Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that ~~Mr.~~/Miss Niyati Savant

of Computer (C31) Department, Semester VI with

Roll No. 2103156 has completed a course of the necessary

experiments in the subject AI under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2023 - 2024

Teacher In-Charge

Head of the Department

Date 15/04/2024

Principal

TSEC

CONTENTS

Title: The Impact of Artificial Intelligence and Robotics on Higher Education

Research paper – https://rdcu.be/dzewE

{ Exploring the impact of Artificial Intelligence and robots on higher education through literature-based design fictions }

Published in Springer

Introduction:

The potential impact of Artificial Intelligence (AI) and robots on higher education (HE) is a subject of considerable interest but remains complex and multifaceted. Currently, literature on the topic is fragmented, with separate discussions surrounding AI for education, learning analytics, and educational data mining. Moreover, AI is not a singular technology but an evolving concept, making its implications for HE difficult to fully grasp. While some applications of AI and robotics in HE are already mature, others are just beginning to be imagined, creating a temporal mix of past, present, and future impacts.

Understanding the holistic implications of AI and robots in HE is crucial due to the range of challenges they pose, including pedagogic, practical, ethical, and social justice considerations. The introduction of these technologies into educational settings is expected to encounter hurdles, as seen in past technological adoptions. Critiques in the educational literature often focus on concerns about dehumanizing the learning experience and the commercialization of education.

Domain

The domain of this study is higher education, encompassing universities, colleges, and research institutions. It focuses on how AI and robotics are reshaping teaching methodologies, research processes, and administrative functions in the HE sector. In classrooms and training centers, AI-powered adaptive learning tailors educational content to each student's needs, while plagiarism detection ensures academic integrity. Teachers and trainers can even leverage data analytics to predict student performance so they can intervene early if they spot problems.

AI has also played a significant role in democratizing access to education, especially for those in remote or underprivileged areas. AI-driven language translation tools and real-time transcription services have broken down language barriers, enabling students worldwide to access educational content from anywhere in the world. AI-powered virtual tutors can provide one-on-one support and guidance, supplementing traditional classroom instruction and making quality education accessible to a broader audience.

Summary

The review encompasses both AI and robotics, across various functions of HE, including teaching, research, administrative functions, and smart campus initiatives. Systematic searches combined with snowballing techniques were employed to gather relevant literature, focusing on recent applications and trends in AI and robotics, as well as their social implications. The review aims to provide a holistic view of the potential impacts of AI and robots on HE, laying the groundwork for the development of design fictions that instantiate these issues in a fictional form.

Table 1 Summary of the design fictions

	Technologies involved	Time frame	Genre	Area of application to HE
Fiction 1: AIDan, the teaching assistant	Intelligent tutoring systems, adaptive pedagogical agents, use of sensors to allow affective/embodied adaptivity	Future	Traditional design scenario	Teaching
Fiction 2: Footbotball	Robots	Future	Soliloquy	Extra curricula activity
Fiction 3: CriticalBot in conversation	Conversational agent	Present	Dialogue	Teaching
Fiction 4: The intelligent campus app	Smart campus: way-finding, nudging	Present/near future	Mundane, day in the life	Estates management/ Teaching
Fiction 5: Research Management Suite TM	Text and Data Mining, auto summarisation, auto writing	Future	Marketing and PR material	Research
Fiction 6: Verbatim minutes of University AI project steering committee: AI implementation phase 3	Not defined	Near future	Meeting minutes	All
Fiction 7: Dashboards	Data mining, conversational agents	Future	Soliloquy	Administration/Teaching
Fiction 8: Minnie, the AI admin assistant	Conversational agents	Near future	Surreal, cyberpunk dystopia	Administration, Wider social infrastructure

Problem Identified

The literature on AI in learning predominantly focuses on various types of systems designed to directly engage and support students' learning processes. These include Intelligent Tutoring Systems (ITS), Automatic Writing Evaluation (AWE) tools, Conversational Agents (Chatbots or virtual assistants), and Adaptive Pedagogical Agents. These technologies aim to personalize learning experiences, provide feedback, and facilitate interaction between learners and educational content. While some of these technologies, such as AWE and ITS, are relatively mature, there is a wide range of different systems within each category, each with its own capabilities and potential applications.

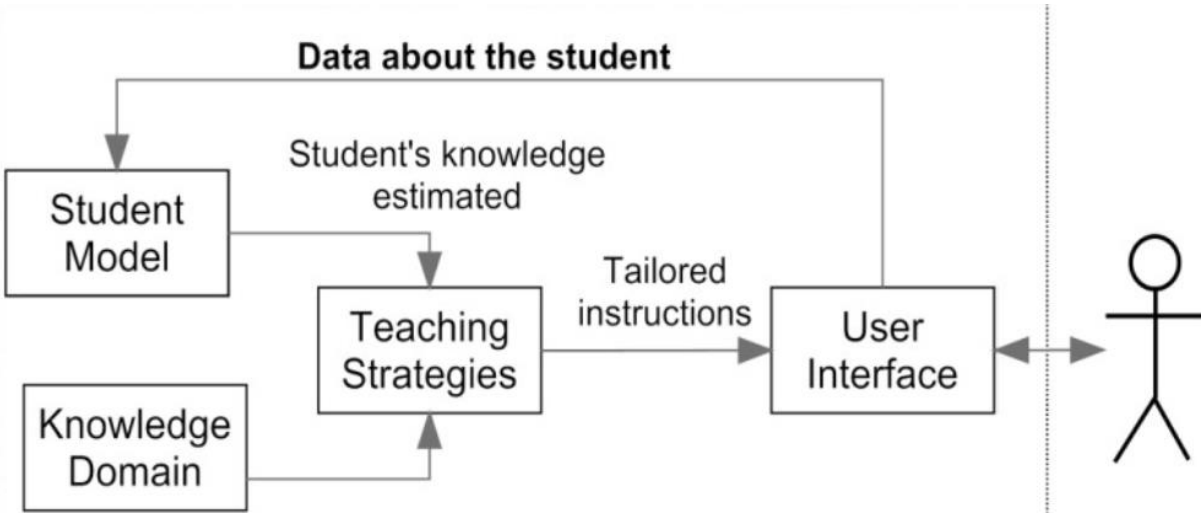
However, much of the literature primarily discusses the development and evaluation of these AI technologies from a technical standpoint, often neglecting pedagogical and ethical considerations. Some authors, like Luckin and Holmes, advocate for a broader perspective, envisioning how AI can address broader challenges in education, such as personalization, continuous monitoring of performance, and the development of 21st-century skills.

To illustrate this vision, a fictional narrative, inspired by their work, envisions an educational setting where AI plays a central role in supporting teaching and learning. In this narrative, AI serves as a teaching assistant, personalized learning experiences are continuously monitored and adjusted based on individual performance and emotional states, and students develop long-term relationships with AI companions. The narrative also highlights the potential benefits of AI in supporting lifelong learning and personalized services across various aspects of university life.

However, the narrative also acknowledges potential concerns, such as the implications of continuous monitoring for surveillance and the ethical implications of personalizing learning experiences through AI. This fictional scenario offers a glimpse into a possible future where AI transforms education but also raises questions about its broader societal implications and ethical considerations.

Agent studied and Algorithm or Search Techniques Used:

1] ITS



As its name suggests, an AI intelligent tutoring system is a computer-based learning system that uses artificial intelligence to provide human-like lessons without a human teacher.AI intelligent tutoring systems allow you to create a personalised learning experience that provides immediate instruction and feedback to learners, usually without human intervention.

Intelligent Tutoring Systems (ITS) utilize various AI-based technologies across different components to create personalized and adaptive learning experiences. Let's delve into each component and explore the AI-based technologies commonly used:

1. User Interface (UI):

1) Natural Language Processing (NLP): NLP enables ITS to understand, interpret, and generate human language. Techniques include:

a)Text Tokenization: Breaking down text into smaller units (tokens), such as words or sentences.

b)Part-of-Speech Tagging: Assigning grammatical tags (noun, verb, adjective, etc.) to each word in a sentence.

c)Named Entity Recognition (NER): Identifying and classifying named entities (e.g., people, organizations, locations) in text.

2) Speech Recognition: This technology converts spoken language into text, allowing users to interact with the ITS through voice commands. Techniques include:
   a) Acoustic Modelling: Modelling the acoustic characteristics of speech sounds to identify phonemes and words.
   b) Language Modelling: Predicting the sequence of words in a spoken utterance based on statistical language models.
3) Gesture Recognition: Gesture recognition systems interpret hand movements, gestures, or body poses made by users. Techniques include:
   a) Depth Sensing: Using depth-sensing cameras (e.g., Microsoft Kinect) to capture 3D information about the user's gestures.

b) Feature Extraction: Extracting relevant features from the captured gestures (e.g., hand shape, movement trajectory).
c) Eye Tracking: Eye tracking technology monitors and analyzes the movements and fixations of the user's eyes. Techniques include:
d) Corneal Reflection: Using infrared light to illuminate the eyes and track the reflections from the cornea.
e) Pupil Centre Corneal Reflection (PCCR): Calculating the gaze direction based on the position of the pupil and corneal reflection.

2. Teaching Strategies:

1) Machine Learning Algorithms: Various machine learning algorithms are employed to develop adaptive teaching strategies. These include:
   a) Decision Trees: Hierarchical models that make decisions based on the features of the input data.
   b) Neural Networks: Deep learning models composed of interconnected layers of artificial neurons that learn complex patterns.
   c) Reinforcement Learning: Learning through trial and error, where the system receives feedback (rewards or penalties) based on its actions.
2) Expert Systems: Expert systems utilize knowledge representation and inference mechanisms to emulate the decision-making of domain experts. Techniques include:
   a) Rule-Based Systems: Systems that use a set of rules (if-then statements) to make decisions or provide recommendations.
   b) Inference Engines: Engines that apply logical reasoning to derive new information from existing knowledge.
3) Adaptive Algorithms: These algorithms dynamically adjust teaching strategies based on real-time feedback and student performance. Techniques include:
   a) Adaptive Sequencing: Modifying the sequence of learning activities or content based on the student's progress and preferences.
   b) Adaptive Feedback: Tailoring feedback messages to address the specific needs and misconceptions of individual students.

3. Knowledge Domain:

1) Knowledge Representation: Techniques for representing and organizing domain-specific knowledge include:
   a) Semantic Networks: Graph-based structures representing concepts and their relationships.
   b) Ontologies: Formal representations of domain knowledge, specifying concepts, properties, and relationships.
2) Semantic Web Technologies: These technologies facilitate the integration and interoperability of educational resources, including:
   a) Resource Description Framework (RDF): A framework for describing and linking resources on the web.
   b) Web Ontology Language (OWL): A language for defining ontologies and specifying relationships between concepts.
3) Domain-Specific Reasoning Engines: Reasoning engines apply domain-specific rules and heuristics to infer new knowledge or solve problems. Techniques include:
   a) Forward Chaining: Starting with known facts and using rules to derive new conclusions.

b) Backward Chaining: Starting with a goal and working backward to determine the conditions necessary to achieve it.

4. Student Model:

1) Data Mining and Learning Analytics: Techniques for analysing student data and building predictive models include:
   a) Classification: Predicting discrete class labels (e.g., pass/fail) based on input features.
   b) Clustering: Grouping students into clusters based on similarities in their learning behaviours.
2) Bayesian Networks: Bayesian networks model the probabilistic relationships between variables in the student's learning environment. Techniques include:
   a) Parameter Learning: Estimating the parameters (probabilities) of the network based on observed data.
   b) Inference: Using the network to make probabilistic predictions or decisions about new data.
3) Cognitive Modelling: Techniques for simulating cognitive processes include:
   a) Production Systems: Representing cognitive processes as a set of production rules (if-then statements).
   b) ACT-R (Adaptive Control of Thought—Rational): A cognitive architecture that models human cognition using symbolic and procedural representations.

By leveraging these AI-based technologies, Intelligent Tutoring Systems can provide personalized, adaptive, and effective learning experiences tailored to the needs and preferences of individual learners.

Future Work:

Future research in this area may include further exploration of the ethical considerations surrounding AI and robotics in HE, the development of guidelines for responsible implementation, and the investigation of long-term societal impacts. Additionally, ongoing analysis of emerging trends and advancements in AI and robotics will be crucial for informing future strategies and practices in the higher education sector. Future work may also involve the development of innovative tools to support academic productivity, the exploration of new teaching methodologies, and the enhancement of student learning experiences through AI and robotics integration.

Niyati Savant
TE - C31
2103156

Experiment - 2

Aim - Implement DFS, DLS, DFID search algorithm in Python

Theory -

Depth first search, the tree is expanded depthwise ie deepest node of current branch of the ~~tread~~ tree as expanded. As it reaches the leaf, it backtracks to previous node.
Explored nodes with no descendants in fringe is removed.

- DFS uses LIFO fringe ie stack - the most recent topmost node is chosen first. As node is expanded, it is dropped from fringe and successors r added.
- It can be implemented recursively and non-recursively as well
- In DLS or depth limited search, DFS is done with only a predetermined depth limit to avoid the infinite loop condition.
- The nodes of specified depth limited are treated as if they don't have any successors.
Although it resolves the infinite-path problem, it introduces the incompleteness problem as the shallowest goal could be beyond specified depth
- To resolve this, the IDDFS or DFID i.e Depth first Iterative Deepening Search is implemented.
- IDDFS is a combination of BFS and DFS

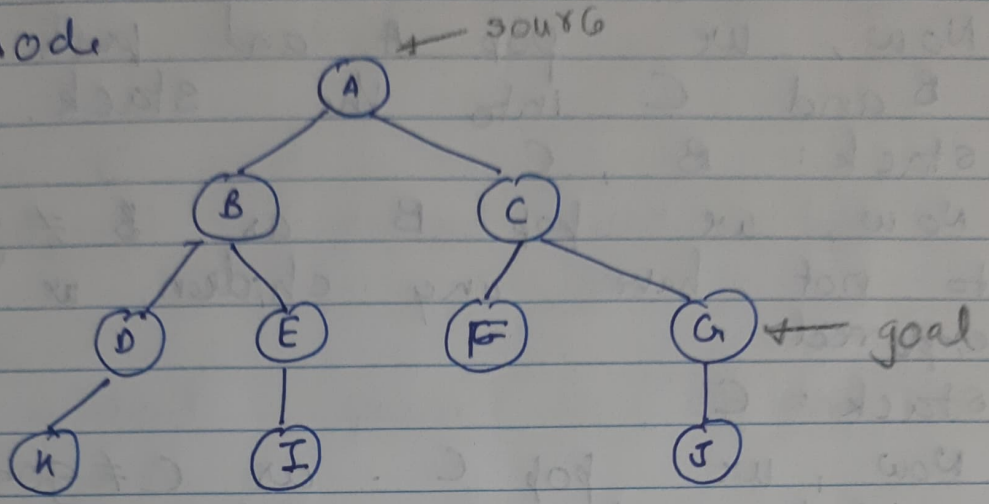
Niyati Savant
TE C31
2103156

- The search happens depth wise but at a time the depth limit is incremented by one. Hence it iteratively deepens down.
- It turns out to be BFS as it explores a complete layer of new nodes at each iteration before going on next layer by gradually increasing depth limit - first 0, then 1, 2 - - - until goal is found and thus guarantees a solution
- All these search technique are Undirectional or uninformed searched techniques as they use only information from problem defination

Parameters	DFS	DLS	IDDFS
Completeness	No	No	Yes
Optimality	No	No	Yes
Time Complexity	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space Complexity	$O(b^m)$	$O(b^l)$	$O(b^d)$

where m - maximum depth of any path in the tree
l - depth limit specified
d - depth of shallowest goal node
b - branching factor

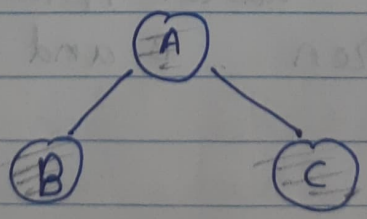
Eg-
Consider A as source node and G as goal node



→ Find goal node with IDBFS

1] For iteration 0, we consider only depth 0 i.e only source node 'A'. As 'A' is not a goal node & is treated as if it does not have successors, we have not found goal node yet & move to next iteration.

