4).
```

2. List

Length

```
list_length([],0).
list_length([_|TAIL],N) :-
    list_length(TAIL,N1),
    N = N1 + 1.
```

```
| ?- [len].
compiling C:/Users/aryal/Desktop/New folder/len.pl for byte code...
C:/Users/aryal/Desktop/New folder/len.pl compiled, 2 lines read - 657 bytes written, 6 ms

yes
| ?- list_length([a,b,c,d],Len).

Len = 4

yes
| ?- list_length([[a,b],[c,d]],Len).

Len = 2

yes
| ?- list_length([],Len).

Len = 0

yes
| ?- |
```

Concatenation

```
list_concat([],L,L).
list_concat([X1|L1],L2,[X1|L3]) :- list_concat(L1,L2,L3).
```

```
| ?- [concat].
compiling C:/Users/aryal/Desktop/New folder/concat.pl for byte code...
C:/Users/aryal/Desktop/New folder/concat.pl compiled, 2 lines read - 527 bytes written, 13 ms

(15 ms) yes
| ?- list_concat([1,2],[a,b],NewList).

NewList = [1,2,a,b]

yes
| ?- list_concat([x,y],[a,b],NewList).

NewList = [x,y,a,b]

yes
```

3. Recursion

Factorials.

```
factorial(0,1).
factorial(N,M) :-
```

```
    N>0,
    N1 is N-1,
    factorial(N1, M1),
    M is N*M1.
```

```
| ?- [prog3].
compiling C:/Users/aryal/Desktop/New folder/prog3.pl for byte code...
C:/Users/aryal/Desktop/New folder/prog3.pl compiled, 6 lines read - 858 bytes written, 10 ms

(16 ms) yes
| ?- factorial(3,M).

M = 6 ?

yes
| ?- factorial(9,M).

M = 362880 ?

yes
| ?- factorial(5,M).

M = 120 ?

yes
| ?-
```

Name → Shivam Narlawde

Roll → 29 Batch → IV^th

Subject → A.I

Date:

Narlawde.

* Experiment No-8 *

Aim → Implement Bayesian Belief Network – Borglary Alarm problem.

Theory → Bayesian Belief Network is a graphical Representation of diffrent probablistic relationship among random variables in a particular set. It is a classifier with no Dependencies on atmbutes i.e it is condition independent. Due to its feature of Joint probability, the Probability of in Bayesian Belief Network is derived, Based on a condition – P (atmbute/parent) i.e probabilty of an atmbute, true over parent atmbute. Real word applications are Probabilistic in nature, and to represent athe relatatimship between multiple events., we need a Byesian Network. It can be also used in variovs tasks including prediction anomaly detection, diagnostic, automated insight, reasoning, time series Prediction & decision making uncertanity.

Bayesian Network can be used for Building models from data expert opinions and it consists of two parts.

- Directed Acyclic Graph
- Table of Conditional Probabilities.

Date: 17/11/2023

Algorithm → 1) Import all required libraries
2) Create a Baysian Network
3) Define Conditional Probability distribution (CPP)
4) Add CPPs to the Network structure
5) Check if the Model is valid, throw an exception otherwise
6) Print the probability distribution & plot the model
7) Perform variable Animation for inference
8) Calculate the Probability of Burglary if there is a Burglary and a earth-quake
9) Print the Probabilities.

Conclusion → Thereform Burglary Alarm Problem is implemented.

Program:

```
# Import libraries
import pgmpy.models
import pgmpy.inference
import networkx as nx
import pylab as plt
# Create a bayesian network
model = pgmpy.models.BayesianModel([('Burglary', 'Alarm'),
                                    ('Earthquake', 'Alarm'),
                                    ('Alarm', 'JohnCalls'),
                                    ('Alarm', 'MaryCalls')])
# Define conditional probability distributions (CPD)
# Probability of burglary (True, False)
cpd_burglary = pgmpy.factors.discrete.TabularCPD('Burglary', 2, [[0.001], [0.999]])
# Probability of earthquake (True, False)
cpd_earthquake = pgmpy.factors.discrete.TabularCPD('Earthquake', 2, [[0.002], 0.998])
# Probability of alarm going of (True, False) given a burglary and/or earthquake
cpd_alarm = pgmpy.factors.discrete.TabularCPD('Alarm', 2, [[0.95, 0.94, 0.29, 0.001],
                                                           [0.05, 0.06, 0.71, 0.999]],
                                              evidence=['Burglary', 'Earthquake'],
                                              evidence_card=[2, 2])
# Probability that John calls (True, False) given that the alarm has sounded
cpd_john = pgmpy.factors.discrete.TabularCPD('JohnCalls', 2, [[0.90, 0.05],
                                                              [0.10, 0.95]],
                                             evidence=['Alarm'],
                                             evidence_card=[2])
# Probability that Mary calls (True, False) given that the alarm has sounded
cpd_mary = pgmpy.factors.discrete.TabularCPD('MaryCalls', 2, [[0.70, 0.01],
                                                              [0.30, 0.99]],
                                             evidence=['Alarm'],
                                             evidence_card=[2])
# Add CPDs to the network structure
model.add_cpds(cpd_burglary, cpd_earthquake, cpd_alarm, cpd_john, cpd_mary)
# Check if the model is valid, throw an exception otherwise
model.check_model()
# Print probability distributions
print('Probability distribution, P(Burglary)')
print(cpd_burglary)
print()
print('Probability distribution, P(Earthquake)')
print(cpd_earthquake)
print()
print('Joint probability distribution, P(Alarm | Burglary, Earthquake)')
print(cpd_alarm)
print()
print('Joint probability distribution, P(JohnCalls | Alarm)')
print(cpd_john)
print()
```

```
print('Joint probability distribution, P(MaryCalls | Alarm)')
print(cpd_mary)
print()
# Plot the model
nx.draw(model, with_labels=True)
plt.savefig('C:\\DATA\\Python-data\\bayesian-networks\\alarm.png')
plt.close()
# Perform variable elimination for inference
# Variable elimination (VE) is a an exact inference algorithm in bayesian networks
infer = pgmpy.inference.VariableElimination(model)
# Calculate the probability of a burglary if John and Mary calls (0: True, 1: False)
posterior_probability = infer.query(['Burglary'], evidence={'JohnCalls': 0, 'MaryCalls': 0})
# Print posterior probability
print('Posterior probability of Burglary if JohnCalls(True) and MaryCalls(True)')
print(posterior_probability)
print()
# Calculate the probability of alarm starting if there is a burglary and an earthquake (0: True,
1: False)
posterior_probability = infer.query(['Alarm'], evidence={'Burglary': 0, 'Earthquake': 0})
# Print posterior probability
print('Posterior probability of Alarm sounding if Burglary(True) and Earthquake(True)')
print(posterior_probability)
print()
```

```
Requirement already satisfied: pgmpy in /usr/local/lib/python3.7/dist-packages (0.1.18)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.7/dist-packages (from pgmpy) (3.0.8)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.0.2)
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.10.0+cu111)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.1.0)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (from pgmpy) (0.10.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.3.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from pgmpy) (4.64.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.21.6)
Requirement already satisfied: networkx in /usr/local/lib/python3.7/dist-packages (from pgmpy) (2.6.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.4.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas->pgmpy) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas->pgmpy) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->pgmpy) (3.1.0)
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from statsmodels->pgmpy) (0.5.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch->pgmpy) (4.1.1)
/usr/local/lib/python3.7/dist-packages/pgmpy/models/BayesianModel.py:10: FutureWarning: BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.
  FutureWarning,
Probability distribution, P(Burglary)
+-------------+-------+
| Burglary(0) | 0.001 |
+-------------+-------+
| Burglary(1) | 0.999 |
+-------------+-------+

