NCR Campus, Modinagar

(A Constituent SRM IST, Chennai, T.N.)

Delhi-Meerut Road, Sikari Kalan, Modinagar – 201204, GHAZIABAD (U.P.)

Artificial Intelligence Lab

Lab Record

(Jan – May 2022)

Name	Hardik Gupta
Reg No.	RA1911026030027
Degree / Branch	B-Tech, CSE with AI and ML
Semester / Section	6th Semester, A
Course Code	18CSC305J
Course Title	Artificial Intelligence
Faculty	Mr. Rahul Pandey

SRM IST, DELHI-NCR CAMPUS, MODINAGAR

Department Of Computer Science and Engineering

Registration Number

R	A	1	9	1	1	0	2	6	0	3	0	0	5	5

BONAFIDE CERTIFICATE

It is to be certified that the bonafide practical record submitted by **Hardik Gupta(RA1911026030027)** of 6th semester for Bachelor of Technology degree in the Department of Computer Science and Engineering, Delhi-NCR Campus, SRM IST has been done for the course **Artificial Intelligence Lab (18CSC305J)** during the academic semester session Jan 2022 – May 2022.

---------- ----------

Dr. R. P. Mahapatra

Head of the Department Assistant Professor

Computer Science & Engg. Computer Science & Engg.

Submitted for the University Examination held on _____

Examiner 1 Examiner 2

INDEX

EX NO: 1

DATE: 20/1/21

Implementation of toy problem

Problem Statement:

Stock buy and sell to maximize the profit: The cost of a stock on each day is given in an array, find the max profit that you can make by buying and selling in those days. For example, if the given array is {100, 180, 260, 310, 40, 535, 695}, the maximum profit can earn by buying on day 0, selling on day 3. Again, buy on day 4 and sell on day 6. If the given array of prices is sorted in decreasing order, then profit cannot be earned at all.

Tools Used: Google Collaboratory

Algorithm:

1. Input the stock prices as array.
2. Find the local minima and store it as starting index. If not exists, return.
3. Find the local maxima. and store it as ending index. If we reach the end, set the end as ending index.
4. Update the solution (Increment count of buy sell pairs)
5. Repeat the above steps if end is not reached.

Optimization Technique:

1. A simple approach is to try buying the stocks and selling them on every single day when profitable and keep updating the maximum profit so far.
2. If we are allowed to buy and sell only once, then we can use following algorithm. Maximum difference between two elements. Here we are allowed to buy and sell multiple times. Use local descent technique to get all prices as input and use global optima to find maxima and minima.
3. The outer loop runs till I become n-1. The inner two loops increment value of I in every iteration. So overall time complexity is O(n).

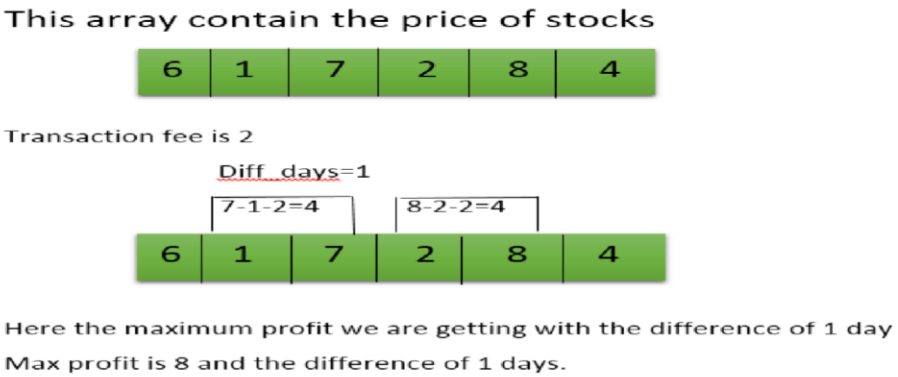


Fig 1.1 optimization technique representation

Programming Code:

```
def stockBuySell(price, n):


    if (n == 1):
        return


    i = 0
    while (i < (n - 1)):


        while ((i < (n - 1)) and
                (price[i + 1] <= price[i])):
            i += 1


        if (i == n - 1):
            break


        buy = i
        i += 1


        while ((i < n) and (price[i] >= price[i - 1])):
            i += 1


        sell = i - 1
```

```
        print("Suggestion to Buy on day: ",buy,"\t",
                "Suggstion to Sell on day: ",sell)


n = int(input("Enter the trading days: "))
print("Input the daily stock price: ")
price = [int(input()) for i in range(n)]


print(stockBuySell(price, n))
```

Output Screenshots:

Result: We are able to solve the stock buying and selling puzzle to earn max profit. We did iteration and used it to find local minima and maxima which in turn calculated the greatest profit margins needed to get puzzle solved.