2] For i = 1, we consider depth - 1

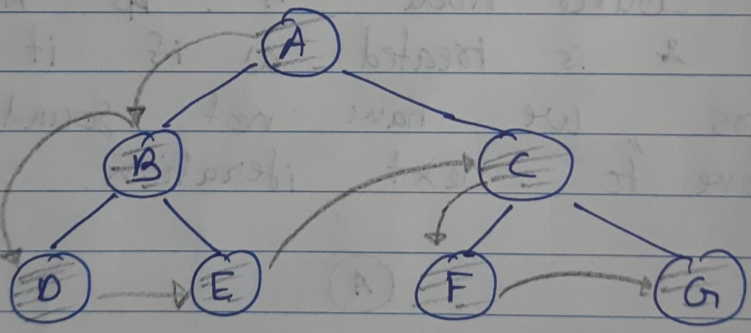


We start at 'A'. As it is not goal, we

Niyati Savant
TE C31
2103156

add it to the stack.
stack : A
Now, we pop A and push its children B and C into the stack.
stack : B , C
Now, we pop B as B ≠ goal and considered to not have any children, we considers it to be explored
stack : C
Now, we pop C. as C ≠ goal we have reached end but did not ~~get~~ go so we move to next iteration.

3] For i = 2 or depth 2.



. Start with source A. A is not goal, ~~now~~ so it is added into stack then popped and its children B and C are pushed.

Stack : B C

Next, pop B, B ≠ goal push children of B into stack

Niyati Savant
TE C31
2103156

Stack: D E C
Now pop D, D ≠ goal and is leaf. Thus push nothing into stack.

Stack: E C
Now, pop E, E ≠ goal and is leaf.

Stack: C
As C ≠ goal, push its children into stack after popping C
Stack: F G.
Pop F as F is not goal. and is leaf nothing is pushed into stack.

Stack: G
As G is the goal node, it is poped and we have found our shallowest goal

Path taken:

iteration 0: A
iteration 1: A → B → C
iteration 2: A → B → D → E → C → F → G

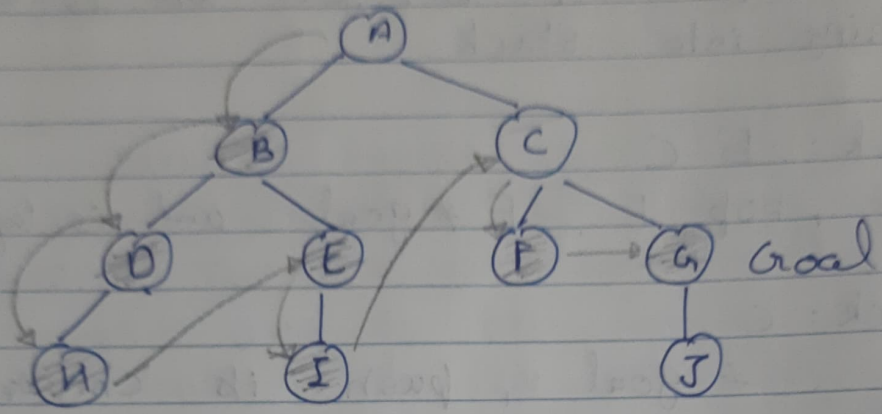
Instead, if we had followed DLS with depth 1, we would not have found the goal and in case we used DFS only the path would be -

(P.T.O)

Niyati Savant
TE - C31
2103156

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Path : A → B → D → H → E → I → C → F → G



- DFS is easy to implement and needs lesser memory as it stores only current path. However, it does not guarantee shortest path or completeness as it can get stuck in loops. It is used in path finding, cycle detection, topological sorting
- DLS avoids infinite loops by setting depth limit and control exploration depth in resource constrained environment but is not complete and can degrade performance if depth is not selected properly
- IDDFS is memory efficient, guarantees optimal and completeness and balances DFS and BFS. However it may explore some node multiple times at different depth levels and not suitable for graphs with high branching factors. Used for puzzle solving, game tree search and state space search.

Experiment 2

Aim - Implement DFS/DLS/DFID search algorithm in Python.

Code:

```
Tree = {
    "A": ['C', 'B'],
    "B": ['E', 'D'],
    "C": ['G', 'F'],
    "D": ['H'],
    "E": ['J', 'I'],
    "F": ['L', 'K'],
    "G": ['M'],
    "H": [],
    "I": [],
    "J": [],
    "K": [],
    "L": [],
    "M": []
}


def DFS(root, target, stack):
    while len(stack) != 0:
        current_node = stack.pop()
        if current_node == target:
            print("Found goal")
            break
        else:
            children = Tree[current_node]
            stack.extend(children)
        print(stack)


total_depth = 3


def DLS(current, limit, current_depth, goal, stack):
    stack.pop()
    if (current_depth > limit):
        return False
    if current == goal:
        return True
    else:
        children = Tree[current]
        stack.extend(children[::-1])
        print(stack)
        for child in children:
            if DLS(child, limit, (current_depth + 1), goal, stack):
                return True


def IDDFS(goal):
    limit = 0
    found = False
    stack = ['A']
```

```
    while not found:
        print(f"At depth limit {limit}:")
        found = DLS('A', limit, 0, goal, stack)
        stack = ['A']
        limit += 1
        if limit > total_depth:
            print("NOT Exist")
            break
    if found:
        print(f"Found at depth {limit - 1}")

print("Niyati's Code for DFS DLS & IDDFS")

print("The Tree structure is:{Parent:children}")
print(Tree)
want_to_continue = 1
while want_to_continue == 1:
    root_node = input("Enter Root Node: ")
    goal_node = input("Enter Goal Node: ")
    user_inp = input("What algorithm to use? Press 1 for DFS, 2 for DLS and 3 for IDDFS: ")
    stack = ['A']
    print(stack)
    if user_inp == '1':
        DFS(root_node, goal_node, stack)
        stack = ['A']
    elif user_inp == '2':
        limit = int(input("Enter depth limit: "))
        if DLS(root_node, limit, 0, goal_node, stack):
            print("Found within given depth")
        else:
            print("Not Found within given depth")
        stack = ['A']
    elif user_inp == '3':
        IDDFS(goal_node)
    else:
        print("Enter a valid number")
        stack = ['A']

    want_to_continue = int(input("Press 1 to continue and anything else to exit: "))
```

Output:

Niyati's Code for DFS DLS & IDDFS

The Tree structure is:{Parent : children}

{'A': ['C', 'B'], 'B': ['E', 'D'], 'C': ['G', 'F'], 'D': ['H'], 'E': ['J', 'I'], 'F': ['L', 'K'], 'G': ['M'], 'H': [], 'I': [], 'J': [], 'K': [], 'L': [], 'M': []}

Enter Root Node: A

Enter Goal Node: G

What algorithm to use? Press 1 for DFS, 2 for DLS and 3 for IDDFS: 1

['A']

['C', 'B']

['C', 'E', 'D']

['C', 'E', 'H']

['C', 'E']

['C', 'J', 'I']

['C', 'J']

['C']

['G', 'F']

['G', 'L', 'K']

['G', 'L']

['G']

Found goal

Press 1 to continue and anything else to exit: 1

Enter Root Node: A

Enter Goal Node: G

What algorithm to use? Press 1 for DFS, 2 for DLS and 3 for IDDFS: 2

Enter depth limit: 1

['A']

['B', 'C']

['B', 'F', 'G']

['D', 'E']

Not Found within given depth

Press 1 to continue and anything else to exit: 1

Enter Root Node: A

Enter Goal Node: G

What algorithm to use? Press 1 for DFS, 2 for DLS and 3 for IDDFS: 3

At depth limit 0:

['A']

['B', 'C']

At depth limit 1:

['A']

['B', 'C']

['B', 'F', 'G']

['D', 'E']

At depth limit 2:

['A']

['B', 'C']

['B', 'F', 'G']

Found at depth 2

Press 1 to continue and anything else to exit: 0

Niyati Savant
TE C31
2103156 Experiment - 3

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE
22/02/24

Aim - Implement BFS, UCS algorithm in python

Theory -

- As the name suggest, in BFS or Breadth First Search algorithm, the tree is expanded breadth wise. The root is expanded first and then its successors and so on. In turn, all nodes at a particular depth in the tree are expanded first and then search proceeds to next level node expansion
- The shallowest unexpanded node is chosen for expansion.
- A FIFO queue is used for the fringe and the newly inserted nodes will automatically be placed after their parents.
- Thus, Children nodes, which are deeper than their parents, go to the back of the queue and old nodes that are shallower are expanded first
- UCS or Uniform Cost Search, is a BFS with all paths having same cost and to make it work we make a simple extension. instead of expanding shallowest node in BFS, the node with lowest path cost is expanded first in UCS.
- It is implemented using priority queue ordered by path cost. Its implementation is same as BFS with the addition of extra check for a shorter path. The priority queue contains total cost

Niyati Savant
TE C31
2103156

from root to the node. UCS gives minimum path cost the maximum priority.
Thus it returns best cost path encountered first and will never go for other possible paths. The solution path is cost optimal ast algorithm never expands a node with cost greater than cost of shortest path in the tree

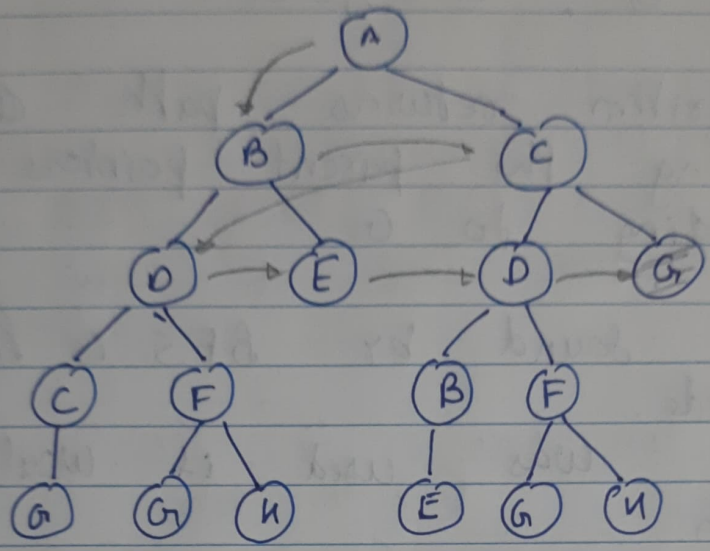
Parameters	BFS	UCS
Completeness	Yes	Yes
Optimality	Yes	Yes
Time Complexity	$O(b^d)$	$O(b^{C^*/\epsilon})$
Space Complexity	$O(b^d)$	$O(b^{C^*/\epsilon})$

where,

b - branching factor
d - depth of shallowest goal node
C^* - Cost of optimal solution
ϵ - Minimum cost of each action

Niyati Savant
TE - C31
2103156

Example - Source - A, Goal 'G' - BFS



→ 1] Queue : A
Remove A, expand it and push its children

2] Queue : B C
Remove B, expand and add D, E at end

→ 3] Queue : C D E
Remove C, expand and add D, G at end

4] Queue : D E D G
Remove D, expand and add C, F

5) Queue : E D G C F
Remove E, has no children - add nothin

6] Queue : D G C F
Remove D, expand and add B, F

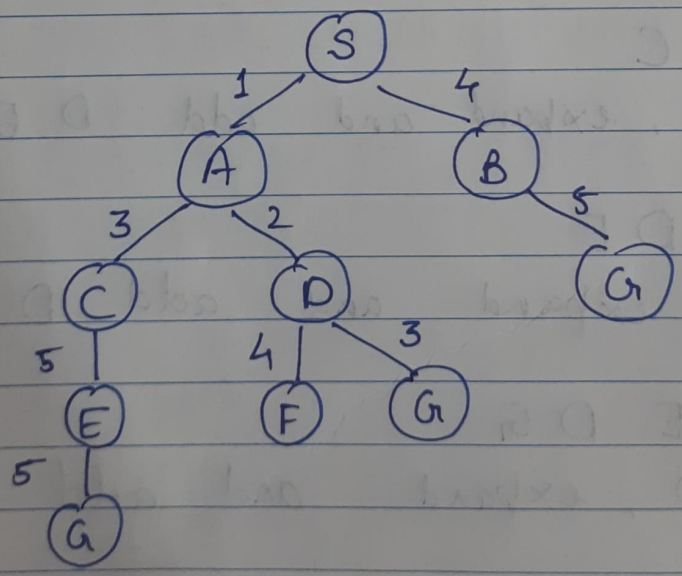
Niyati Savant
TE - C31
2103156

7] Queue - G C F B F
Remove G. G = goal.

The algorithm returns path A - C - G by following the parent pointers of nodes corresponding to G.

This G found by BFS is the shallowest goal node.
If DFS was used it would have found the path
A → B → D → C → G

Example -



1] Start from 'S'.
Queue : S(0)

Niyati Savant
TE - C31
2103156

2] Remove S and add A(1) and B(4)
Queue : A(1) B(4)

3] Remove A(1) as it is not goal, we enque its children C(4) and D(3)
Queue : ~~B(4)~~ D(3) B(4) C(4)
(In case of equal priorities alphabetical node is decided).

4] Remove D(3), it is not goal, enque children G(6) and F(7)
Queue : B(4), C(4), G(6), F(7)

5] Remove B(4), it is not goal, enque children G(9)
Queue : C(4), G(6), F(7), G(9)

6] Remove C(4), not goal, enque E(9)
Queue : G(6), F(7), E(9), G(9)

7] Remove G(6), G is goal, we found the optimal path

S → A → D → G
and path cost is 6.