Probability distribution, P(Earthquake)
+---------------+-------+
| Earthquake(0) | 0.002 |
+---------------+-------+
| Earthquake(1) | 0.998 |
+---------------+-------+

Joint probability distribution, P(Alarm | Burglary, Earthquake)
+------------+---------------+---------------+---------------+---------------+
| Burglary   | Burglary(0)   | Burglary(0)   | Burglary(1)   | Burglary(1)   |
+------------+---------------+---------------+---------------+---------------+
| Earthquake | Earthquake(0) | Earthquake(1) | Earthquake(0) | Earthquake(1) |
+------------+---------------+---------------+---------------+---------------+
| Alarm(0)   | 0.95          | 0.94          | 0.29          | 0.001         |
+------------+---------------+---------------+---------------+---------------+
| Alarm(1)   | 0.05          | 0.06          | 0.71          | 0.999         |
+------------+---------------+---------------+---------------+---------------+
```

Joint probability distribution, P(JohnCalls | Alarm)

Alarm	Alarm(0)	Alarm(1)
JohnCalls(0)	0.9	0.05
JohnCalls(1)	0.1	0.95

Joint probability distribution, P(MaryCalls | Alarm)

Alarm	Alarm(0)	Alarm(1)
MaryCalls(0)	0.7	0.01
MaryCalls(1)	0.3	0.99

Finding Elimination Order: : 0% 0/2 [00:00<?, ?it/s]

Eliminating: Alarm: 100% 2/2 [00:00<00:00, 38.42it/s]

Posterior probability of Burglary if JohnCalls(True) and MaryCalls(True)

Burglary	phi(Burglary)
Burglary(0)	0.2842
Burglary(1)	0.7158

Finding Elimination Order: : 0/0 [00:00<?, ?it/s]

0/0 [00:00<?, ?it/s]

Posterior probability of Alarm sounding if Burglary(True) and Earthquake(True)

Alarm	phi(Alarm)
Alarm(0)	0.9500
Alarm(1)	0.0500

Date:_____

* Experiment NO - 7 *

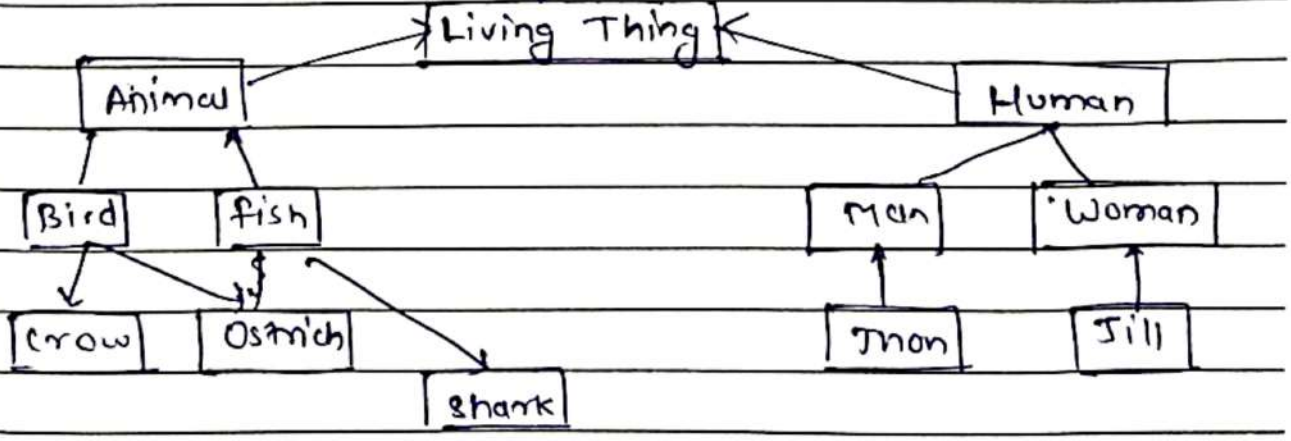
Aim → To implement first order Logic using PROLOG : Representation of knowledge in semantic Network and Inheritance

Theory →

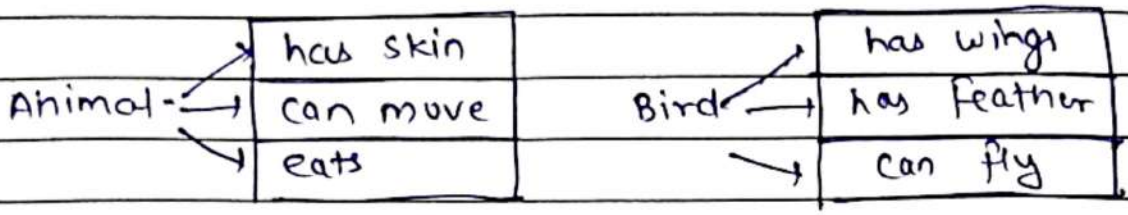
Problem Statement → Consider the following knowledge be Represented into corresponding Semantic Network Representation.

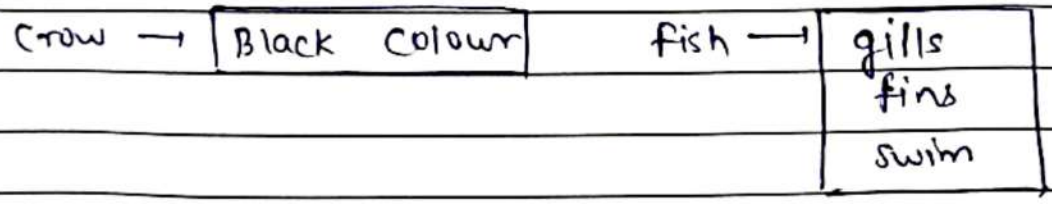
John is a man. Jill is a women. Every man is human. Every women is human. Every human is livingthing. All Birds and fishes are animals. Every living thing Breathes. All animals have skin, can move and eat. Ostrich has long leg and it is tall. All fishes have fins, gills & can swim and crow is Black.

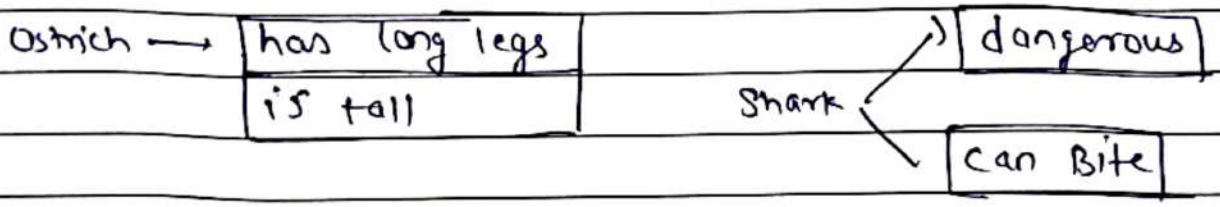
Shark is dangerous and can Bite.



Date: _____







facts →
1) (animal, Living Things)
2) inst (Ostrich, Bird)
3) prop (livingthing, Breathe)

Rules →

Sub class (x,y) → subc (x,y)
Sub class (x,y) → subc (x,y)

? — subclass (bird, animal)
True.

Date:_____