EX NO: 2

DATE: 28/1/21

Developing agent program for real world

Problem Statement:

Given a graph color its vertices and edges such that no two adjacent vertices or edges have the same color using minimum number of colors and return the Chromatic number.

Tools Used: Google Collaboratory, VS code with windows terminal

Algorithm:

Greedy algorithm approach:

1. Color first vertex with first color.
2. Loop for remaining V-1 vertices.
3. Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it.
4. If all previously used colors appear on vertices adjacent to v, assign a new color to it.
5. Repeat the following for all edges. 4. Index of color used is the chromatic number.

Optimization Technique:

1. Graph coloring problem is to assign colors to certain elements of a graph subject to certain constraints. Vertex coloring is the most common graph coloring problem.
2. The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using the same color.
3. The other graph coloring problems like Edge Coloring (No vertex is incident to two edges of same color) and Face Coloring (Geographical Map Coloring) can be transformed into vertex coloring.
4. Chromatic Number: The smallest number of colors needed to color a graph G is called its chromatic number. For example, the following can be colored at least 2 colors.

Programming Code:

Vertex coloring:

```
class Graph:


    def __init__(self, edges, N):

        self.adj = [[] for _ in range(N)]


        for (src, dest) in edges:
            self.adj[src].append(dest)
            self.adj[dest].append(src)



def colorGraph(graph):


    result = {}


    for u in range(N):


        assigned = set([result.get(i) for i in graph.adj[u] if i in result])


        color = 1
        for c in assigned:
            if color != c:
                break
            color = color + 1


        result[u] = color

    for v in range(N):
        print("Color assigned to vertex", v, "is", colors[result[v]])



if __name__ == '__main__':
```

```
    colors = ["", "RED", "GREEN", "BLUE", "PINK", "YELLOW", "ORANGE",
              "BLACK", "BROWN", "WHITE", "PURPLE", "VOILET"]


    edges = [(0, 1), (0, 2), (0, 4), (0, 5), (1, 2), (2, 3), (3, 4), (4, 5)]


    N = 6


    graph = Graph(edges, N)


    colorGraph(graph)
```

Edge coloring:

```
#include<bits/stdc++.h>
using namespace std;
int n, e, i, j;
vector<vector<pair<int, int> > > g;
vector<int> color;
bool v[111001];
void col(int n) {
   queue<int> q;
   int c = 0;
   set<int> vertex_colored;
   if(v[n])
      return;
      v[n] = 1;
   for(i = 0;i<g[n].size();i++) {
      if(color[g[n][i].second]!=-1) {
         vertex_colored.insert(color[g[n][i].second]);
      }
   }
   for(i = 0;i<g[n].size();i++) {
      if(!v[g[n][i].first]) {
         q.push(g[n][i].first);
      }
      if(color[g[n][i].second]==-1) {
         while(vertex_colored.find(c)!=vertex_colored.end())
            c++;
            color[g[n][i].second] = c;
            vertex_colored.insert(c);
            c++;
```

```
      }
   }
   while(!q.empty()) {
      int temp = q.front();
      q.pop();
      col(temp);
   }
   return;
}
int main() {
   int u,w;
   set<int> empty;
   cout<<"Enter number of vertices and edges respectively:";
   cin>>n>>e;
   cout<<"\n";
   g.resize(n); //number of vertices
   color.resize(e,-1); //number of edges
   memset(v,0,sizeof(v));
   for(i = 0;i<e;i++) {
      cout<<"\nEnter edge vertices of edge "<<i+1<<" :"<<"\n";
      cin>>u>>w;
      u--; w--;
      g[u].push_back(make_pair(w,i));
      g[w].push_back(make_pair(u,i));
   }
   col(0);
   for(i = 0;i<e;i++) {
      cout<<"Edge "<<i+1<<" is coloured with colour "<<color[i]+1
      << "\n";
   }
}
```