For DFS path would be S → A → C → E → G and
for BFS path would be S → A → B → C → D → G.

Niyati Savant
TE - C31
2103156

BFS is guranteed to find shortest, path ie optional one in unweighted graph. It also uses less memory compared to DFS but may need more memory for graphs with large branching factor. They may be slower than DFS but are used for traversal like network analysis and finding shortest path.

UCS guranteed optimal solution and can handel graphs of non-uniform ~~width~~ edge weights and is also more memory efficient than A* search like algorithms. However, in worst cases, they have time complexity similar to DFS (especially in non-uniform edge costs) and cannot be used in complex or unknown cost functions. They are used in cases of road networks, optimal solution problems like resource allocation, scheduling etc

Experiment 3

AIM - Implement Breadth-First Search/UCS algorithm in Python.

Code –

```
Tree_BFS = {
"A": ['C', 'B'],
"B": ['E', 'D'],
"C": ['G', 'F'],
"D": ['H'],
"E": ['J', 'I'],
"F": ['L', 'K'],
"G": ['M'],
"H": [],
"I": [],
"J": [],
"K": [],
"L": [],
"M": []
}
def BFS(node,goal):
  queue = []
  queue.append(node)
  print(queue)
  while len(queue)!=0:
    node = queue[0]
    if queue[0]==goal:
      return("Goal node found")

    else:
      queue.pop(0)
      children = Tree_BFS[node]
      queue.extend(children)
      print(queue)
```

```
  return("Not exist After exploring all nodes")



def uniform_cost_search(goal, start):
        global graph,cost
        answer = []
        queue = []

        for i in range(len(goal)):
                answer.append(10**8)
        queue.append([0, start])

        visited = {}
        count = 0

        while (len(queue) > 0):

                queue = sorted(queue)
                p = queue[-1]
                del queue[-1]
                p[0] *= -1

                if (p[1] in goal):

                        index = goal.index(p[1])

                        if (answer[index] == 10**8):
                                count += 1

                        if (answer[index] > p[0]):
                                answer[index] = p[0]
```

```
                        del queue[-1]

                        queue = sorted(queue)
                        if (count == len(goal)):
                                return answer


                if (p[1] not in visited):
                        for i in range(len(graph[p[1]])):
                                queue.append( [(p[0] + cost[(p[1], graph[p[1]][i])])* -1,
graph[p[1]][i]])

                visited[p[1]] = 1

        return answer



graph,cost = [[] for i in range(8)],{}

graph[0].append(1)
graph[0].append(3)
graph[3].append(1)
graph[3].append(6)
graph[3].append(4)
graph[1].append(6)
graph[4].append(2)
graph[4].append(5)
graph[2].append(1)
graph[5].append(2)
graph[5].append(6)
graph[6].append(4)
```

```
cost[(0, 1)] = 2
cost[(0, 3)] = 5
cost[(1, 6)] = 1
cost[(3, 1)] = 5
cost[(3, 6)] = 6
cost[(3, 4)] = 2
cost[(2, 1)] = 4
cost[(4, 2)] = 4
cost[(4, 5)] = 3
cost[(5, 2)] = 6
cost[(5, 6)] = 3
cost[(6, 4)] = 7


goal = []


print("Niyati's Code for BFS & UCS")

print("The Tree structure is:{Parent:children}")
print(Tree_BFS)
want_to_continue = 1
while want_to_continue == 1:
    root_node = input("Enter Root Node: ")
    goal_node = input("Enter Goal Node: ")
    user_inp = input("What algorithm to use? Press 1 for BFS, 2 for UCS: ")
    stack = ['A']
    if user_inp == '1':
        print(stack)
        BFS(root_node, goal_node)
        stack = ['A']
```

```
    elif user_inp == '2':
        goal.append(int(goal_node))
        answer = uniform_cost_search(goal, int(root_node))
        print("Minimum cost from 0 to 6 is = ",answer[0])


    else:
        print("Enter a valid number")


    want_to_continue = int(input("Press 1 to continue and anything else to exit: "))
```

Output –

```
Niyati's Code for BFS & UCS
The Tree structure is:{Parent:children}
{'A': ['C', 'B'], 'B': ['E', 'D'], 'C': ['G', 'F'], 'D': ['H'], 'E': ['J', 'I'], 'F': ['L', 'K'], 'G': ['M'], 'H':
 [], 'I': [], 'J': [], 'K': [], 'L': [], 'M': []}
Enter Root Node: A
Enter Goal Node: G
What algorithm to use? Press 1 for BFS, 2 for UCS: 1
['A']
['A']
['C', 'B']
['B', 'G', 'F']
['G', 'F', 'E', 'D']
Press 1 to continue and anything else to exit: 1
Enter Root Node: 0
Enter Goal Node: 6
What algorithm to use? Press 1 for BFS, 2 for UCS: 2
Minimum cost from 0 to 6 is =  3
Press 1 to continue and anything else to exit:
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 4

Aim - Implement Greedy Best First Search / A* search algorithm in Python

Theory -

Greedy Best-First Search is an AI search algorithm that attempts to find the most promising path from a given starting point to a goal. It prioritizes paths that appear most promising regardless of whether or not it is actually the shortest. It works by evaluating cost of each possible path and expanding the one with lowest cost.

The algorithm works by using a heuristic function to determine which path is most promising. The heuristic function considers cost of current path and estimated cost of remaining paths. If cost of current is lower than estimated of remaing paths, choose current one.

Algorithm -

- It works by evaluating cost of each possible path and expanding the one with lowest cost.
- A heuristic function is used to determine most promising path
- It considers cost of current path and estimated

cost of remaing paths.
- If Cost of current is lower than estimated of remaing choose current path.
- Repeate till goal is reached.

A* is a variation of Best First Search where evaluation of state depends on heuristic value and its distance from start state.

The value of f(n) is

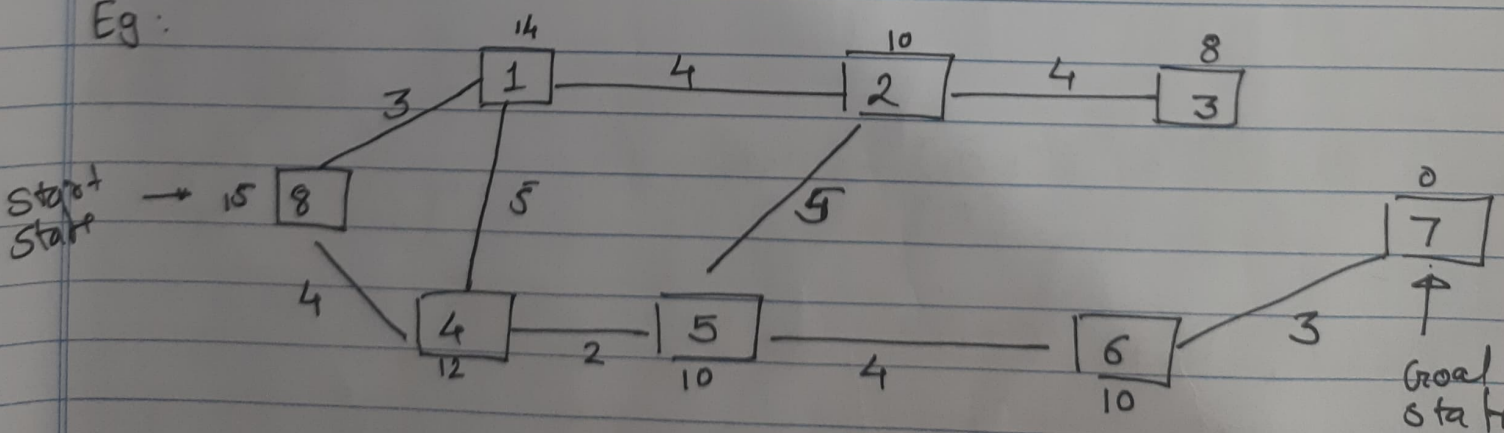
$$f(n) = g(n) + h(n)$$

where g(n) - Cost of heuristic estimation to reach the node n from root.

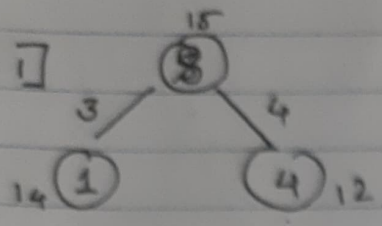
h(n) - Cost of cheapest path from n to goal

A* is Complete and Optimal. Its time complexity and Space Complexity is $O(b^m)$

Eg:



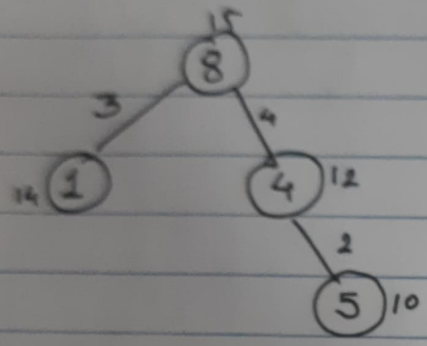
1]



Open : 4 (12+4) , 1 (14+3)
= 4 (16) , 1 (17)

Closed : 8 (15)

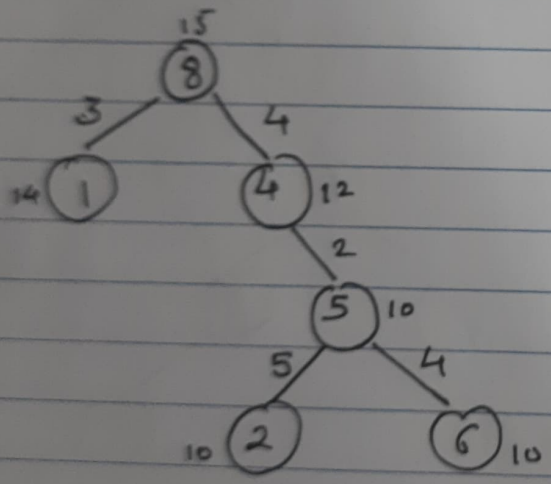
2]



Open : 5 (6+10) , 1 (14+3)
= 5 (16) , 1 (17)

Closed : 8 (15) , 4 (16)

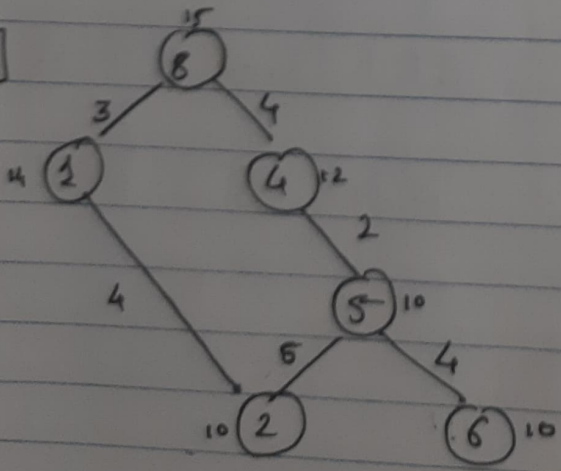
3]



Open : 1 (14+3) , 6 (10+10) , 2 (10+11)
: 1 (17) , 6 (20) , 2 (21)

Closed : 8 (15) , 4 (16) , 5 (16)

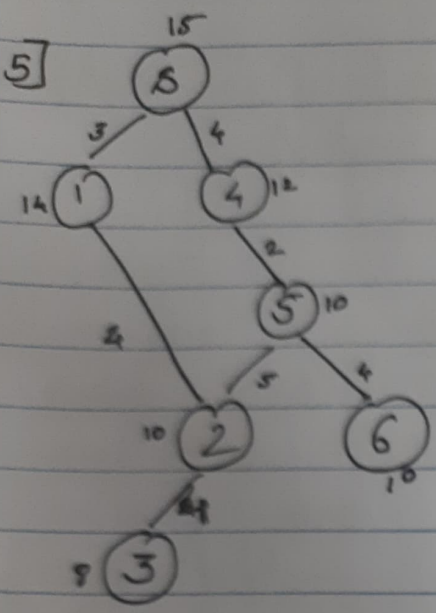
4]



Open : 2 (10+7) , 6 (10+10)
: 2 (17) , 6 (20)

Closed : 8 (15) , 4 (16) , 5 (16)
1, (17)

5]

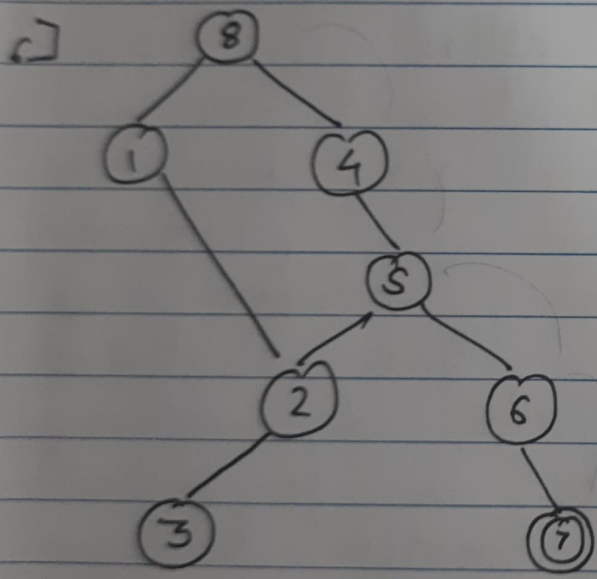


Open: 3 (8+11), 6 (10+10)

Closed: 8 (15), 4 (16), 5 (16)
1 (17), 2 (17)

Open: 6 (20)

Closed: 8 (15), 4 (16), 5 (16),
1 (17), 2 (17), 3 (19)



Open: 7 (0+13)

Closed: 8 (15), 4 (16), 5 (16)
1 (17), 2 (17), 3 (19),
6 (20)

Path Cost = 13

Path: 8 → 4 → 5 → 6 → 7

Experiment 4

AIM - Implement Greedy Best First Search / A* search algorithm in Python.

Code: Greedy Best First Search –

```
graph = {
    'S': [('A', 3), ('B', 2)],
    'A': [('C', 4), ('D', 1)],
    'B': [('E', 3), ('F', 1)],
    'E': [('H', 5)],
    'F': [('I', 2), ('G', 3)],
    'C': [('A', 4)],
    'D': [('A', 1)],
    'E': [('B', 3)],
    'I': [('F', 2)],
    'G': [('F', 3)]
}



dist = {}
h = {
    'S': 13, 'A': 12, 'B': 4, 'C': 7, 'D': 3, 'E': 8, 'F': 2, 'H': 4, 'I': 9, 'G': 0
}



def bestFit(start, target):
    q = []
    q.append((h[start], start))
    for x in graph:
        dist[x] = 10000
    dist[start] = 0

    while q:
```

```
        print()
        q = sorted(q)
        curr = q.pop(0)
        print(curr)
        curr_dist = dist[curr[1]]
        for node in graph[curr[1]]:
            if curr_dist+node[1] < dist[node[0]]:
                print(node, end=" ")
                dist[node[0]] = curr_dist + node[1]
                q.append((h[node[0]], node[0]))
    print(dist)


print("Niyati's Code for Greedy Best First Search")
root_node = input("Enter Root Node: ")
goal_node = input("Enter Goal Node: ")


bestFit(root_node, goal_node)
```

Output –

```
Niyati's Code for Greedy Best First Search
Enter Root Node: S
Enter Goal Node: G