```
? — subclass ( fish , living Thing )
        True
? — Instance ( ostrich , Bird )
        True
? — instance ( ostrich , animal )
        True
? — property ( Black , crow )
        True
? — property ( fly , crow )
        True
? — property ( move , crow )
        True.
```

Conclusion — Hence, implementation of first order logic using PROLOG for Representation of knowledge in semantic Network.

Smt. Indira Gandhi College of Engineering
Ghansoli – Navi Mumbai

Computer Engineering Department
Academic Year 2022-23 (Even Sem)

Student Name: Sanskruti Palekar Roll No: 32 Class: TE Sem: VI

Course Name: Artificial Intelligence Lab

Course Code: CSL604

Experiment No. 03

Experiment Title: to Implement uninformed search algorithm DFS & IDLS for problem solving in AI

Date of Performance	Date of Submission	Marks (10)					Sign / Remark
		A	B	C	D	E	
		2	3	2	2	1	
		Total Marks					

A: On Time Submission B: Understanding C: Analytical Skill

D: Critical Thinking E: Presentation

Experiment No 3: To Implement Uninformed search algorithm DFS and IDLS for problem solving in AI.

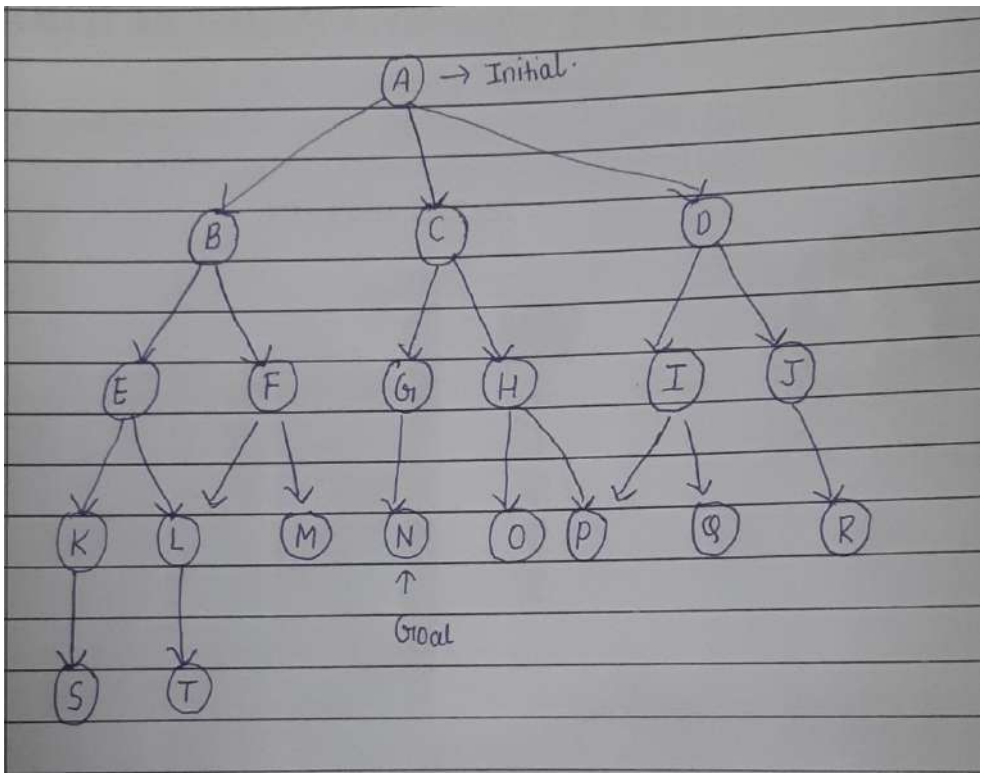


Fig 1: Graph

1. **DFS(Depth first Search) :**

Program:

```
graph = {

'A':['B','C','D'],
'B':['E','F'],
'C':['G','H'],
'D':['I','J'],
'E':['K','L'],
'F':['L','M'],
'G':['N'],
'H':['O','P'],
'I':['P','Q'],
'J':['R'],
'K':['S'],
'L':['T'],
  'M':[],   'N':[],
'O':[],
  'P':['U'],
```

```
  'Q':[],   'R':[],
'S':[],
```

```
  'T':[],
  'U':[] } print(graph) inp = input("Enter a goal node")
visited = set() # Set to keep track of visited nodes of
graph.
 def dfs(visited, graph, node):  #function for
dfs      if node not in visited:         if node
== inp:             print( node)
exit()         print (node)
visited.add(node)         for neighbour in
graph[node]:
            dfs(visited, graph, neighbour)

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

Output:

```
PS C:\Users\USER\Desktop\python folder> & C:/python/python.exe "c:/Users/USER/Desktop/python folder/DFS1.py"
{'A': ['B', 'C', 'D'], 'B': ['E', 'F'], 'C': ['G', 'H'], 'D': ['I', 'J'], 'E': ['K', 'L'], 'F': ['L', 'M'], 'G': [
'N'], 'H': ['O', 'P'], 'I': ['P', 'Q'], 'J': ['R'], 'K': ['S'], 'L': ['T'], 'M': [], 'N': [], 'O': [], 'P': ['U'],
 'Q': [], 'R': [], 'S': [], 'T': [], 'U': []}
Enter a goal node : N
Following is the Depth-First Search
A
B
E
K
S
L
T
F
M
C
G
N
PS C:\Users\USER\Desktop\python folder>
```

2. IDLS(Iterative Depth Limited Search):

Program:

```
def DLS(start, goal, path, level, maxD):
    path.append(start)
    if start == goal:
        return path
    if level == maxD:
        return False

    for child in graph[start]:
        if DLS(child, goal, path, level + 1, maxD):
            return path
    return False


nodes = input("\n\tEnter Graph Nodes: ")
nodes = string_to_list(nodes)

for node in nodes:
    children = input("\tEnter Children for the Node " + node + ": ")
    children = string_to_list(children)
    graph[node] = children

while True:
    goal = input("\n\tEnter Goal Node: ")
    if goal == 'exit' or goal == 'EXIT':
        break
    goal = string_to_list(goal)
    goal = goal[0]
```

```
        maxD = int(input("\tEnter Graph depth:- "))
        start = list(graph)[0]
        i = 0
        output = ''

        print("\n\t Searching for '" + goal + "'")
        while i <= maxD:
            print('\n')
            path = list()
            res = DLS(start, goal, path, 0, i)
            print('\tlevel: ', i)
            if (res):
                print("\tPath: ", end="")
                pathFormat = ''
                for j in range(len(path)):
                    if j == len(path) - 1:
                        pathFormat += path[j]
                    else:
                        pathFormat += path[j] + ' --> '
                print(pathFormat)
                output = "\tGoal Reached!"
                path.clear()
            else:
                print("\tPath: ", end="")
                pathFormat = ''
                for j in range(len(path)):
                    if j == len(path) - 1:
                        pathFormat += path[j]
                    else:
                        pathFormat += path[j] + ' --> '
                print(pathFormat)
                print("\tNo path available for the goal node in level", i)
            i = i + 1
        if output == '':
            output = "\tNode '"+goal+"' is not present in the graph!"
        print(output)
```

Output:

```
PS C:\Users\USER\Desktop\python folder> & C:/python/python.exe "c:/Users/USER/Desktop/python folder/new.py"
Enter Goal Node : N

level 0
Path : ['A']
No Goal Node Found

level 1
Path : ['A','B','C','D']
No Goal Node Found

level 2
Path : ['A','B','E','F','C','G','H','D','I','J']
No Goal Node Found

level 3
Path : ['A','B','E','K','L','F','M','C','G','N']
Goal Node Found
PS C:\Users\USER\Desktop\python folder>
```

Smt. Indira Gandhi College of Engineering
Ghansoli – Navi Mumbai

Computer Engineering Department
Academic Year 2022-23 (Even Sem)

Student Name: Sanskruti Palekar Roll No: 32 Class: TE Sem: VI

Course Name: Artificial Intelligence Lab

Course Code: CSL604

Experiment No. 04

Experiment Title: To Implement A* search – Informed Search Algorithm for problem solving in AI

Date of Performance	Date of Submission	Marks (10)					Sign / Remark
		A	B	C	D	E	
		2	3	2	2	1	
		Total Marks					

A: On Time Submission B: Understanding C: Analytical Skill

D: Critical Thinking E: Presentation

Experiment No 4: To Implement A* Search – Informed Search Algorithm for problem solving in AI.