Face coloring:

```
import networkx as nx

G = nx.Graph()
colors = {0:"green", 1:"blue", 2:"red", 3:"yellow"}
G.add_nodes_from([1,2,3,4,5])
G.add_edges_from([(1,2), (2,3), (3,5), (3,4), (4,5)])

nodes = list(G.nodes)
edges = list(G.edges)
some_colors = ['green','blue','red','yellow']

no_of_faces = len(edges)+2-len(nodes)-1
def regionColour(regions):
    print("NO OF FACES : "+str(regions+3))
    for i in range(1,regions+4):
```

```
        print("FACE: "+some_colors[i%4])

regionColour(no_of_faces)
```

Output Screenshots:

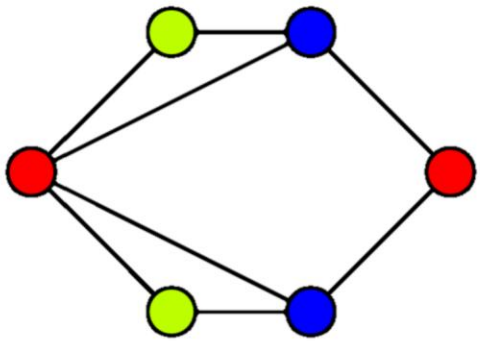


Fig 2.1 Vertex output

```
Color assigned to vertex 0 is RED
Color assigned to vertex 1 is GREEN
Color assigned to vertex 2 is BLUE
Color assigned to vertex 3 is RED
Color assigned to vertex 4 is GREEN
Color assigned to vertex 5 is BLUE
```

Fig 2.2 vertex coloring output on google Collaboratory

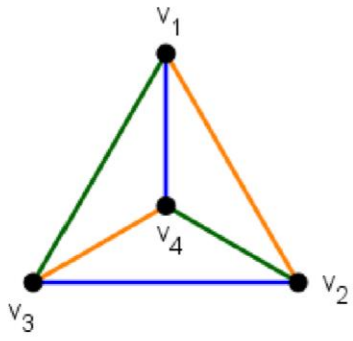


Fig 2.3 Edge coloring output

```
Enter number of vertices and edges respectively:4 6


Enter edge vertices of edge 1 :
1 2

Enter edge vertices of edge 2 :
2 3

Enter edge vertices of edge 3 :
1 4

Enter edge vertices of edge 4 :
4 2

Enter edge vertices of edge 5 :
3 1

Enter edge vertices of edge 6 :
3 4
Edge 1 is coloured with colour 1
Edge 2 is coloured with colour 2
Edge 3 is coloured with colour 2
Edge 4 is coloured with colour 3
Edge 5 is coloured with colour 3
Edge 6 is coloured with colour 1
```

Fig 2.4 Edge coloring output on terminal

Fig 2.6 Face coloring output

```
NO OF FACES : 4
FACE: blue
FACE: red
FACE: yellow
FACE: green
```

Fig 2.5 Face coloring output on Collab

Result: Experiment is completed and no 2 adjacent vertices, edges or faces have same color and the experiment is done using greedy algorithm approach.

EX NO: 3

DATE: 06/02/21

Implementation of constraint satisfaction problems

Problem Statement:

Given 6 cryptarithmetic problems, we need to find solution for all and write code for one problem.

Tool used: Google Collaboratory

Algorithm:

1. The library is initialized,
2. The FD variables are declared, and initial bounds to them assigned (note the special bounds for the variables M and S),
3. An array packing all FD variables is created,
4. The rest of the constraints are generated (all variables must be different, and the equality defining the arithmetic operation must hold),
5. A call to the solver is made, to search for values and assign them to the variables, and
6. The final solution is printed

Optimization technique:

1. We have used Constraint programming, which is an optimization technique that emerged from the field of artificial intelligence.
2. It is characterized by two key ideas: To express the optimization problem at a high level to reveal its structure and to use constraints to reduce the search space by removing, from the variable domains, values that cannot appear in solutions.
3. These lectures cover constraint programming in detail, describing the language of constraint programming, its underlying computational paradigm and how it can be applied in practice.