(13, 'S')
('A', 3) ('B', 2)
(4, 'B')
('E', 3) ('F', 1)
(2, 'F')
('I', 2) ('G', 3)
(0, 'G')

(8, 'E')

(9, 'I')

(12, 'A')
('C', 4) ('D', 1)
(3, 'D')

(7, 'C')
{'S': 0, 'A': 3, 'B': 2, 'E': 5, 'F': 3, 'C': 7, 'D': 4, 'I': 5, 'G': 6}
PS C:\Engineering\3rd Year\Sem VI\PRACTICALS\AI>
```

Niyati_Savant_TE_C31_2103156

Code: A* Search –

```
import heapq

class Node:
    def __init__(self, name, cost=0, parent=None):
        self.name = name
        self.cost = cost
        self.parent = parent
        self.priority = 0

    def __lt__(self, other):
        return self.priority < other.priority


def heuristic(node, goal, heuristic_map):
    return heuristic_map.get(node.name, 0)


def a_star_search(start, goal, graph, heuristic_map):
    open_list = []
    heapq.heappush(open_list, (0, Node(start)))
    visited = set()

    while open_list:
        _, current_node = heapq.heappop(open_list)

        if current_node.name in visited:
            continue
        visited.add(current_node.name)

        if current_node.name == goal:
            path = []
```

```
        while current_node:
            path.append((current_node.name, current_node.priority))  # Include priority in the result
            current_node = current_node.parent
        return path[::-1]


    for neighbor, cost in graph[current_node.name].items():
        if neighbor not in visited:
            neighbor_node = Node(neighbor, current_node.cost + cost, current_node)
            neighbor_node.priority = neighbor_node.cost + heuristic(neighbor_node, goal,
heuristic_map)
            heapq.heappush(open_list, (neighbor_node.priority, neighbor_node))


  return None




# Graph data
graph = {
  'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
  'Bucharest': {'Urziceni': 85, 'Pitesti': 101, 'Giurgiu': 90, 'Fagaras': 211},
  'Craiova': {'Drobeta': 120, 'Rimnicu': 146, 'Pitesti': 138},
  'Drobeta': {'Mehadia': 75, 'Craiova': 120},
  'Eforie': {'Hirsova': 86},
  'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
  'Giurgiu': {'Bucharest': 90},
  'Hirsova': {'Urziceni': 98, 'Eforie': 86},
  'Iasi': {'Neamt': 87, 'Vaslui': 92},
  'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
  'Mehadia': {'Lugoj': 70, 'Drobeta': 75},
  'Neamt': {'Iasi': 87},
  'Oradea': {'Zerind': 71, 'Sibiu': 151},
  'Pitesti': {'Rimnicu': 97, 'Craiova': 138, 'Bucharest': 101},
  'Rimnicu': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},
```

```
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu': 80},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},
    'Vaslui': {'Iasi': 92, 'Urziceni': 142},
    'Zerind': {'Arad': 75, 'Oradea': 71}
}

# Heuristic values
heuristic_map = {
    'Arad': 366,
    'Bucharest': 0,
    'Craiova': 160,
    'Drobeta': 242,
    'Eforie': 161,
    'Fagaras': 176,
    'Giurgiu': 77,
    'Hirsova': 151,
    'Iasi': 226,
    'Lugoj': 244,
    'Mehadia': 241,
    'Neamt': 234,
    'Oradea': 380,
    'Pitesti': 100,
    'Rimnicu': 193,
    'Sibiu': 253,
    'Timisoara': 329,
    'Urziceni': 80,
    'Vaslui': 199,
    'Zerind': 374
}
```

```
cities = [
    "Arad",
    "Bucharest",
    "Craiova",
    "Dobreta",
    "Eforie",
    "Fagaras",
    "Giurgiu",
    "Hirsova",
    "Iasi",
    "Lugoj",
    "Mehadia",
    "Neamt",
    "Oradea",
    "Pitesti",
    "Rimnicu_Vilcea",
    "Sibiu",
    "Timisoara",
    "Urziceni",
    "Vaslui",
    "Zerind"
]

print("Niyati's Code for A* algorithm")
for i in range(len(cities)):
    print(f"{i}. {cities[i]}")
start_city = int(input("Enter no. for  Start City: "))
start_city = cities[start_city]

goal_city = int(input("Enter no. Goal City: "))
goal_city = cities[goal_city]
```

path = a_star_search(start_city, goal_city, graph, heuristic_map)

print(f"Path from {start_city} to {goal_city}:", path)

Output –

```
Niyati's Code for A* algorithm
0. Arad
1. Bucharest
2. Craiova
3. Dobreta
4. Eforie
5. Fagaras
6. Giurgiu
7. Hirsova
8. Iasi
9. Lugoj
10. Mehadia
11. Neamt
12. Oradea
13. Pitesti
14. Rimnicu_Vilcea
15. Sibiu
16. Timisoara
17. Urziceni
18. Vaslui
19. Zerind
Enter no. for  Start City: 0
Enter no. Goal City: 1
Path from Arad to Bucharest: [('Arad', 0), ('Sibiu', 393), ('Rimnicu', 413), ('Pitesti', 417), ('Bucharest', 418)]
PS C:\Engineering\3rd Year\Sem VI\PRACTICALS\AI>
```

Niyati Savant
TE - C31

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

2103156 Experiment 5

Aim - Implement Genetics/ Hill Climbing in Python

Theory -

Genetic Algorithms are adaptive heuristic search algorithms that belong to larger part of evolutionary algorithms. These are based on ideas of natural selection and genetics. These are intelligent exploitation of random searches provided with historical data to direct the search into the region of better performance in solution space. They are used to generate high-quality solutions for optimization problems and search problems.

Each generation consists of a population of individuals and each individuals of consecutive generations to solve a problem. Each generation consists of a population of individuals and each individual represents a point in search space and a possible solution.

Operators of Genetic Algorithms

Once initial generation is created the algorithm evolves the generation using -

1) Selection Operator - The idea is to give preference to individuals with good fitness scores and allow them to pass their genes to successive generations.
2) Crossover Operator - This represents mating between

Niyati Savant
TE - C3
2103156

They are selected using selection operator and crossover sites are randomly chosen and exchanged to create a new offspring.

3] Mutation Operator - The key idea is to insert random genes in offspring to maintain population diversity.

Advantages -

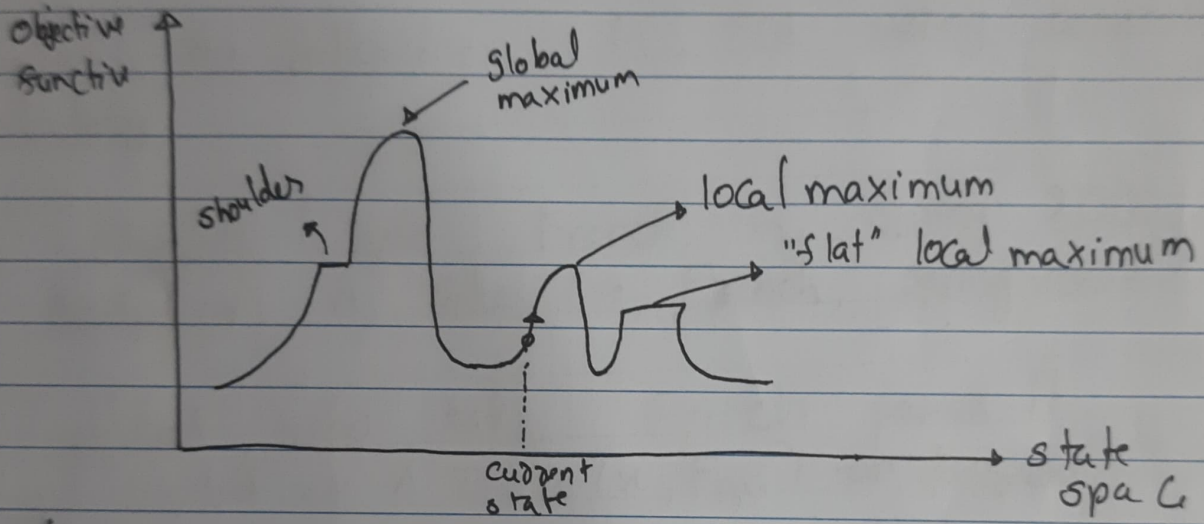
1) They are Robust
2) Provide optimisation over large space state
3) Unlike traditional AI, they do not break on slight change in input or noise.

Hill Climbing -

A simple optimization algorithm to find best possible solution for a given problem. It is a local search algorithm used in optimization problems and goal is find best solution from set of possible solutions. It starts with initial solution and then iteratively makes small changes to improve the solution. The changes are based on a heuristic function that evaluates solution quality. It makes these small changes until it reaches a local maximum. The variations including ~~stp~~ steepest ascent Hill Climbing, first-choice Hill Climbing, simulated annealing. It is used in a variety of optimization problems, such as scheduling, route planning, and resource allocation.

Nryati Savant
TE - C31
2103156

Advantages -
- Simple and intuitive algorithm
- Used in wide variety of optimization problems
- Efficient to find local optima.
- Easily modified and extended



Different regions in State Space Diagram:
- Local maximum - state better than its neighboring state.
- Global maximum - Best possible state in space diagram
- Plateau - flat region of state space with same value neighboring states.
- Ridge - region higher than neighbor but itself has a slope.
- Current State - region where we are currently present
- Shoulder - Plateau that has an uphill edge.

Experiment 5

AIM - Implement Genetics / Hill Climbing in Python.

Code – Genetics

```
import numpy as np

population_size = 6
gene_length = 4
num_generations = 100
crossover_rate = 0.5
mutation_rate = 0.1
num_parents = int(population_size / 2)

print("Niyati's Code for Genetics algorithm")

def initialize_population(size, gene_length):
    return np.random.randint(0, 31, (size, gene_length))

def calculate_fitness(chromosome):
    objective_value = abs(sum([chromosome[i] * (i+1) for i in range(len(chromosome))]) - 30)
    fitness_value = 1 / (1 + objective_value)  # Modified to match the provided logic
    return fitness_value

def select_parents(population, fitness, num_parents):
    fitness_sum = np.sum(fitness)
    probability = fitness / fitness_sum
    chosen = set()  # Set to keep track of which individuals have been chosen

    parents = np.empty((num_parents, population.shape[1]))
    for parent_num in range(num_parents):
        rand = np.random.rand()
        cumulative_probability = 0.0
        for i in range(len(probability)):
```

```
            if i not in chosen:
                cumulative_probability += probability[i]
                if rand <= cumulative_probability:
                    parents[parent_num, :] = population[i, :]
                    chosen.add(i)  # Mark this individual as chosen
                    break
    return parents


def crossover(parents, offspring_size, crossover_rate):
    offspring = np.empty(offspring_size)
    for k in range(offspring_size[0]):
        if np.random.rand() < crossover_rate:
            parent1_idx = k % parents.shape[0]
            parent2_idx = (k+1) % parents.shape[0]
            crossover_point = np.random.randint(1, offspring_size[1])
            offspring[k, 0:crossover_point] = parents[parent1_idx, 0:crossover_point]
            offspring[k, crossover_point:] = parents[parent2_idx, crossover_point:]
    return offspring

def mutate(offspring_crossover, mutation_rate):
    for idx in range(offspring_crossover.shape[0]):
        for gene in range(offspring_crossover.shape[1]):
            if np.random.rand() < mutation_rate:
                offspring_crossover[idx, gene] = np.random.randint(0, 31)
    return offspring_crossover

population = initialize_population(population_size, gene_length)
print("Initial Population:\n", population)

for generation in range(num_generations):
    fitness = np.array([calculate_fitness(individual) for individual in population])
```

```
    print(f"\nGeneration {generation} Fitness:\n", fitness)
    parents = select_parents(population, fitness, num_parents)
    print("Selected Parents:\n", parents)
    offspring_crossover = crossover(parents, (population_size - num_parents, gene_length),
crossover_rate)
    print("Crossover Offspring:\n", offspring_crossover)
    offspring_mutation = mutate(offspring_crossover, mutation_rate)
    print("Mutated Offspring:\n", offspring_mutation)
    population[:num_parents, :] = parents
    population[num_parents:, :] = offspring_mutation
    break
def genetic_algorithm(population, population_size, gene_length, num_generations, crossover_rate,
mutation_rate, num_parents):


    for generation in range(num_generations):
        fitness = np.array([calculate_fitness(individual) for individual in population])
        parents = select_parents(population, fitness, num_parents)
        offspring_crossover = crossover(parents, (population_size - num_parents, gene_length),
crossover_rate)
        offspring_mutation = mutate(offspring_crossover, mutation_rate)


        population[:num_parents, :] = parents
        population[num_parents:, :] = offspring_mutation


    final_fitness = np.array([calculate_fitness(individual) for individual in population])
    best_index = np.argmax(final_fitness)  # Use argmax because we are using inverted fitness values
    best_solution = population[best_index]


    print("\nFinal Best Solution:\n", best_solution)
    print("With Fitness Score:", final_fitness[best_index])



genetic_algorithm(population, population_size, gene_length, num_generations, crossover_rate,
mutation_rate,num_parents)
```

Output –

```
Niyati's Code for Genetics algorithm
Initial Population:
 [[28 14 25 10]
 [16  5 17 15]
 [ 4  9 10 24]
 [29 20 27  1]
 [16 15  3  0]
 [ 7  1  0 23]]

Generation 0 Fitness:
 [0.00704225 0.00925926 0.00840336 0.008      0.03846154 0.01388889]
Selected Parents:
 [[2.90000000e+001 2.00000000e+001 2.70000000e+001 1.00000000e+000]
 [1.60000000e+001 1.50000000e+001 3.00000000e+000 0.00000000e+000]
 [3.56043054e-307 4.45037520e-307 3.11521375e-307 2.78145267e-307]]
Crossover Offspring:
 [[2.90000000e+001 2.00000000e+001 2.70000000e+001 1.00000000e+000]
 [1.60000000e+001 1.50000000e+001 3.00000000e+000 0.00000000e+000]
 [3.56043054e-307 2.00000000e+001 2.70000000e+001 1.00000000e+000]]
Mutated Offspring:
 [[2.90000000e+001 2.00000000e+001 2.70000000e+001 1.00000000e+000]
 [1.60000000e+001 1.50000000e+001 3.00000000e+000 0.00000000e+000]
 [3.56043054e-307 2.00000000e+001 2.70000000e+001 1.00000000e+000]]