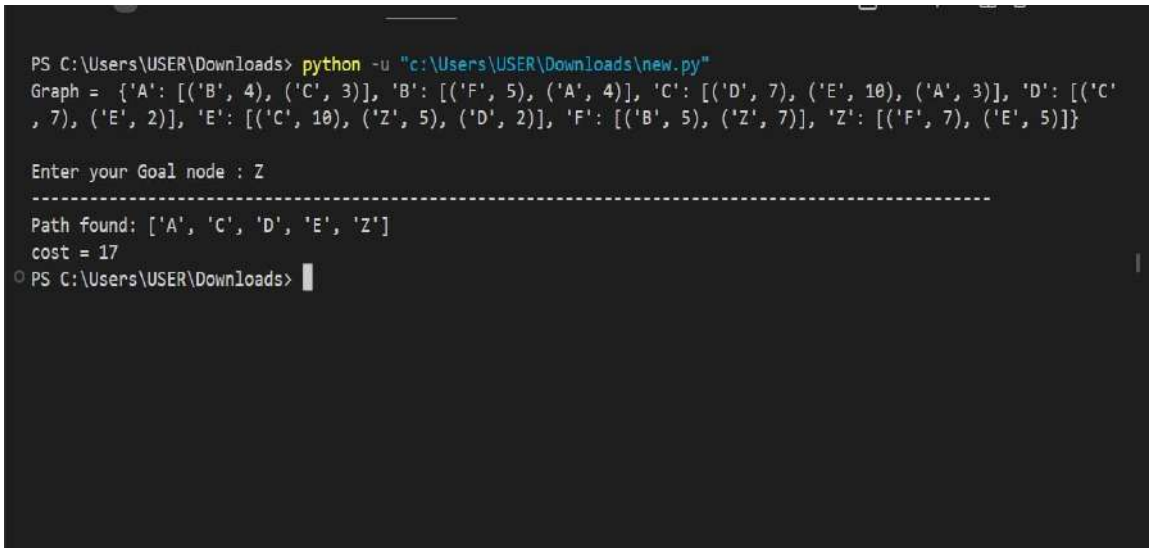
Program

```
def aStarAlgo(start_node, stop_node):
    open_set =
set(start_node)
closed_set = set()     g = {}
parents = {}
g[start_node] = 0
    parents[start_node] =
start_node     i=1     path_list=[]
while len(open_set) > 0:

        n = None
        #node with lowest f() is found         for v in open_set:
if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
```

```
                if n == stop_node or Graph_nodes[n]
== None:
            pass         else:             for
(m, weight) in get_neighbors(n):
                                  if m not in open_set and m
not in closed_set:
                    open_set.add(m)
parents[m] = n                      g[m] = g[n]
+ weight                 else:
if g[m] > g[n] + weight:
g[m] = g[n] + weight
parents[m] = n                          if m in
closed_set:
closed_set.remove(m)
open_set.add(m)
         if n == None:
print('Path does not exist!')
return None          if n == stop_node:
            path = []
while parents[n] != n:
                path.append(n)
n = parents[n]
path.append(start_node)
path.reverse()
             print("-------------------------------------------------------
---------------------------------------")
             print('Path found:
{}'.format(path))             print("cost
=",g[goal])
            # print(cost)
return path
open_set.remove(n)
closed_set.add(n)
print('Path does not exist!')
return None


def get_neighbors(v):
if v in Graph_nodes:
        return Graph_nodes[v]
else:
        return None
```

```
 def
heuristic(n):
H_dist = {
        'A': 14,
        'B': 12,
        'C': 11,
        'D': 6,
        'E': 4,
        'F': 11,
        'Z': 0
    }     return
H_dist[n]

Graph_nodes = {
    'A': [('B', 4), ('C', 3)],
    'B': [('F', 5),('A',4)],
    'C': [('D', 7), ('E', 10), ('A', 3)],
    'D': [ ('C', 7), ('E', 2)],
    'E': [('C', 10), ('Z', 5),('D', 2)],
    'F': [('B', 5), ('Z', 7)],
    'Z': [('F',7),('E',5)]
} print("Graph = ",Graph_nodes) goal =
input("\nEnter your Goal node : ")
aStarAlgo('A', goal)
```

Output:



	Smt. Indira Gandhi College of Engineering Ghansoli – Navi Mumbai Computer Engineering Department Academic Year 2022-23 (Even Sem)

Student Name: Sanskruti Palekar Roll No: 32 Class: TE Sem: VI Course

Name: Artificial Intelligence Lab

Course Code: CSL604

Experiment No. 05

Experiment Title: To Implement Hill Climbing Algorithm – SIMPLE & STEEPEST variation for problem solving in AI .

Date of Performance	Date of Submission	Marks (10)					Sign / Remark
			A B C		D		
		2	3	2	2	1	
		Total Marks					

A: On Time Submission B: Understanding C: Analytical Skill D: Critical Thinking E: Presentation

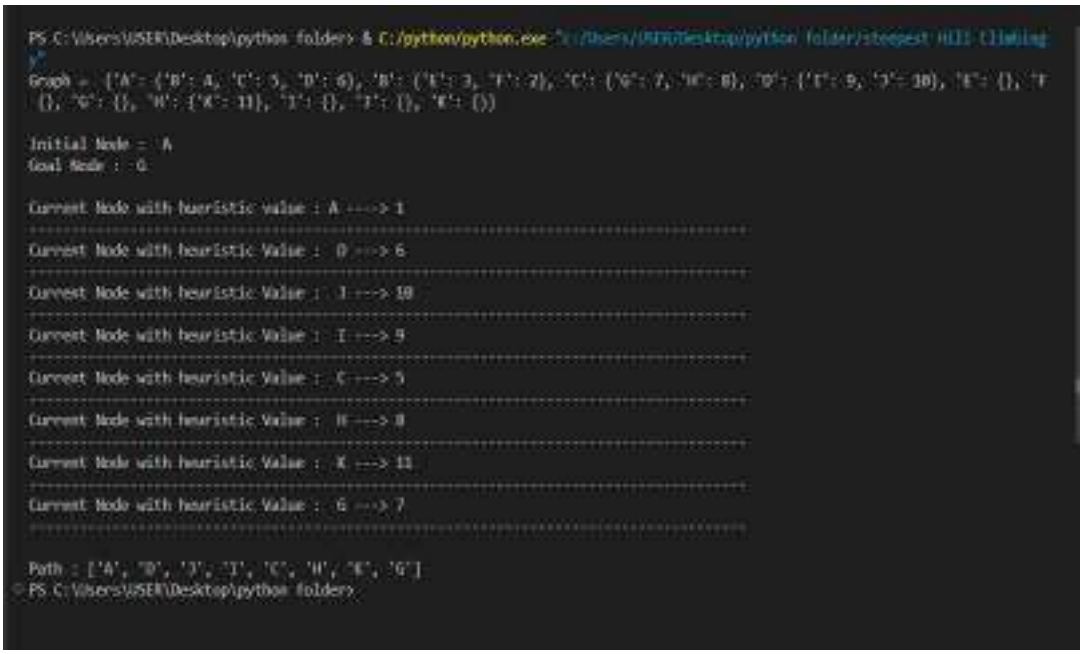
Experiment No 5: To Implement Hill Climbing – SIMPLE and STEEPEST Variation for problem solving in AI1. Steepest Hill climbing