Calculations:

1. Question: SEND + MORE = MONEY

S E N D

\+ M O R E

M O N E Y

- From Column 5, M=1, since it is the only carry-over possible from the sum of 2 single digit numbers in column 4.
- To produce a carry from column 4 to column 5 'S + M' is at least 9 so 'S=8or9' so 'S+M=9or10' & so 'O = 0 or 1'. But 'M=1', so 'O = 0'.
- If there is carry from Column 3 to 4 then 'E=9' & so 'N=0'. But 'O = 0' so there is no carry & 'S=9' & 'c3=0'.
- If there is no carry from column 2 to 3 then 'E=N' which is impossible, therefore there is carry & 'N=E+1' & 'c2=1'.
- If there is carry from column 1 to 2 then 'N+R=E mod 10' & 'N=E+1' so 'E+1+R=E mod 10', so 'R=9' but 'S=9', so there must be carry from column 1 to 2. Therefore 'c1=1' & 'R=8'.
- To carry 'c1=1' from column 1 to 2, we must have 'D+E=10+Y' as Y cannot be 0/1 so D+E is at least 12. As D is at most 7 & E is at least 5 (D cannot be 8 or 9 as it is already assigned). N is at most 7 & 'N=E+1' so 'E=5or6'.
- If E were 6 & D+E at least 12 then D would be 7, but 'N=E+1' & N would also be 7 which is impossible. Therefore 'E=5' & 'N=6'. D+E is at least 12 so that we get 'D=7' & 'Y=2'.

SOLUTION:

9 5 6 7

+ 1 0 8 5

---

1 0 6 5 2

VALUES:

S=9

E=5

N=6

D=7

M=1

O=0

R=8

Y=2

2. Question: BASE + BALL = GAMES

B A S E

+ B A L L

G A M E S

- Assuming numbers can't start with 0, G is 1 because two four-digit numbers can't sum to 20000 or more.
- SE+LL=ES or 1ES.

- If it is ES, then LL must be a multiple of 9 because SE and ES are always congruent mod 9. But LL is a multiple of 11, so it would have to be 99, which is impossible.
- So SE+LL=1ES. LL must be congruent to 100 mod 9. The only multiple of 11 that works is 55, so L is 5.
- SE+55=1ES. This is possible when E+5=S. The possibilities for ES are 27, 38, or 49.
- BA+BA+1=1AM. B must be at least 5 because B+B (possibly +1 from a carry) is at least 10.
- If A is less than 5, then A+A+1 does not carry, and A must be even. Inversely, if A is greater than 5, it must be odd. The possibilities for A are 0, 2, 4, 7, or 9.

    * 0 does not work because M would have to be 1.

    * 2 and 7 don't work because M would have to be 5.

    * 9 doesn't work because M would also have to be 9.

- So, A is 4, M is 9, and B is 7. This leaves 38 as the only possibility for ES. The full equation is:

7483

+ 7455

14938

3. Question: TWO + TWO = FOUR

T W O

+T W O

FOUR

- F = 1 for carry over T >= 5.
- ‘O’ can’t be 0 as R will be 0. So IT can't be 5 so let T>=6
- If T = 6, O = 2 and R = 4 and W + W = U for W can’t be 1,2,6,4. W < 4 as to avoid carry over. WE can't be 3 as U will be 6.
- So, T = 7, so, O can be 4 or 5 depending on whether or W + W > 10. If O is 4 then R = 8. WE can't be 1,2. So W = 3
- If W = 3 then U = 6 hence

Here is the answer:

7 3 4

\+ 7 3 4

---

1 4 6 8

---

4. Question: CROSS + ROADS= DANGER

C R O S S

+ R O A D S

D A N G E R

- Since it is already mentioned that the carry value of resultant cannot be 0 then let's presume that the carry value of D is 1. We know that the sum of two similar values is even, hence R will have an even value Hence S+S= R So R is an even number for sure. So, the value of R can b (0, 2, 4, 6, 8)

- Value of R cannot be 0 as two different values cannot be allotted the same digit, (if S=10 then their sum = 20 carry forward 2, then the value of R= 0) which is not possible.[SEP]IF S= 1: Not possible since D has the same value. IF S = 2
- Then R= 4 which is possible Hence S= 2 and R= 4C4+C+R= A+10C4+C+4= A+10C4+C>5 (Being the value of carry will be generated when the value of C is greater than 5
- C= 9+S+D = E +2+1= E
- Therefore E= 3+C+R= A+10C4+9+4= A+10Therefore A= 3 but it cannot be possible as E= 3.
- Now let's Consider S+D+C1= E2+1+0= 3. Therefore E= 3 making C2= 0 since 2+1=3.
- Now let's consider the equation again:C+R+C4= A+109+4+0= A+1013= A+10.
- Therefore A= 3 but E= 3.
- So, A is not equal to 3.
- Again, considering A= 5 and S+D+C1= E3+1+0= E therefore E= 4 and C2= 0.
- Now considering the equation R+O = N+ 0= N. So, 6+0+C3 <= t 3.
- Let C3= 1, then O< or equal to 2. That is O= 0, 1, 2.
- Let O =2. Again, considering R+O+C3= N6+2+1= N. Hence, N= 9 but C= 9 so N cannot be equal to 9. Now let N= 8 and C3= 0. Let us consider equation G= 7. Hence, D= 1 S= 3 A= 5 G= 7 C= 9 O= 2 E= 4 R= 6 N= 8.