Final Best Solution:
 [ 8 11  0  0]
With Fitness Score: 1.0
PS C:\Engineering\3rd Year\Sem VI\PRACTICALS\AI>
```

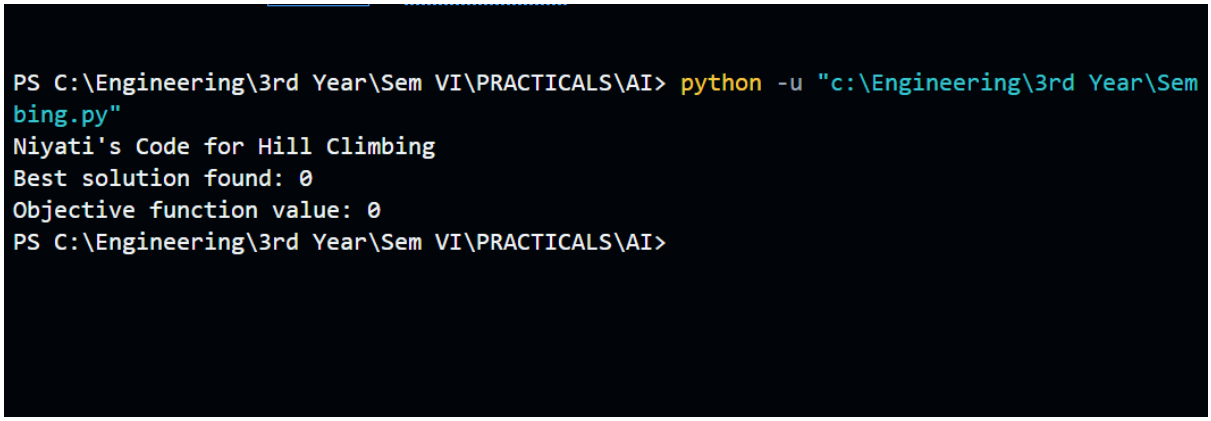
Niyati_Savant_TE_C31_2103156

Code – Hill Climbing

```
import random

# Objective function to be maximized
def objective_function(x):
    return -x ** 2

# Generate initial solution randomly
def generate_initial_solution():
    return random.randint(-100, 100)

# Generate neighbour solutions
def generate_neighbours(solution):
    neighbours = []
    for delta in [-1, 1]:
        neighbours.append(solution + delta)
    return neighbours

# Get highest quality neighbour of current solution
def get_best_neighbour(neighbours):
    best_neighbour = neighbours[0]
    best_quality = objective_function(best_neighbour)
    for neighbour in neighbours[1:]:
        neighbour_quality = objective_function(neighbour)
        if neighbour_quality > best_quality:
            best_quality = neighbour_quality
            best_neighbour = neighbour
    return best_neighbour

# Hill climbing algorithm
def hill_climbing():
    current_solution = generate_initial_solution()
```

```
    # print("Initial Solution: ", current_solution)
    while True:
        neighbours = generate_neighbours(current_solution)
        best_neighbour = get_best_neighbour(neighbours)
        if objective_function(best_neighbour) <= objective_function(current_solution):
            return current_solution
        current_solution = best_neighbour