Program:

```
graph={
    'A':{'B':4,'C':5,'D':6},
    'B':{'E':3, 'F':2},
    'C':{'G':7,'H':8},
    'D': {'I':9,'J':10},
    'E':{},
    'F':{},
    'G':{},
    'H':{'K':11},
    'I':{},
    'J':{},
    'K':{}
}

print("Graph = ",graph)
start = 'A' heuristic=
1 pre_hre = -1 path =
[] route = [] visited
=[]
nodes = []

local_maxima=[]
Goal = 'G'
path.append(start)
visited.append(heuristic)
nodes.append(start)
route.append(start)
print("\nInitial Node : ", start)
print("Goal Node : ", Goal,"\n")
print("Current Node with hueristic value :",start,"---->" ,heuristic)


def fun(hue, pre_hr, start,goal):
    node = start
    pre_hre = pre_hr
    heuristic = hue
    while pre_hre != heuristic:
        pre_hre = heuristic child =
        graph[node].items() if node
        == goal:
            return True
        for item in child:
            if item[1] not in local_maxima:
                if item[1] > heuristic:
                    heuristic = item[1]
                    path.append(item[0])
                    node = item[0]
```

```
            if heuristic not in visited:
                print("------------------------------------------------------------
- ----------------------------") print("Current Node with heuristic
                Value : ",node ,'--->',
heuristic)
                route.append(node)


            if heuristic not in visited or node not in nodes:
            visited.append(heuristic) nodes.append(node) #
            else:
            # visited.pop(heuristic)
            # nodes.pop(node)

        local_maxima.append(heuristic)
        visited.pop(-1) nodes.pop(-1) if
        (len(visited) == 1 ):
        fun(visited[0],-1,nodes[0],goal)
        else:
            fun(visited[-1],visited[-2],nodes[-1],goal)
fun(1,-1,'A','G')
print("------------------------------------------------------------------------
- ---------------")
print("\nPath :",route)
```

Output:



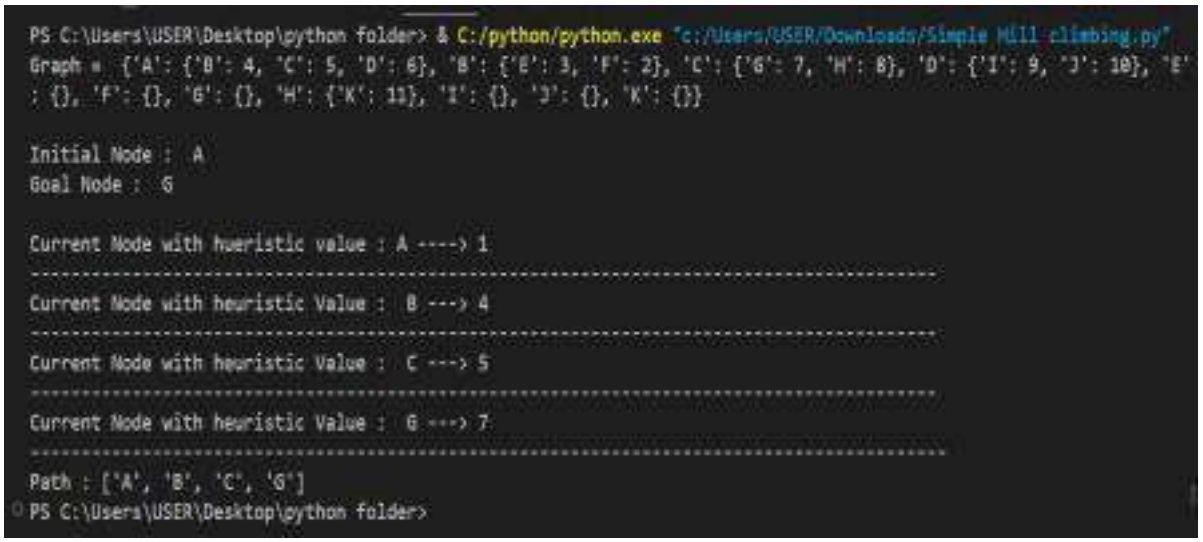
2. Simple Hill Climbing

Program:

```
graph={
    'A':{'B':4,'C':5,'D':6},
    'B':{'E':3, 'F':2},
    'C':{'G':7,'H':8},
    'D': {'I':9,'J':10},
    'E':{},
    'F':{},
    'G':{},
    'H':{'K':11},
    'I':{},
    'J':{},
    'K':{}
}
print("Graph = ",graph)
start = 'A'
heuristic= 1
pre_hre = -1
path = []
visited =[]
nodes = []
Goal = 'G'
local_maxima=[]
path.append(start)
visited.append(heuristic)
nodes.append(start)
print("\nInitial Node : ", start)

print("Goal Node : ", Goal,"\n") print("Current Node with hueristic
value :",start,"---->" ,heuristic) def fun(hue, pre_hr, start,goal):
    node = start
    pre_hre = pre_hr
    heuristic = hue
    while pre_hre != heuristic:
        pre_hre = heuristic child =
        graph[node].items()

        if node == goal:
            return True
        for item in child:
            if item[1] not in local_maxima:
                if item[1] > heuristic:
                heuristic = item[1]
                path.append(item[0]) node =
                item[0] break
        if heuristic not in visited:
            print("----------------------------------------------------------
```

```
- ------------------------------") print("Current Node with heuristic
            Value : ",node ,'--->',
heuristic)

        if heuristic not in visited or node not in nodes:
            visited.append(heuristic) nodes.append(node)


    local_maxima.append(heuristic)
    visited.pop(-1) nodes.pop(-1) if
    (len(visited) == 1 ):
    fun(visited[0],-1,nodes[0],goal)

    else:
        fun(visited[-1],visited[-2],nodes[-1],goal)


fun(1,-1,'A','G')
print("----------------------------------------------------------------------------
- --------------------")
print("Path :",path)
```

Output:



1. SUM OF N NATURAL NOS. USING RECURSION

```
sum_n(0, 0).
sum_n(N, Sum) :-
   N > 0,
   N1 is N - 1,
   sum_n(N1, Sum1),
   Sum is N + Sum1.
```

```
GNU Prolog 1.5.0 (64 bits)
Compiled Jul  8 2021, 12:33:56 with cl
Copyright (C) 1999-2021 Daniel Diaz

compiling C:/GNU-Prolog/examples/ExamplesC/examp.pl for byte code...
C:/GNU-Prolog/examples/ExamplesC/examp.pl compiled, 6 lines read - 836 bytes written, 11 ms
| ?- sum_n(5,Sum).

Sum = 15 ?
```

2. COMPUTING FACTORIAL OF A POSITIVE NUMBER

```
fact(0,1).
 fact(N,F):-
(
      N>0 ->
      (
         N1 is N-1,
         fact(N1,F1),
         F is N*F1
       )
   ).
```

```
GNU Prolog 1.5.0 (64 bits)
Compiled Jul  8 2021, 12:33:56 with cl
Copyright (C) 1999-2021 Daniel Diaz

compiling C:/GNU-Prolog/examples/ExamplesC/examp.pl for byte code...
C:/GNU-Prolog/examples/ExamplesC/examp.pl compiled, 11 lines read - 875 bytes written, 11 ms
| ?- fact(5,F).

F = 120 ?
```