9 6 2 3 3

\+ **6 2 5 1 3**

---

1 5 8 7 4 6

---

5. Question: AA + BB + CC = ABC

```
  A A
+ B B
+ C C
-----
A B C
-----
```

- The digits are distinct and positive.
- Let's first focus on the value A, when we add three 2-digit numbers the most you get is in the 200's (ex: AA + BB + CC = ABC à 99 + 88 + 77 = 264).
- From this, we can tell that the largest value of A can be 2. So, Either A = 1 or A = 2.
- Now focus on value B, let's take the unit digit of the given question: A + B +C = C (units).
- This can happen only if A + B = 0 (in the units) à A and B add up to 10.
- Two possibilities: 11 + 99 + CC = 19C à (1) or 22 + 88 + CC = 28C (2)
- Take equation (2), 110 + CC = 28C
- Focus on ten's place, 1 + C = 8, here C = 7. Then 22 + 88 + 77 = 187
- Thus, Equation (2) is not possible.
- From Equation (1), 11+99+CC = 19C à 110 + CC =19C à 1 + C = 9, then C = 8.
- 11 + 99 + 88 = 198. Hence, solved A = 1, B = 9 and C = 8

6. Question: N O + G U N + N O = H U N T

```
  N O
+G U N
+ N O
-----
H U N T
-----
```

- Here H = 1, from the NUNN column we must have "carry 1," so G = 9, U = zero.

- Since we have “carry” zero or 1 or 2 from the “ONOT” column, correspondingly we have N + U = 10 or 9 or 8. But duplication is not allowed, so N = 8 with “carry 2” from ONOT.
- Hence, O + O = T + 20 – 8 = T + 12. Testing for T = 2, 4 or 6, we find only T = 2 acceptable, O = 7.
- So we have,

```
  87
+ 908
+  87
------
1082
------
```

Programming Code:

For Question: SEND + MORE = MONEY

```
def solutions():
  # letters = ('s', 'e', 'n', 'd', 'm', 'o', 'r', 'y')
  all_solutions = list()
  for s in range(9, -1, -1):
    for e in range(9, -1, -1):
      for n in range(9, -1, -1):
        for d in range(9, -1, -1):
          for m in range(9, 0, -1):
            for o in range(9, -1, -1):
              for r in range(9, -1, -1):
                for y in range(9, -1, -1):
                  if len(set([s, e, n, d, m, o, r, y])) == 8:
                    send = 1000 * s + 100 * e + 10 * n + d
                    more = 1000 * m + 100 * o + 10 * r + e
                    money = 10000 * m + 1000 * o + 100 * n + 10 * e + y


                    if send + more == money:
                      all_solutions.append((send, more, money))
```

```
    return all_solutions


print(solutions())
```

Output Screenshots:

```
def solutions():
    # letters = ('s', 'e', 'n', 'd', 'm', 'o', 'r', 'y')
    all_solutions = list()
    for s in range(9, -1, -1):
        for e in range(9, -1, -1):
            for n in range(9, -1, -1):
                for d in range(9, -1, -1):
                    for m in range(9, 0, -1):
                        for o in range(9, -1, -1):
                            for r in range(9, -1, -1):
                                for y in range(9, -1, -1):
                                    if len(set([s, e, n, d, m, o, r, y])) == 8:
                                        send = 1000 * s + 100 * e + 10 * n + d
                                        more = 1000 * m + 100 * o + 10 * r + e
                                        money = 10000 * m + 1000 * o + 100 * n + 10 * e + y

                                        if send + more == money:
                                            all_solutions.append((send, more, money))
    return all_solutions

print(solutions())
```

```
[(9567, 1085, 10652)]
```

Fig 3.1 output

Result: We have taken 6 cryptarithmetic problems and solved them as well as programmed one problem into code and displayed the output.

EX NO: 4

DATE: 11/2/21

Implementation and Analysis of DFS and BFS for an application

Problem Statement:

DFS:

Xenny's is competing in a race and his car has X liters of fuel. There are N milestones in the competition. It takes no fuel at all to travel between gas stations, but at the \(i^{th}\) gas station, \(P_i\) amount of petrol is drained.