best_solution = hill_climbing()
print("Niyati's Code for Hill Climbing")
print("Best solution found:", best_solution)
print("Objective function value:", objective_function(best_solution))
```

Output –



THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 6

Aim - Knowledge representation and creating a knowledge base for Wumpus world

Theory -

Q1] Represent the following in FOL using consistent vocabulary

a. Some students took French in Spring 2001.
→ $\exists x$ Student(x) $\wedge$ Takes(x, French, spring 2001).

b. Every student who takes French passes it.
→ $\forall x, s$ Student(x) $\wedge$ Takes(x, French, s) $\Rightarrow$ Passes(x, F, s)

c. Only one student took Greek in Spring 2001.
→ $\exists x$ Student(x) $\wedge$ Takes(x, Greek, spring2001) $\wedge$ $\forall y$ $y \neq x$ $\rightarrow$ $\sim$ Takes(y, Greek, Spring 2001).

d. The best score in Greek is always higher than best score in French.
→ $\forall s$ $\exists x$ $\forall y$ Score(x, Greek, s) > Score(y, French,

e. Every person who buys a policy is smart.
→ $\forall x$ Person(x) $\wedge$ [$\exists y, z$ Policy(y) $\wedge$ Buys(x, y, z)] → Smart(x)

f. No person buys an expensive policy.
→ $\forall x, y, z$ Person(x) $\wedge$ Policy(y) $\wedge$ Expensive(y) → $\sim$ Buy(x, y, z)

There is an agent who sells policies only to people who are not insured.

$\exists x$ Agent $(x) \wedge \forall y, z$ Policy $(y) \wedge$ Sells (x, y, z) $\rightarrow$ [Person $(z) \wedge \sim$ Insured (z)]

There is a barber who shaves all men who do not shave themselves

$\exists x$ Barber $(x) \wedge \forall y$ Man $(y) \wedge \sim$Shave (y, y) $\rightarrow$ Shave (x, y).

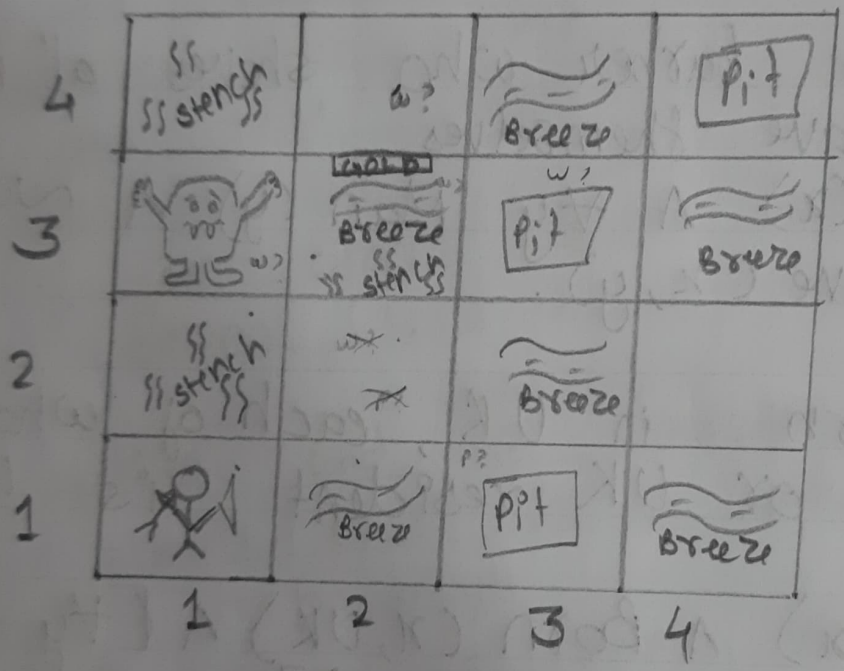
A person born in UK, each of whose parents is a UK citizen or UK resident, is UK citizen by birth.

$\forall x$ Person $(x) \wedge$ Born $(x, UK) \wedge$ [$\forall y$ Parent (y, x) $\rightarrow$ [[$\exists r$ Citizen (y, UK, r)] $\vee$ Resident (y, UK)]] $\rightarrow$ Citizen $(x, UK, Birth)$.

A person born outside UK, one of whose parents is a UK citizen by birth, is UK citizen by descent.

$\forall x$ Person $(x) \wedge \sim$Born $(x, UK) \wedge$ [$\exists y$ Parent (y, x) $\wedge$ Citizen $(y, UK, Birth)$] $\rightarrow$ Citizen $(x, UK, DesCnf)$

Describe the Wumpus world problem in detail and create a knowledge base for the Wumpus world and prove that Wumpus is present in [1, 3].


4
3
2
1
1
2
3
4
Stench
Breeze
Pit
Stench
Breeze
Pit
Breeze
Stench
Breeze
Breeze
Pit
Breeze

→ Wumpus is an early computer game, a.k.a "Hunt the Wumpus" and was developed by Gregory Yob. originally written in BASIC (Beginner's All purpose Symbolic Instruction Code). It is a map-based game.

It is like a cave that represents rooms that are connected by passage ways. Wumpus is a monster who lives in one ~~end~~ of the room and eats player. In our 4×4 grid, Wumpus is at (3,1).
Player can start from any position (We start from (1,1)).

The sprite and its features are-
1) Few rooms [(1,3), (3,3) and (4,4)] have bottomless pits to trap player.
2] Breeze is in a room with pits in neighborhood room. [(1,2), (1,4), (2,3), (3,2), (3,4) and (4,3)]
3] Stench in room with WUMPUS in neighbor room [(2,1), (3,2) and (4,1)].
4] Agent has arrow to shoot in straight line to kill WUMPUS
5] The room with gold (3,2) glitters.

Apart ~~#~~ from these, 2 percepts - bump [agent walks into wall) and scream (created if wumpus is killed) can be accepted.

An agent receives percepts while exploring cave rooms which can be represented as 5 element list [stench, breeze, glitter, bump, scream]
The action agent can perform.
- Move - Move forward
- Turn - Turn right or left by 90°
- Grab - Pick gold
- Shoot - shoot arrow

Actions are repeted till WUMPUS is killed or player is killed

Exploring WUMPUS world environment.

1] Initially agent is at (1, 1). The first percept is [n, n, n, n, n]. {n - none}. Thus, agent can move to (1, 2) or (2, 1).

2] Lets move to (1, 2). As (1, 1) is visited it is marked with V. The player gets the percept: [n, breeze, n, n, n]. Therefore (1, 2) is marked with 'B' and rooms (1, 3) (2, 2) are mark with "P?" as one or both may have bit and are unsafe. Therefore we move back to (1, 1).

3] We move to (2, 1) where percept is [stench, n, n, n, n] which means either (2, 2) or (3, 1) has WUMPUS. No breeze percept indicate (2, 2) cannot have pit and from step 2

we understand (2,2) cannot have ~~ppt~~ Wumpus as (1,2) showed no stench. Thus (2,2) is safe.

4) Agent received (n, n, n, n, n) at (2,2). Thus (2,3) and (3,2) are safe

5) Move to [3,2]. Agent gets (stench, breeze, glitter, n, n]
Field 1 shows - (3,1), (3,3), (4,2) can have Wumpus.
Field 2 shows - (3,1) (33) (2,2) (4 2) can have pit
Field 3 shows (3,2) has gold.
Agent grabs gold.

6] As (3,2) has stench, Wumpus may be (3,1), (3,3) or (4,2). Thus only safe is coming back to (2,1). As this has stench and we have already proved (2,2) does not have Wumpus we can conclude (3,1) has Wumpus. Agent can shoot.

Atomic proposition variables -
i,j indicate room [i, j]

1) P_{ij} - True if there is ppt.
2) B_{ij} - True if there is breeze
3) W_{ij} - True if there is Wumpus
4) S_{ij} - True if stench
5) V_{ij} - True if visited
6) G_{ij} - True if glitter & thus gold
7) OK_{ij} - True is safe.

Apply MP rule with R1 i.e
$\sim S_{11} \rightarrow \sim W_{11} \wedge \sim W_{21} \wedge \sim W_{12}$ and $\sim S_{11}$
gives output $\sim W_{11} \wedge \sim W_{12} \wedge \sim W_{12}$

- Apply And-Elimination rule to $\sim W_{11} \wedge \sim W_{12} \wedge \sim W_{21}$ we get:
$\sim W_{11}$, $\sim W_{12}$, $\sim W_{21}$

- Apply modus ponens to $\sim S_{12}$ and R_2 ~~to get~~ i.e $\sim S_{12} \rightarrow \sim W_{12} \wedge \sim W_{22} \wedge \sim W_{13}$ to get
$\sim W_{12} \wedge \sim W_{22} \wedge \sim W_{13}$

Apply And-Elimination to get $\sim W_{12}$, $\sim W_{22}$, $\sim W_{13}$

- Apply modus ponens to S_{12} and R_4 i.e $S_{21} \rightarrow W_{31} \vee W_{21} \vee W_{22} \vee W_{11}$ to get $W_{31} \vee W_{21} \vee W_{22} \vee W_{11}$

- Apply unit resolution on above and $\sim W_{11}$ to get
$W_{31} \vee W_{21} \vee W_{22}$

- Apply unit resolution on $W_{31} \vee W_{21} \vee W_{22}$ ~~$\vee \sim W_{22}$~~ and $\sim W_{22}$ to get $W_{31} \vee W_{21} \vee$ ~~W_{22}~~.

- Apply unit resolution on $W_{31} \vee W_{21}$ and $\sim W_{21}$ to get W_{31}.

Hence proved it is in ~~room~~ (3,1)

g 28/03/24

(A)

Experiment 7

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Aim - Planning for Blocks World problem

Theory -

- Describe blocks World ~~prob~~ planing problem and discuss how it can be solved using Classical planning method.

→ ~~The principal drawback of the~~

The blocks world problm is as follows. There is a table on which some blocks are placed. Some blocks may or may not be stacked on other blocks. We have a robot arm to pick up or put down the blocks. The arm can move only one block at a time, and no other block must be stacked on top of the block to be moved. Our aim is to change configuration of the blocks from initial state to goal state, both of which are specified

Predicates are statements that helps us convey the information about a configuration in blocks world.
List of predicates -

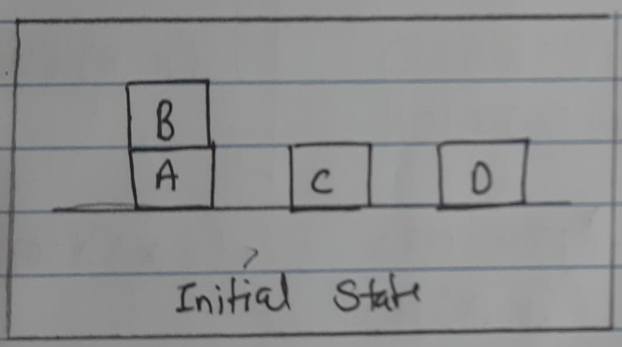
1. ON (A,B) : Block A is on B
2. ONTABLE (A) : A is on table
3. CLEAR (A) : Nothing is on top of A
4. HOLDING (A) : Arm is holding A
5. ARMEMPTY : Arm is holding nothing

The predicates are used to represent Initial and goal state.

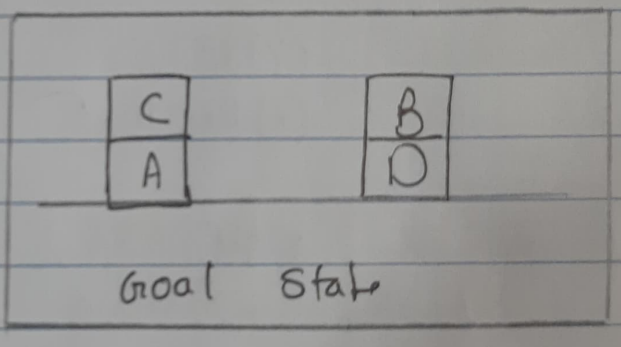
The operations that the arm can perform-

1. STACK(X,Y): Stacking X on Y
2. UNSTACK(X,Y): Picking X which is on top of Y
3. PICKUP(X): Pick X which is on top of table
4. PUTDOWN(X): Put X on table

The operations have certain preconditions which must be satisfied to perform the same and are represented as predicates.



Initial State



Goal State

~~In this~~ This environment is fully observable, finite, deterministic, static and discrete and can be solved using classical planning

Using the predicates, we can describe the states as

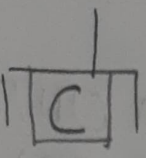
Initial State - ON(B,A) ∧ ONTABLE(A) ∧ ONTABLE(C) ∧ ONTABLE(D) ∧ CLEAR(B) ∧ CLEAR(D) ∧ CLEAR(C) ∧ ARMEMPTY

Goal State - ON(C,A) ∧ ON(B,D) ∧ ONTABLE(A) ∧ ONTABLE(D) ∧ CLEAR(B) ∧ CLEAR(C) ∧ ARMEMPTY

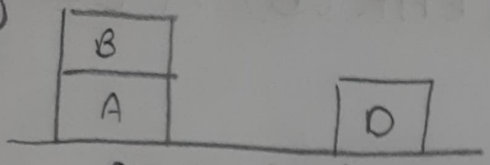
The effect of the operations is shown with 2 lists ADD and DELETE. The ADD list contains predicate that will become true once operation is performed. The DELETE list contains predicates which will cease to be true once operation is performed.

Operator	Precondition	Delete	ADD
STACK(x, y)	CLEAR(y) ∧ HOLDING(x)	CLEAR(y) HOLDING(x)	ARMEMPTY ON(x, y)
UNSTACK(x, y)	ARMEMPTY ∧ ON(x, y) ∧ CLEAR(x)	ARMEMPTY ∧ ON(x, y)	HOLDING(x) ∧ CLEAR(y)
PICKUP(x)	CLEAR(x) ∧ ONTABLE(x) ∧ ARMEMPTY	ONTABLE(x) ∧ ARMEMPTY	HOLDING(x)
PUTDOWN(x)	HOLDING(x)	HOLDING(x)	ONTABLE(x) ∧ ARMEMPTY

The steps to create goal state from start state are -

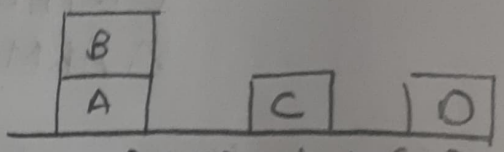


(1)



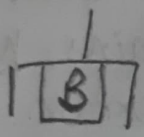
PICKUP(C)

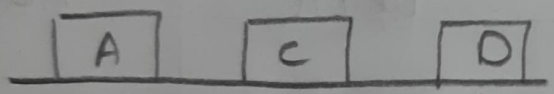
(2)



PUTDOWN (C)

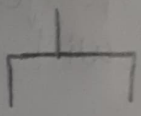
(3)

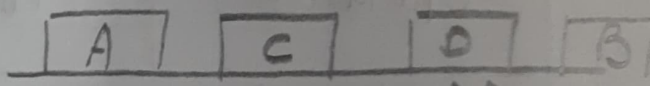




UNSTACK (B, A)

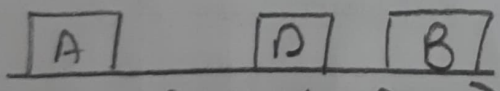
(4)





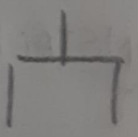
PUTDOWN(B)

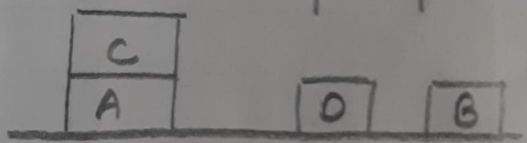
(5)



PICKUP(C)

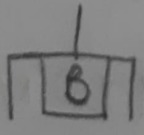
(6)

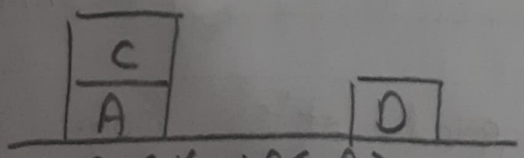




STACK (C, A)

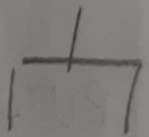
(7)





PICK UP(B)

(8)



STACK(B, D)

Experiment 8

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Aim - Implementing Family tree using prolog

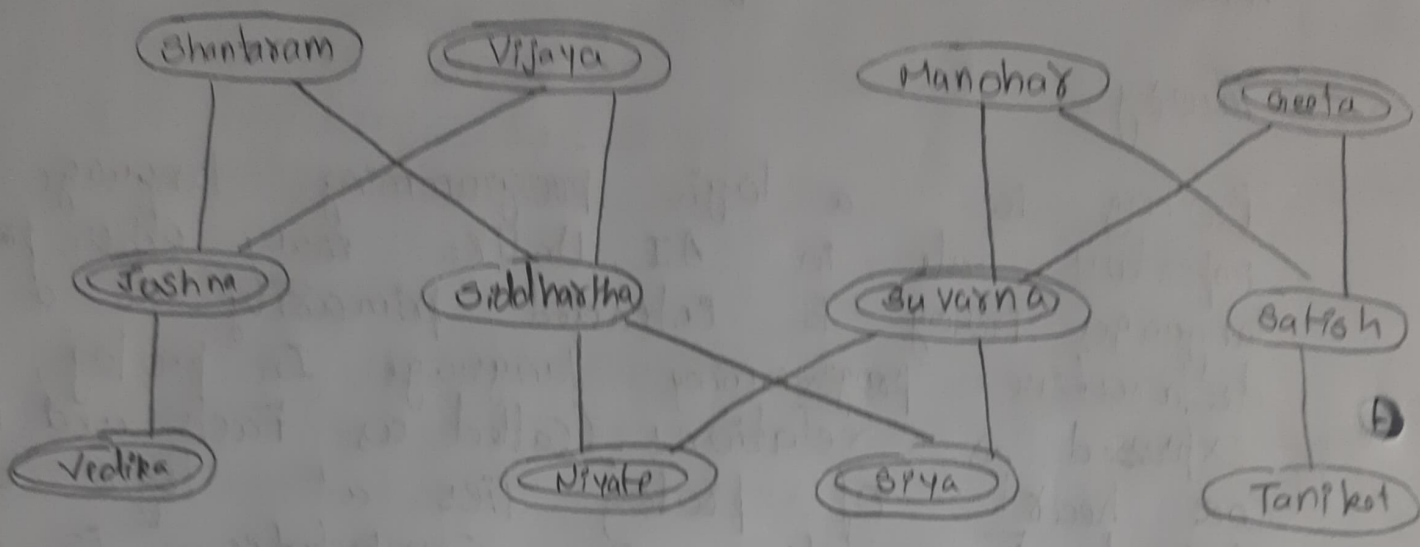
Theory -

Prolog is a logic programming language. It has important role in AI. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at logic being applied. Formulation or Computation is carried out by running a query over these relations.

In prolog, we declare some facts which constitute the Knowledge Base of the system. We can query against the knowledge base. We get an affirmative output if query is in base or implied by it, else output is negative. Knowledge base is similar to database, against which we can query.

Prolog facts are expressed in definite pattern. They contain entities and their relation and are written within parenthesis separated by comma (,). Their relation is expressed at start and outside parentheses. Each fact/rule ends with a dot (.).

Syntax : relation (entity 1, entity 2, ... k^{th} entity)



Savant Family Tree

indicates female

indicates male

Key Features:
1) Unification - The basic idea is, Can the given terms be made to represent the same structure.
2) Backtracking - To satisfy previous task in case of failure
3) Recursion - Basis for any search in program

Advantages -
1) Easy to build database. Less programming effort
2) Easy pattern matching. Recursion based search
3) In built List handling

Disadvantages:
1) LISP dominates w.r.t I/O features
2) Sometime input & output is not easy.

It is still used highly in AI for pattern matching over NLP parse trees.

Experiment 8

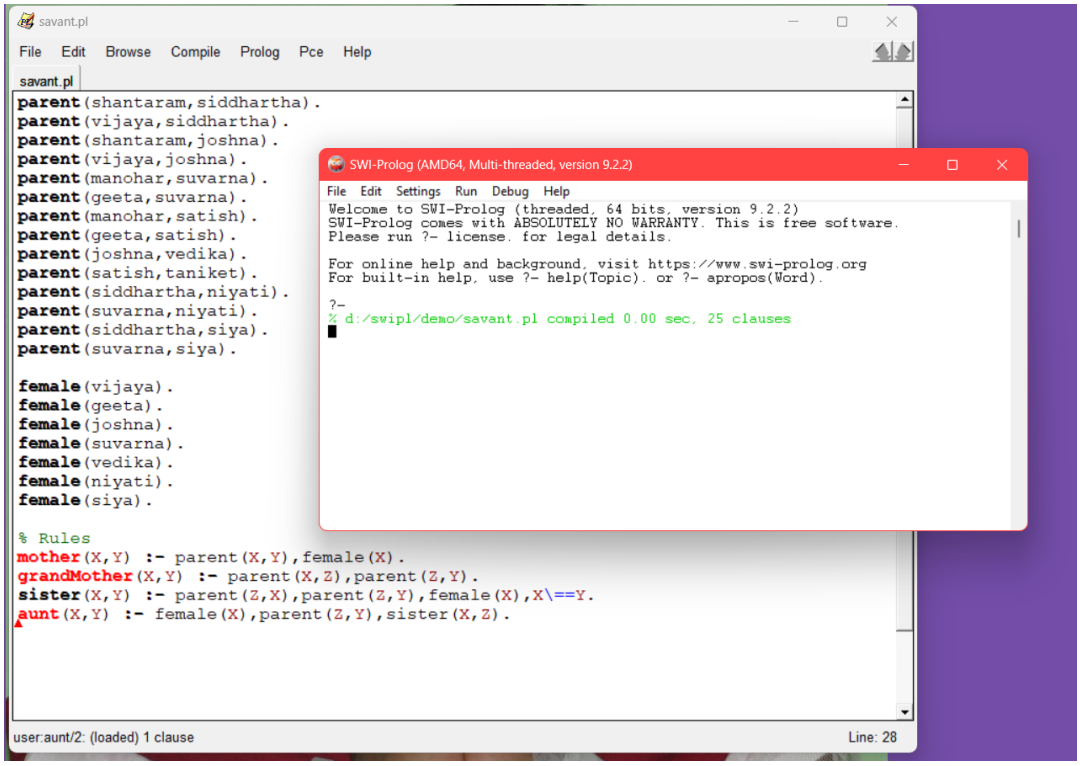
AIM - Implementing Family tree using prolog.

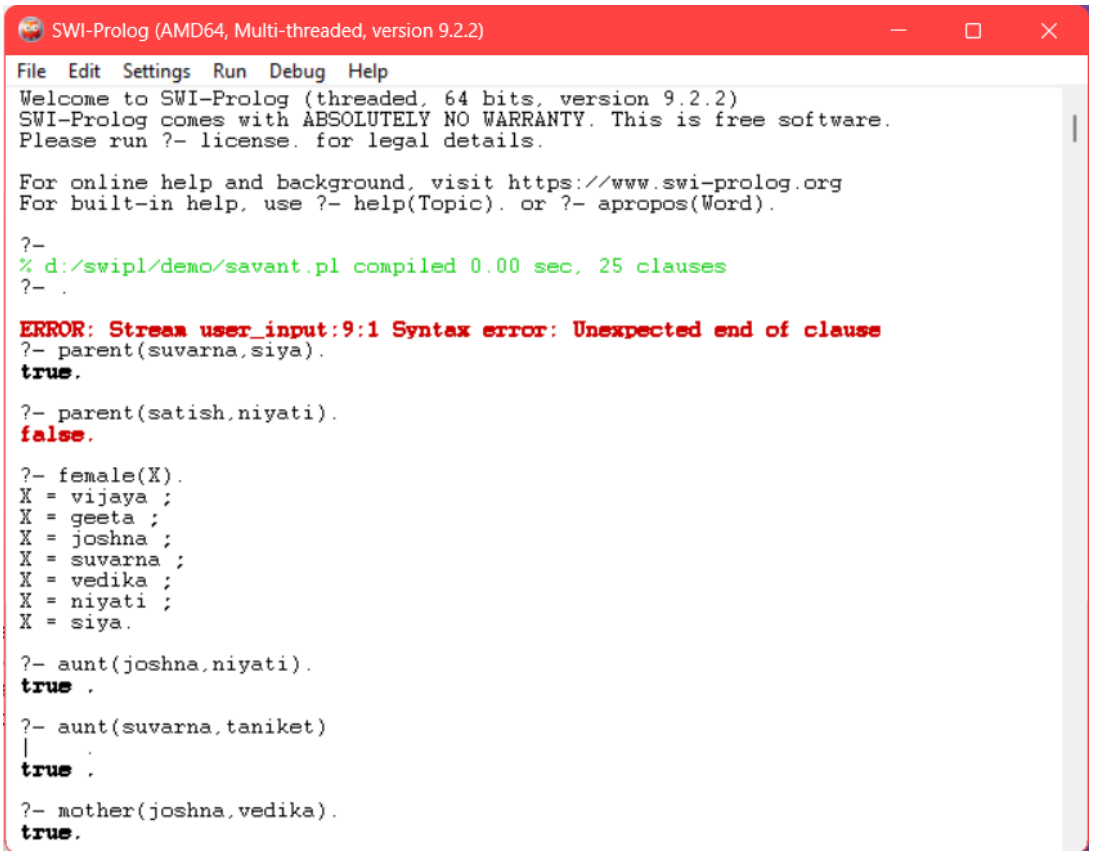
Knowledge Base –

```
parent(shantaram,siddhartha).
parent(vijaya,siddhartha).
parent(shantaram,joshna).
parent(vijaya,joshna).
parent(manohar,suvarna).
parent(geeta,suvarna).
parent(manohar,satish).
parent(geeta,satish).
parent(joshna,vedika).
parent(satish,taniket).
parent(siddhartha,niyati).
parent(suvarna,niyati).
parent(siddhartha,siya).
parent(suvarna,siya).