3. ADD THE ELEMENTS OF AN INTEGER LIST

```
sum_list([], 0).
sum_list([H|T], Sum) :-
   sum_list(T, Rest),
   Sum is H + Rest.
```

length([], 0). length([_|T], N) :- length(T, N1),

```
GNU Prolog 1.5.0 (64 bits)
Compiled Jul  8 2021, 12:33:56 with cl
Copyright (C) 1999-2021 Daniel Diaz

compiling C:/GNU-Prolog/examples/ExamplesC/examp.pl for byte code...
C:/GNU-Prolog/examples/ExamplesC/examp.pl compiled, 3 lines read - 652 bytes written, 10 ms
error: C:/GNU-Prolog/examples/ExamplesC/examp.pl:1: native code procedure sum_list/2 cannot be redefined (igno
| ?- sum_list([1, 2, 3, 4, 5], Sum).

Sum = 15

yes
| ?- |
```

4. COMPUTE THE LENGTH OF A LIST.

N is N1 + 1.

```
GNU Prolog 1.5.0 (64 bits)
Compiled Jul  8 2021, 12:33:56 with cl
Copyright (C) 1999-2021 Daniel Diaz

compiling C:/GNU-Prolog/examples/ExamplesC/examp.pl for byte code...
C:/GNU-Prolog/examples/ExamplesC/examp.pl compiled, 3 lines read - 634 bytes written, 8 ms
error: C:/GNU-Prolog/examples/ExamplesC/examp.pl:1: native code procedure length/2 cannot be redefined (ignore
| ?- length([1, 2, 3, 4, 5], N).

N = 5

yes
| ?-
```

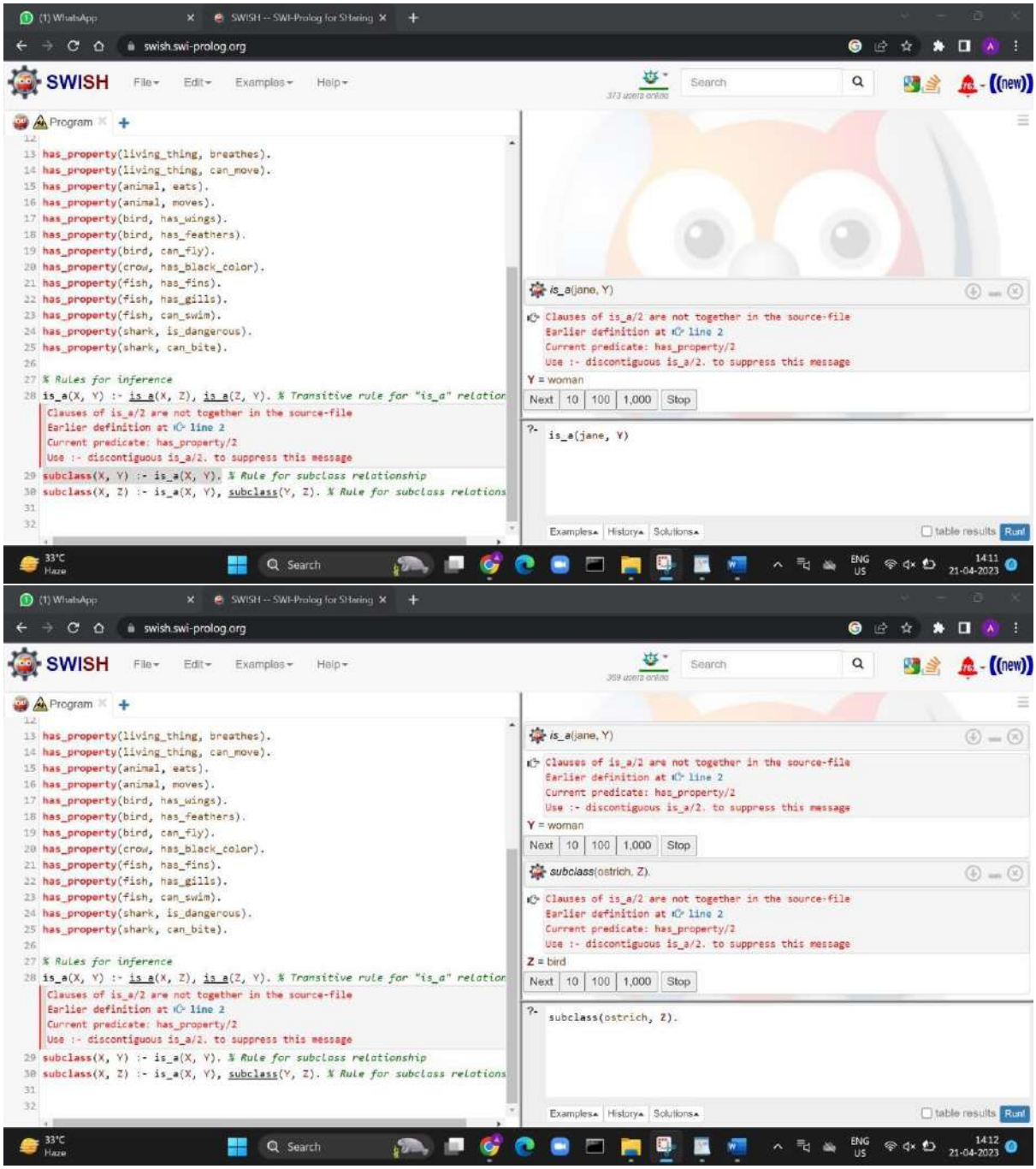
```
GNU Prolog 1.5.0 (64 bits)
Compiled Jul  8 2021, 12:33:56 with cl
Copyright (C) 1999-2021 Daniel Diaz

compiling C:/GNU-Prolog/examples/ExamplesC/examp.pl for byte code...
C:/GNU-Prolog/examples/ExamplesC/examp.pl compiled, 1 lines read - 444 bytes written, 7 ms
error: C:/GNU-Prolog/examples/ExamplesC/examp.pl:1: native code procedure member/2 cannot be redefined (ignore
| ?- member(3, [1, 2, 3, 4, 5]).

true ?

yes
| ?- member(6, [1, 2, 3, 4, 5]).

no
| ?- |
```

5. DETERMINE IF A GIVEN VALUE IS A MEMBER OF THE LIST

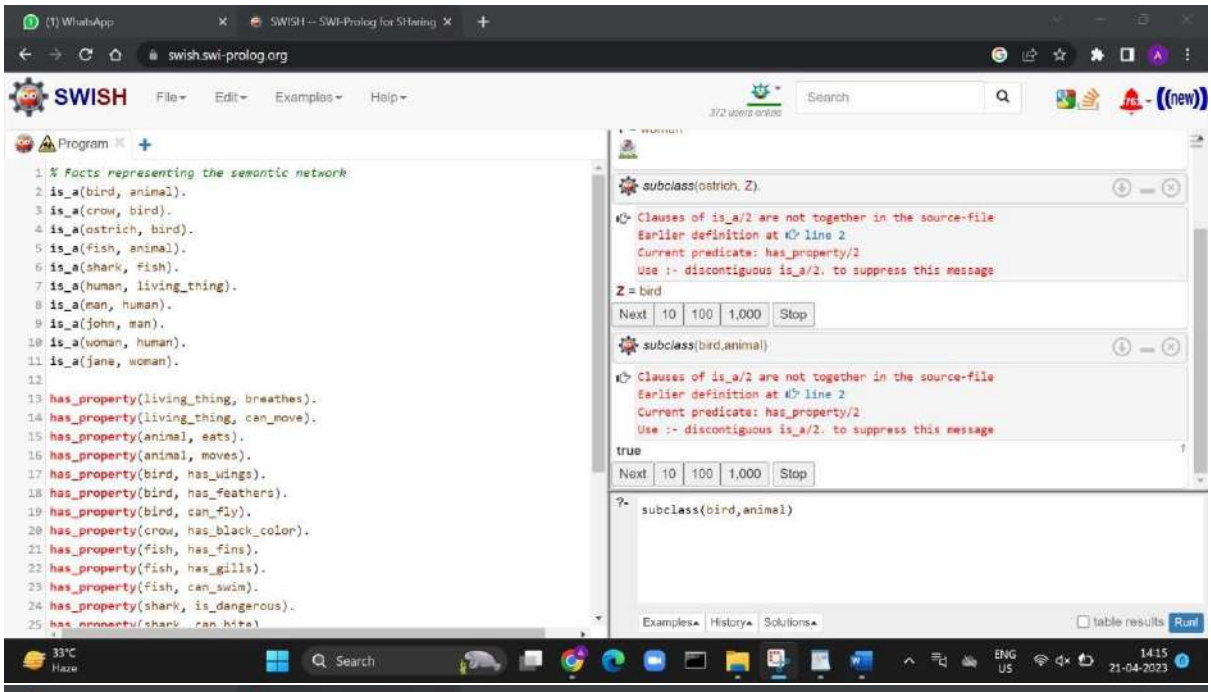
member(X, [X|_]). member(X, [_|T]) :- member(X, T).**Input:**

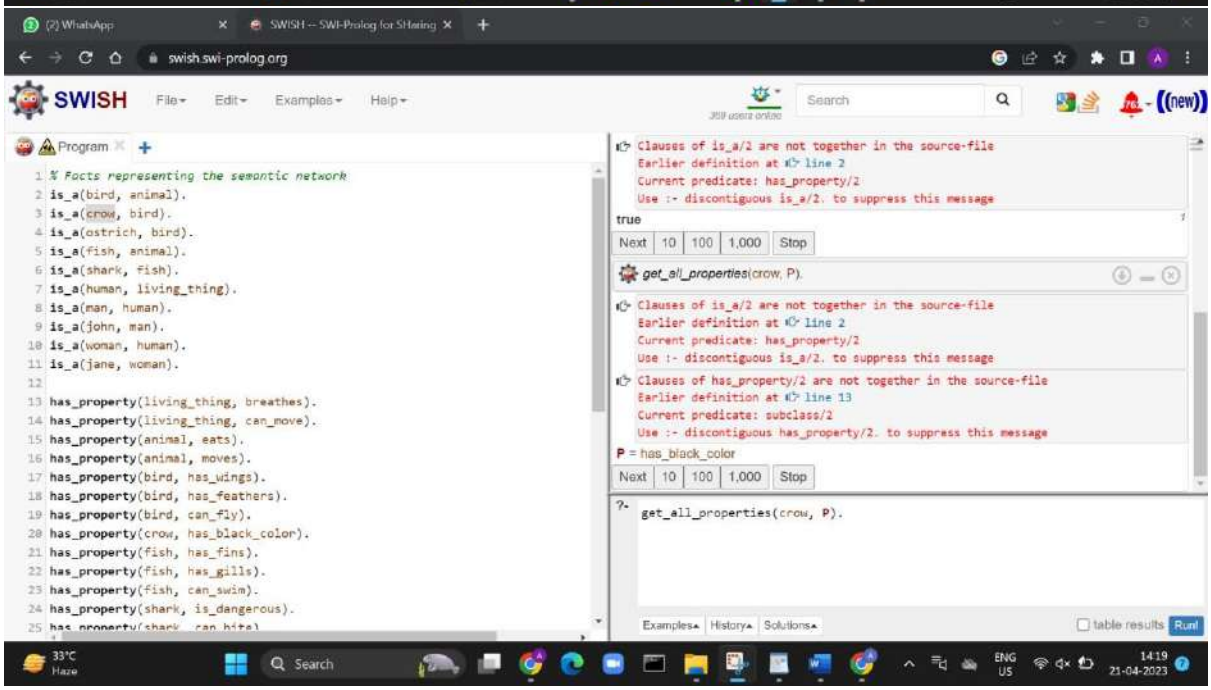
```
% Facts representing the semantic network

is_a(bird, animal). is_a(crow, bird).

is_a(ostrich, bird). is_a(fish, animal).

is_a(shark, fish).

is_a(human, living_thing). is_a(man,

human). is_a(john, man). is_a(woman,

human). is_a(jane, woman).

has_property(living_thing, breathes).

has_property(living_thing,

can_move). has_property(animal,

eats). has_property(animal, moves).

has_property(bird, has_wings).

has_property(bird, has_feathers).

has_property(bird, can_fly).

has_property(crow, has_black_color). has_property(fish, has_fins).

has_property(fish, has_gills). has_property(fish, can_swim).

has_property(shark, is_dangerous). has_property(shark, can_bite). % Rules

for inference is_a(X, Y) :- is_a(X, Z), is_a(Z, Y). % Transitive rule for

"is_a" relationship subclass(X, Y) :- is_a(X, Y). % Rule for subclass

relationship subclass(X, Z) :- is_a(X, Y), subclass(Y, Z). % Rule for

subclass relationship

% Rule for inferring properties
has_property(X, P) :- is_a(X, Y), has_property(Y, P). % Inherit property from superclass

% Query for getting all properties of an object get_all_properties(X,

P) :- has_property(X, P).
```

Output:


(1) WhatsApp
SWISH -- SWI-Prolog for SHaring
swish.swi-prolog.org
SWISH
File
Edit
Examples
Help
373 users online
Search
Program
has_property(living_thing, breathes).
has_property(living_thing, can_move).
has_property(animal, eats).
has_property(animal, moves).
has_property(bird, has_wings).
has_property(bird, has_feathers).
has_property(bird, can_fly).
has_property(crow, has_black_color).
has_property(fish, has_fins).
has_property(fish, has_gills).
has_property(fish, can_swim).
has_property(shark, is_dangerous).
has_property(shark, can_bite).
% Rules for inference
is_a(X, Y) :- is_a(X, Z), is_a(Z, Y). % Transitive rule for "is_a" relation
Clauses of is_a/2 are not together in the source-file
Earlier definition at line 2
Current predicate: has_property/2
Use :- discontiguous is_a/2. to suppress this message
subclass(X, Y) :- is_a(X, Y). % Rule for subclass relationship
subclass(X, Z) :- is_a(X, Y), subclass(Y, Z). % Rule for subclass relations
is_a(jane, Y)
Clauses of is_a/2 are not together in the source-file
Earlier definition at line 2
Current predicate: has_property/2
Use :- discontiguous is_a/2. to suppress this message
Y = woman
Next 10 100 1,000 Stop
?- is_a(jane, Y)
Examples History Solutions
table results
Run!
33°C Haze
Search
ENG US
14:11 21-04-2023