female(vijaya).
female(geeta).
female(joshna).
female(suvarna).
female(vedika).
female(niyati).
female(siya).

mother(X,Y) :- parent(X,Y),female(X).
grandMother(X,Y) :- parent(X,Z),parent(Z,Y).
sister(X,Y) :- parent(Z,X),parent(Z,Y),female(X),X\==Y .
aunt(X,Y) :- female(X),parent(Z,Y),sister(X,Z).
```

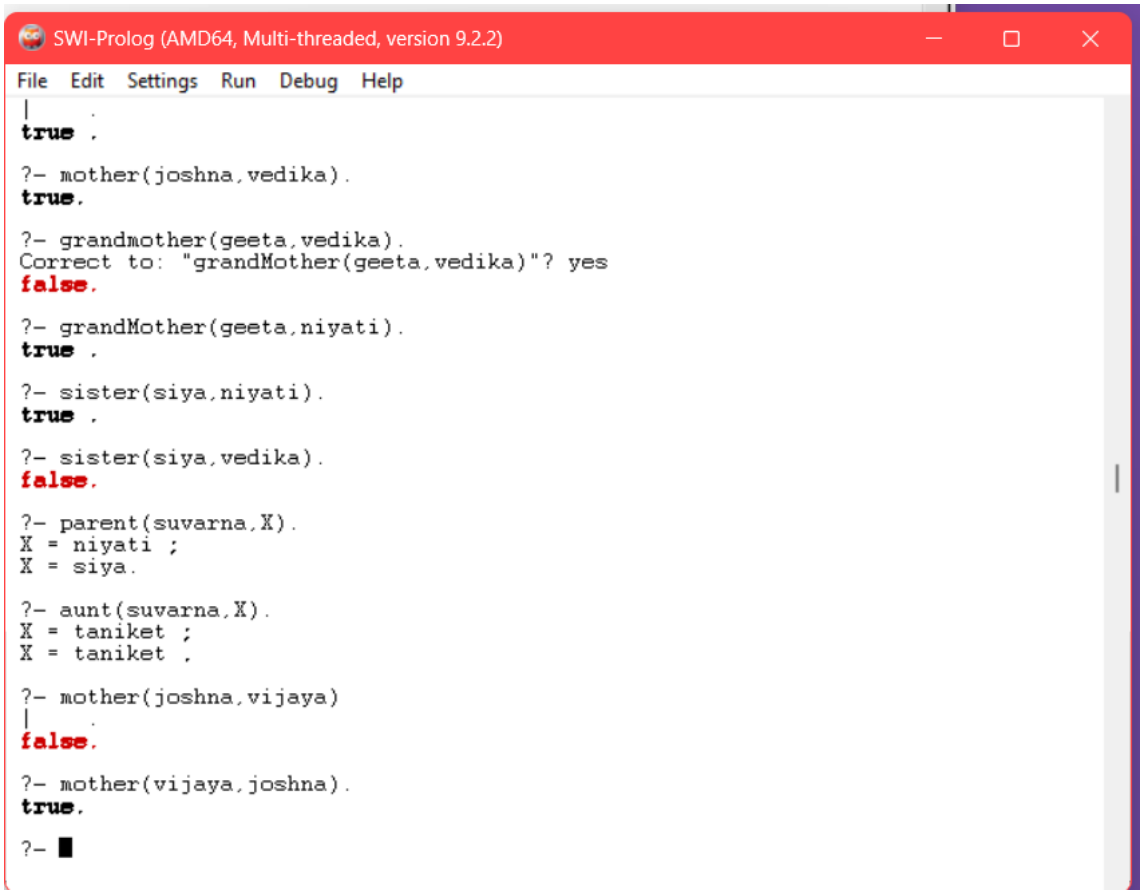
Niyati_Savant_TE_C31_2103156

Screenshots –





Niyati_Savant_TE_C31_2103156



Niyati Savant
2103156
C31

Written Assignment 01

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

25|01|2024

Q1 Give one definition on AI for -

i) Acting Humanly

→ According to Kurzweil, AI, as systems that act like humans can be defined as "The art of creatin machines that perform functions that require intelligence when performed by people."
The definition given by Rich & Knight, is "The study of how to make computers do thing at which, at the moment, people are better."
The Turing Test proposed by Alan Turing propos a satisfactory operational definition of Intelligence The computer passes the test if a human interrogator, after posing some written questions, cannot tell if the written responses come from a person not. For this computer needs NLP, knowledge representation, automated reasoning, ML

ii) Thinking Humanly

→ "The exciting new effort to make computers think machines with minds, in full and literal sense". - Haugeland
"[The automation of] activities that we associa with human thinking, like decision-making, prob solving, learning ..." (Bellman, 1978)
We need a sufficiently precise theory of mi

to express it as a computer program. The field of cognitive science brings together computer models from AI and experimental techniques from psychology to construct testable theories of working of humane mind.

iii) Acting Rationally

→ "Computational Intelligence is the study of design of intelligent agents." - Poole et al.
"AI ... is concerned with intelligent behavior in artifacts."
In this approach, emphasis was on correct inferances as one way to act rationally is to reason logically to the conclusion that a given action will achive one's goals and to act on that conclusion

iv) "Thinking Rationally"

→ "The study of mental faculties through the use of computational models." - Charniak & McDermott
"The study of computations that make it impossible to percive, reason and act."
Aristotle's syllogisms provided patterns for argument structures that always yielded correct conclusions when given right premises initiating field of logic. This logistic tradition with AI hopes

to build on such programs to create intelligent systems.

Q2] Explain the components of AI system in detail.

→ The components of AI are-

1) Perception
In order to work in the environment, intelligent agents need to scan the environment and the various objects in it. Agents scan environment with sense organs like camera, temp sensor. After capturing various scenes, perceiver analyses the different objects in it and extracts their features and relationships

2) Knowledge representation
As information obtained from environment through sensors may not be in format required by the system and so must be standerdized for further processing like learning various patterns, deducing inference, comparing with past objects. Some techniques are Propositional logic and first order logic

3) Learning
The simplest form of learning is unsupervised ie trial and error where program remembers action that gave desired output and discards others. In case of rote learning, the program simply

remembers the problem solution pairs. In other case, solution is given as input to system and program needs to generate solutions for new problems.

4) Reasoning

Logic of generating inferences from given set of facts. It is carried out based on strict rules of validity to perform a specified task. Reasoning can be deductive i.e truth of premises guarantees truth of conclusion or it can be inductive where truth of premises supports conclusion but it cannot be fully dependent on premises

5) Problem-solving

As per type of problem, there is a variety of problem solving strategies. These methods are mainly divided into general purpose methods - applicable to wide range of problems and special purpose which are customized

6) Natural Language Processing

NLP, involves machines or robot to understand and process human language, infer knowledge. It also involves active participation from machine

Q3] write note on categorization of AI

→ AI can be primarily categorized as per capabilities and functionality

A] Based on capabilities

i) Weak or Narrow AI.

The most common and currently available AI which can perform a dedicated task and nothing beyond its field or limitations. It can fail in unpredictable ways if it goes beyond limits. Some examples include Siri, Watson supercomputer, product recommandation systems, self-driving cars, speech and image recognition

ii) General AI-

General AI is a type of intelligence which could perform any intellectual task with human-like efficiency. The idea is to make a system which could be smarter and think like human by its own. Currently, there is no such system that comes under this and these systems are still under research

iii) Super AI.

It is a level of Intelligence of systems at which machines could surpass human intelligence and can perform any task better than human. It is an outcomen of general AI. Some key characteristics include capability to think, reason, solve the problem, make judgments, plan and communicate. It is hypothetical concept.

B] Based on functionality

i) Reactive machines

The most basic types of AI, that do not store memories or past experiences for future actions but only focus on current scenarios and react on it as per possible best action. Examples include IBM's Deep Blue and Google's AlphaGo.

ii) Limited Memory

The machines that can store past experiences or some data for short time period only. Self-driving cars are the best examples as they store recent speed of nearby cars, distance of other cars, speed limit and other information for navigation

iii) Theory of mind

These AI should understand human emotions, beliefs and be able to interact socially like humans. Researchers are making lots of efforts and improvements in these areas.

iv) Self awareness

The future of AI, the machines that are super intelligent and have their own consciousness, sentiments, and self-awareness. These are hypothetical

Q4 Explain problem formulation with examples

→ Given a goal to achieve, problem formulation is the process of deciding what states to be considered & what actions to take to achieve the goal. Problem can be defined formally using 5 components

1) Initial state
The state space is set of all states reachable from initial by executing any sequence of actions. Initial state is the one in which agent starts in
2) Actions.
Set of actions that can be executed or applicable in all possible states. A description of what each action does. aka transition model
3) Successor function
Function that returns a state on executing an action on the current state
4) Goal test
A test to determine whether current state is a goal state. In explicit goal test, test can be carried out just by comparing current state with defined goal state. In implicit test, state cannot be defined explicitly but must be generated by carrying out some computations.
5) Path Cost
The cost associated with each step to be taken to reach goal state and determine cost to reach each state done by cost function. The solution of least cost is optimal

Eg - 8 puzzle problem
It consists of a 3×3 board with tiles having 1 to 8 numbers and a blank tile that can be moved and aim is to ~~arrange~~ them in goal state

1 2 3	1 2 3
4 8 –	4 5 6
7 6 5	7 8 –
Initial state	Goal state

Problem formulation -

- States - Represented by 3×3 matrix structure and '0' denotes blank

1) Initial state : {{1,2,3},{4,8,0},{7,6,5}}
2) Actions : The blank space can move left, right, up, down
3) Successor function : On applying 'down' to start state 5 and blank switch places
4) Goal test : {{1,2,3},{4,5,6},{7,8,0}}
5) Path cost - Number of steps to reach final state

- Solution :

{{1,2,3},{4,8,0},{7,6,5}}
→ {{1,2,3},{4,8,5},{7,6,0}}
→ {{1,2,3},{4,8,5},{7,0,6}}
→ {{1,2,3},{4,0,5},{7,8,6}}
→ {{1,2,3},{4,5,0},{7,8,6}}
→ {{1,2,3},{4,5,6},{7,8,0}}

Path cost = 5 steps

Q5] Explain PEAS properties for Robotic vaccum cleaner

→ PEAS stands for Performance Measure, Environment Actuators, and sensors and is used for performance issues grouped under Task Environment.

- P - Performance ~~Measure~~ Measure
  The objective function to judge performance of the agent For vaccum cleaner it can be the ability to effectively remove dirt and debris from floors and carpets, being easy to use and maintain
- E - Environment
  The real environment where agent need to take deliberate actions. The interior spaces like homes, offices, large halls, as well as different types of flooring (wooden, tiled, granite) and surface (smooth, rough)
- A - Actuators
  The tools, equipment or organs using which agent performs actions in environment. Works as output The motor, fan, brush water compartment, ~~dust~~ dust collectors that generate ~~suction~~ suction, agitate dirt.
- S - Sensors
  These are tools, equipment or organs using which agent captures the state of the environment. Works as input. Like Cameras to detected dirt, joint angle sensors to scan position

Q6] Describe different types of environments with examples.

→ 1) Fully observable Vs Partially observable
When an agent sensor is capable to sense or access the complete state of an agent at each point in time, it is said to be fully observable environment and has no need to keep track of surrounding history. An unobservable environment is the one when the agent has no sensors in all environments.
Eg- Chess, as the board and opponent moves are fully observables.

2) Single-agent vs Multi-agent
An environment consisting of one agent only is a single agent. An environment involving more than one agent is multi-agent environment. which can be further classified as Co-operative and Competitive. A person left alone in a maze is a single-agent and a car driving agent is a multi-agent environment.

3) Deterministic vs Stochastic
When the next state of environment can be completely determined by previous state and the action executed by the agent like vacuum cleaner is deterministic while an idle environment with no change in the state is static like an empty house

4) Episodic vs Sequential

Episodic environment is the one where each of agent's action is divided into an atomic incidents or episodes. The current incident is different from previous and there is no dependency between them For example, a pick and place robot to detect defective parts from the conveyor belts. In case of sequential environments, previous decision can affect all future decisions and the next ~~ager~~ action of the agent depends on what action has taken previously and what action it is supposed to take in future

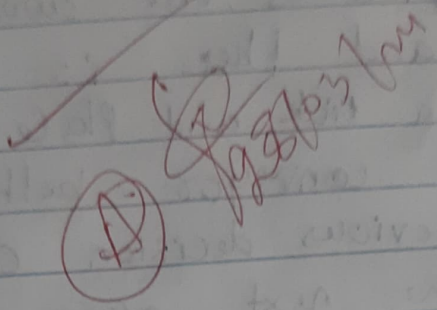
5) Static vs Dynamic

The static environment is the one where environment remains unchanged while the agent is performing given tasks like Sudoku puzzle or vaccums cleaner. If environment is not changing over time but is changed by an agent's performance it is a semi-dynamic environment like a chess game And if the environment changes while an agent is performing some task, it is dynamic environment like automatic car drivers

6) Discrete vs Continuous

When we have distinct and clearly defined inputs and outputs or precepts and actions, then it is called a discrete environment like chess while when a continuous input signal is received

by an agent, all precepts and actions cannot be defined beforehand. For example, an automatic car driving system.



THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Written Assignment 02

Q1] Discuss forward chaining and backward chaining algorithms. Illustrate working for

→ In AI, forward and backward chaining ~~is one~~ of are modes in which the inference engine commonly proceeds. The engine is component of the intelligent system which applies logical rules to knowledge base to infer new information from known facts.

Forward chaining a.k.a forward deduction or forward reasoning is the form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in forward direction to extract more data until a goal is reacheacd. It starts from known facts, triggers all rules ~~the~~ whose premises are satisfied and add their conclusion to known facts and repeats until problem is solved. It is a down-up approach and is data-driven commonly used in expert systems like CLIPS, business, production rule systems.

Backward chaining a.ka. backward deduction or reasoning starts with goal and works backward chaining through rules to find known facts that support the goal. It is a top-down approach. Goal is broken into sub-goals to prove facts true. It is a goal-driven approach used in game theory, authomated theorem proving tools,

inference engines, proof assistants, AI applications

Convert facts to FOL

1) It is a crime for American to sell weapons to enemy nation.

→ American(x) ∧ Weapon(y) ∧ Sell(x, y, z) ∧ enemy(z, America) ⟶ Criminal(x)

2) Nono is enemy

→ Enemy(Nono, America)

3) Nono has missiles

→ · Owns(Nono, x)
· Missile(x)

4) All misslies are sold by West

→ · Missile(x) ∧ Owns(Nono, x) ⟶ Sell(West, x, Nono)

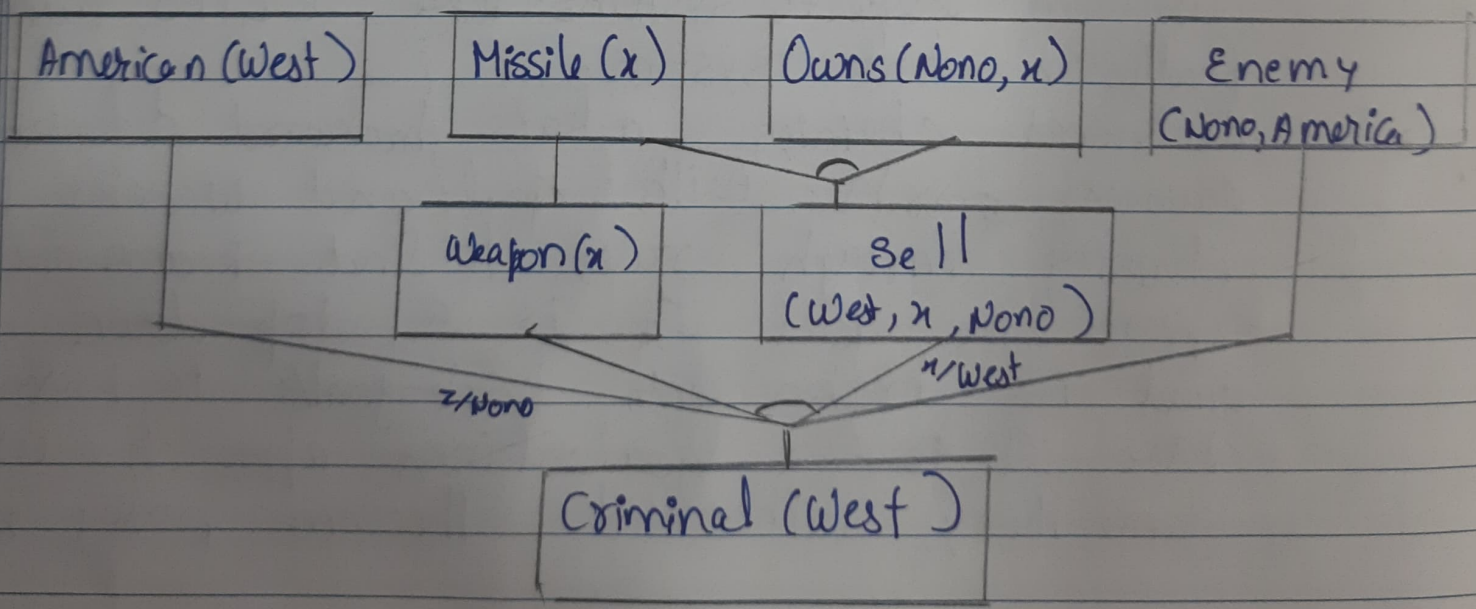
5) Missile is weapon.

→ Missile(x) ⟶ Weapon(x)

6) West is American.

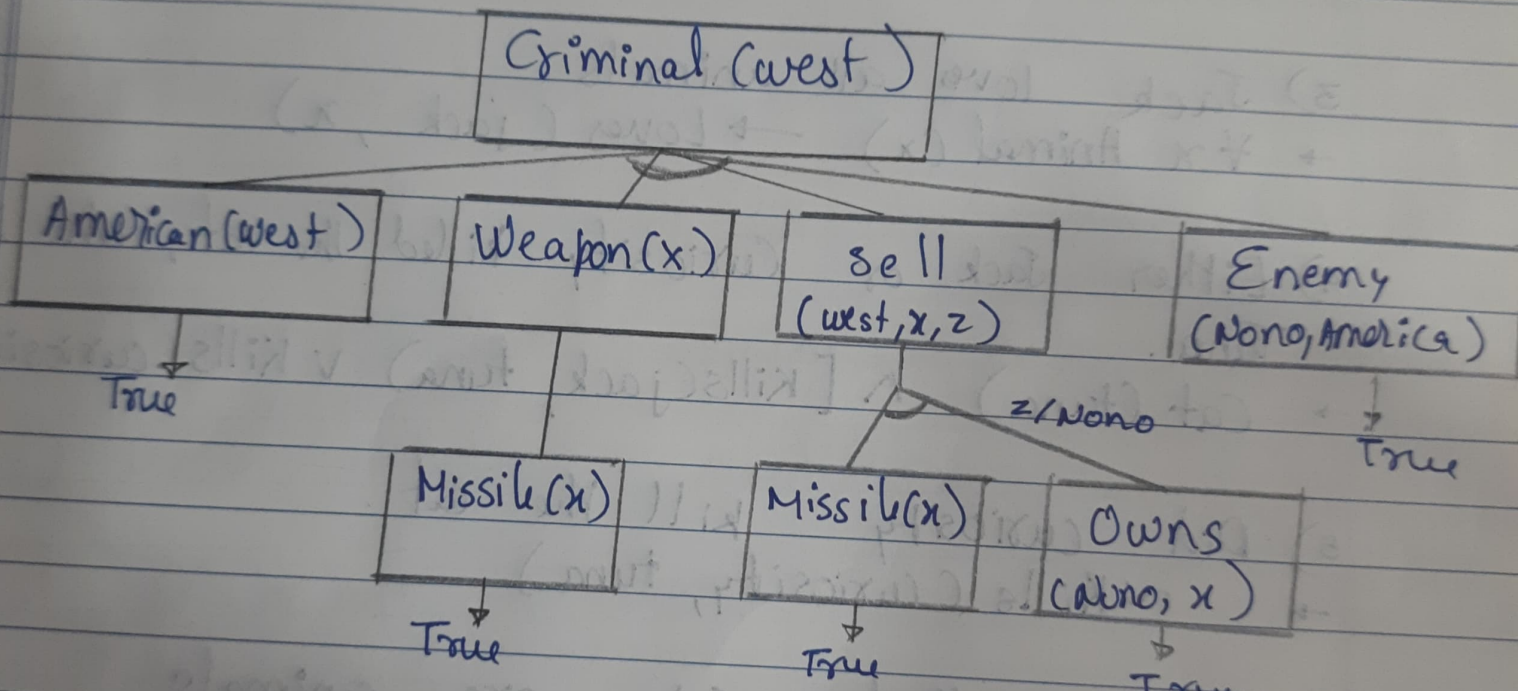
→ American(West)

Forward chaining



We start with known facts and choose sentences with no implications like 2, 3, 6. Next, we see facts which infer from available facts with satisfied premises. So 4 and 5 are added. At last, we see 1 is satisfied by {x/West, z/Nono, x/y} substitution and we reach our goal.

Backward Chaining



We start with goal. West is Criminal. Next we infer other facts from goal. 1 shows predicate Criminal (West) after substituting {West, x}. Next we extract further fact Missile (x) from Weapon (x) and Missle (x) and Owns (Nono, x) thus proving the statements.

Q2] Prove using Resolution -
Curiosity killed the cat

→ 1] Convert statements to FOL

1) Everyone who loves all animals is loved by someone
→ $\forall x [\forall y \ Animal(y) \rightarrow Loves(x,y)] \rightarrow \exists y \ Loves(y,x)$

2) Anyone who kills an animal is loved by no one
→ $\forall x [\exists z \ Animal(z) \wedge Kills(x,z)] \rightarrow \forall y \sim Loves(y,z)$

3) Jack loves all animals
→ $\forall x \ Animal(x) \rightarrow Loves(jack, x)$

4) Either Jack or Curiosity killed the cat, who is named Tuna.
→ $Cat(tuna) \wedge [Kills(jack, tuna) \vee Kills(curiosity, tuna)]$

5) Did curiosity kill the cat.
→ $\sim Kills(curiosity, tuna)$

Implied that cats are animals
→ $\forall x \ Cat(x) \rightarrow Animal(x)$

2] Converting FOL to CNF:

1) $Animal(F(x)) \vee Loves(G(x), x)$
2) $\sim Loves(x, F(x)) \vee Loves(G(x), x)$

3) ~ Animal (x) v loves (Jack, x)
4) Kills (Jack, Tuna) v Kills (curiosity, Tuna)
5) Cat (Tuna)
6) ~ Cat (x) v Animal (x)
7) ~ Kill (Curiosity, Tuna)

For converting FOL to CNF or the conjunctive Normal form the steps followed are -

1) Eliminate implications
$A \rightarrow B \equiv \sim A \vee B$

2) Move ~ inwards - $\sim \forall x\, p \equiv \exists x\, \sim p$

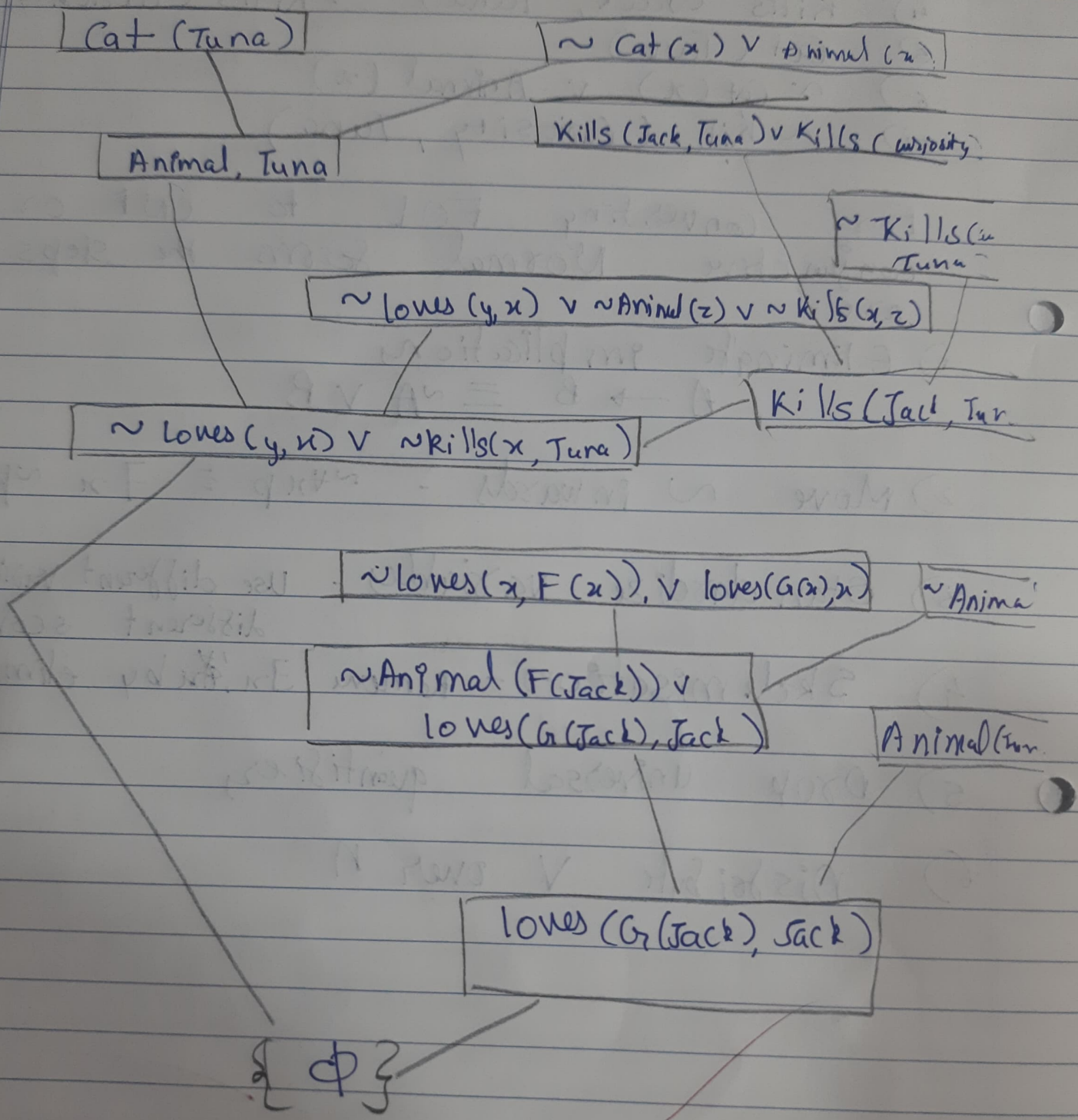
3) Standardize variables - Use different variable for different scopes

4) Skolemisation - Remove $\exists x$, by elimination

5) Drop universal quantifiers

6) Distribute $\vee$ over $\wedge$

Resolution Tree



Thus ~Kills(Curiosity, Tuna) is False
& ∴ Kills (Curiosity, Tuna) is True

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Q3 Discuss Natural Language Processing (NLP)

→ What is NLP?

NLP is a subfield of AI, widely used technology for personal assistants used in various business field. It works on speech provided by user, breaks it down for understanding and processes it. It deals with interaction between computer and humans and involves using computational techniques to process and analyze natural language data like text and speech with the goal of understanding meaning of language

→ Components of NLP

There two main components -

- NLU - Natural language understanding

The speech input gets transformed into useful representations in order to analyse varias aspects as it is ambiguous. There can be lexical ambiguity, syntactical or referential.

- NLG - Natural language Generation

To generate output text, intermediate representation must be converted back to natural language. Thus, sub processes like text planning, sentence planning and text realization.

→ Difficulties in NLP

- Language differences
- less training data

- Development time and resource requirements
- Navigating phrasing ambiguities
- Misspellings and Grammatical Errors
- Mitigating innate biases in NLP algorithms
- Words with multiple meanings
- Addressing multilingualism
- Reducing uncertainty and false positive
- Facilitating continuous conversations

→ Steps involved in NLP

- Lexical Analysis - Recognize, identify sentence structure.
- Syntactic Analysis - Analyze grammar of the sentence
- Semantic Analysis - Extracted actual meaning
- Discourse Integration - verify meaning of a sentence
- Pragmatic Analysis - Verify correctness in given context

→ Role of NLP in A.I

- Automatic Summarization
- Discourse Analysis
- Machine Translation
- Name entity Recognition
- Natural language generation
- Natural Language understanding
- Part-of-speech tagging
- Question answering
- Sentiment Analysis
- Speech recognition
- Speech Segmentation

- Word segmentation
- Word sense disambiguation
- Topic segmentation and recognition

Q4 What is robotics? Discuss role of AI in robotics. And application in health care and agriculture

→ Robot is any automatically operated machine that replaces human efforts and perform functions in a humanlike manner. They were originally built to handle monotonous tasks but have expanded to tasks like fighting fires, cleaning homes and surgeries.

Role of AI in

- Agriculture

The sector which is basis of human civilization. which also depends on seasonal sector depending on ideal weather conditions, optimal soil. The repetitive tasks in agriculture are a waste of farmer's time like seeding, weed control, harvesting, and can be done by robots like Ecorobotix.

- Health Care

Robots can help doctors in performing operations more precisely, be used as prosthetic limbs, provide therapy to patients etc. The da Vinci robot helps to perform surgeries related to heart, head, neck and other sensitive areas.