(1) WhatsApp
SWISH -- SWI-Prolog for SHaring
swish.swi-prolog.org
SWISH
File
Edit
Examples
Help
359 users online
Search
Program
has_property(living_thing, breathes).
has_property(living_thing, can_move).
has_property(animal, eats).
has_property(animal, moves).
has_property(bird, has_wings).
has_property(bird, has_feathers).
has_property(bird, can_fly).
has_property(crow, has_black_color).
has_property(fish, has_fins).
has_property(fish, has_gills).
has_property(fish, can_swim).
has_property(shark, is_dangerous).
has_property(shark, can_bite).
% Rules for inference
is_a(X, Y) :- is_a(X, Z), is_a(Z, Y). % Transitive rule for "is_a" relation
Clauses of is_a/2 are not together in the source-file
Earlier definition at line 2
Current predicate: has_property/2
Use :- discontiguous is_a/2. to suppress this message
subclass(X, Y) :- is_a(X, Y). % Rule for subclass relationship
subclass(X, Z) :- is_a(X, Y), subclass(Y, Z). % Rule for subclass relations
is_a(jane, Y)
Clauses of is_a/2 are not together in the source-file
Earlier definition at line 2
Current predicate: has_property/2
Use :- discontiguous is_a/2. to suppress this message
Y = woman
Next 10 100 1,000 Stop
subclass(ostrich, Z).
Clauses of is_a/2 are not together in the source-file
Earlier definition at line 2
Current predicate: has_property/2
Use :- discontiguous is_a/2. to suppress this message
Z = bird
Next 10 100 1,000 Stop
?- subclass(ostrich, Z).
Examples History Solutions
table results
Run!
33°C Haze
Search
ENG US
14:12 21-04-2023


(1) WhatsApp
SWISH -- SWI-Prolog for SHaring
swish.swi-prolog.org
SWISH
File
Edit
Examples
Help
Search
Program
% Facts representing the semantic network
is_a(bird, animal).
is_a(crow, bird).
is_a(ostrich, bird).
is_a(fish, animal).
is_a(shark, fish).
is_a(human, living_thing).
is_a(man, human).
is_a(john, man).
is_a(woman, human).
is_a(jane, woman).
has_property(living_thing, breathes).
has_property(living_thing, can_move).
has_property(animal, eats).
has_property(animal, moves).
has_property(bird, has_wings).
has_property(bird, has_feathers).
has_property(bird, can_fly).
has_property(crow, has_black_color).
has_property(fish, has_fins).
has_property(fish, has_gills).
has_property(fish, can_swim).
has_property(shark, is_dangerous).
subclass(ostrich, Z).
Clauses of is_a/2 are not together in the source-file
Earlier definition at line 2
Current predicate: has_property/2
Use :- discontiguous is_a/2. to suppress this message
Z = bird
Next 10 100 1,000 Stop
subclass(bird,animal)
true
?- subclass(bird,animal)
Examples History Solutions
table results
Run!
33°C Haze
Search
ENG US
14:15 21-04-2023


(2) WhatsApp
SWISH -- SWI-Prolog for SHaring
swish.swi-prolog.org
SWISH
File
Edit
Examples
Help
Search
Program
% Facts representing the semantic network
is_a(bird, animal).
is_a(crow, bird).
is_a(ostrich, bird).
is_a(fish, animal).
is_a(shark, fish).
is_a(human, living_thing).
is_a(man, human).
is_a(john, man).
is_a(woman, human).
is_a(jane, woman).
has_property(living_thing, breathes).
has_property(living_thing, can_move).
has_property(animal, eats).
has_property(animal, moves).
has_property(bird, has_wings).
has_property(bird, has_feathers).
has_property(bird, can_fly).
has_property(crow, has_black_color).
has_property(fish, has_fins).
has_property(fish, has_gills).
has_property(fish, can_swim).
has_property(shark, is_dangerous).
Clauses of is_a/2 are not together in the source-file
Earlier definition at line 2
Current predicate: has_property/2
Use :- discontiguous is_a/2. to suppress this message
true
Next 10 100 1,000 Stop
get_all_properties(crow, P).
Clauses of has_property/2 are not together in the source-file
Earlier definition at line 13
Current predicate: subclass/2
Use :- discontiguous has_property/2. to suppress this message
P = has_black_color
?- get_all_properties(crow, P).
Examples History Solutions
table results
Run!
33°C Haze
Search
ENG US
14:19 21-04-2023

Smt. Indira Gandhi College Of Engineering
Plot No.1, Sector 8, Ghansoli, Navi Mumbai-400 701.
COMPUTER ENGINEERING DEPARTMENT

!

Roll No: 32

EXPERIMENT NO: 08

Expt Title: EXPT 8 : Implementing Bayesian Network: Burglary Alarm Problem

DOP/DOS	Marks (10)					Sign / Remark
	A	B	C	D	E	
	2	3	2	2	1	
	Total Marks					

A: Prerequisite Knowledge
B: Implementation
C: Oral
D: Content
E: Punctuality& Discipline

PROGRAM:

```
import pgmpy.models
import pgmpy.inference
import networkx as nx
import pylab as plt
# Create a bayesian network
model = pgmpy.models.BayesianNetwork([('Burglary', 'Alarm'),
                                      ('Earthquake', 'Alarm'),
                                      ('Alarm', 'JohnCalls'),
                                      ('Alarm', 'MaryCalls')])
# Define conditional probability distributions (CPD)
# Probability of burglary (True, False)
cpd_burglary = pgmpy.factors.discrete.TabularCPD('Burglary', 2, [[0.001], [0.999]])
# Probability of earthquake (True, False)
cpd_earthquake = pgmpy.factors.discrete.TabularCPD('Earthquake', 2, [[0.002], [0.998]])
# Probability of alarm going of (True, False) given a burglary and/or earthquake
cpd_alarm = pgmpy.factors.discrete.TabularCPD('Alarm', 2, [[0.95, 0.94, 0.29, 0.001],
                                                           [0.05, 0.06, 0.71, 0.999]],
                                              evidence=['Burglary', 'Earthquake'],
                                              evidence_card=[2, 2])
# Probability that John calls (True, False) given that the alarm has sounded
cpd_john = pgmpy.factors.discrete.TabularCPD('JohnCalls', 2, [[0.90, 0.05],
                                                              [0.10, 0.95]],
                                             evidence=['Alarm'],
                                             evidence_card=[2])
# Probability that Mary calls (True, False) given that the alarm has sounded
cpd_mary = pgmpy.factors.discrete.TabularCPD('MaryCalls', 2, [[0.70, 0.01],
                                                              [0.30, 0.99]],
                                             evidence=['Alarm'],
                                             evidence_card=[2])
# Add CPDs to the network structure
model.add_cpds(cpd_burglary, cpd_earthquake, cpd_alarm, cpd_john, cpd_mary)
# Check if the model is valid, throw an exception otherwise
model.check_model()
# Print probability distributions
print('Probability distribution, P(Burglary)')
print(cpd_burglary)
print()
print('Probability distribution, P(Earthquake)')
print(cpd_earthquake)
```

```
print()
print('Joint probability distribution, P(Alarm | Burglary, Earthquake)')
print(cpd_alarm)
print()
print('Joint probability distribution, P(JohnCalls | Alarm)')
print(cpd_john)
print()
print('Joint probability distribution, P(MaryCalls | Alarm)')
print(cpd_mary)
print()
# Plot the model
nx.draw(model, with_labels=True)
plt.savefig('C:\\DATA\\Python-data\\bayesian-networks\\alarm.png')
plt.close()
# Perform variable elimination for inference
# Variable elimination (VE) is a an exact inference algorithm in bayesian networks
infer = pgmpy.inference.VariableElimination(model)
# Calculate the probability of a burglary if John and Mary calls (0: True, 1: False)
posterior_probability = infer.query(['Burglary'], evidence={'JohnCalls': 0, 'MaryCalls': 0})
# Print posterior probability
print('Posterior probability of Burglary if JohnCalls(True) and MaryCalls(True)')
print(posterior_probability)
print()
# Calculate the probability of alarm starting if there is a burglary and an earthquake (0: True,
False)
posterior_probability = infer.query(['Alarm'], evidence={'Burglary': 0, 'Earthquake': 0})
# Print posterior probability
print('Posterior probability of Alarm sounding if Burglary(True) and Earthquake(True)')
print(posterior_probability)
print()
```