Find the number milestones Xenny crosses before his car gets out of fuel.

BFS: The witch of Hegwarts

Little PandeyG is a curious student, studying in HEgwarts. Being smarter, faster and displaying more zeal for magic than any other student, one by one he managed to impress the three hidden witches of the school. They knew his secret desire to be a warrior, so each of them gave him some super power to use if he's up for a fight against one of his enemies.

Tools Used: Google Collaboratory

Algorithm:

DFS:

1. SET STATUS = 1 (ready state) for each node in G
2. Push the starting node A on the stack and set its STATUS = 2 (waiting state)
3. Repeat Steps 4 and 5 until STACK is empty
4. Pop the top node N. Process it and set its STATUS = 3 (processed state)
5. Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their
6. STATUS = 2 (waiting state)
7. [END OF LOOP]
8. EXIT

BFS:

1. SET STATUS = 1 (ready state)
   for each node in G
2. Enqueue the starting node A
   and set its STATUS = 2
   (waiting state)
3. Repeat Steps 4 and 5 until
   QUEUE is empty
4. Dequeue a node N. Process it
   and set its STATUS = 3
   (processed state).
5. Enqueue all the neighbors of
   N that are in the ready state
   (whose STATUS = 1) and set
   their STATUS = 2
   (waiting state)
   [END OF LOOP]
6. EXIT

Optimization Technique:

1. For solving DFS gas stations we need N milestones in the competition. It takes no fuel at all to travel between gas stations, so we need to iterate each input till we get petrol value left=0.
2. For BFS in the witch of hegwarts, we need the first witch: She gave PandeyG the power to take away one unit of strength away from his enemies. - Eg .
3. The second witch: Jealous of the first witch, the second one gave the kid the power to halfen the strength of his enemies. - E.g.
4. The third witch: Even better, she gave him the power to reduce the strength of his enemies to one third of what it initially was. - E.g.
5. The witches, though, clearly told him that he'll be only able to use these powers if the strength of the opponent is an integer, otherwise not. Given the value 'k' - k being the units of the enemy of PandeyG's strength, we can find out the minimum number of magical hits he'll be needing to defeat his opponent, using his powers.

Programming Code:

DFS code:

```
n,k=map(int,input().split())
l=list(map(int,input().split()))
cou=0
for i in range(n):
    if(k>=l[i]):
        cou+=1
        k-=l[i]
    else:
        break
print(cou+1)
```

BFS code:

```
from sys import stdin,stdout
def fn(n):
    if n<=1:return 0
    if n in dp:return dp[n]
    opt1=opt2=float('inf')
    opt1=1+(n&1)+fn(n//2)
    opt2=1+(n%3)+fn(n//3)
    dp[n]=min(opt1,opt2)
    return dp[n]
for i in range(int(stdin.readline())):
    n=int(stdin.readline())
    # a=list(map(int,stdin.readline().split()))
    dp={}
    print(fn(n))
```

Output Screenshots:

Fig 4.1 Output for DFS

Fig 4.2 Output for BFS

Result: For the selected problems we have got the output in both BFS and DFS using google collab.

EX NO: 5

DATE: 25/2/21

Developing Best first search and A* Algorithm for real world problems

Problem Statement:

We have task of performing Best first search and then A* algorithm for finding node path of given scenarios.

Tools Used: Google Collaboratory

Algorithm:

Best first search:

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
3. Repeat the next steps until GOAL node is reached
4. If OPEN list is empty, then EXIT the loop returning 'False'
5. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also capture the information of the parent node
6. If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path
7. If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list
8. Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

A*search:

1. Initialize the open list
2. Initialize the closed list

   put the starting node on the open

   list (you can leave its f at zero)
3. while the open list is not empty

   a) find the node with the least f on the open list, call it "q"

b) pop q off the open list

c) generate q's 8 successors and set their parents to q

d) for each successor

i) if successor is the goal, stop search successor.g = q.g + distance between successor and q

successor.h = distance from goal to

successor (This can be done using many

ways, we will discuss three heuristics-

Manhattan, Diagonal and Euclidean

Heuristics)

successor.f = successor.g + successor.h

ii) if a node with the same position as

successor is in the OPEN list which has a

lower f than successor, skip this successor

iii) if a node with the same position as

successor is in the CLOSED list which has

a lower f than successor, skip this successor

otherwise, add the node to the open list

end (for loop)

e) push q on the closed list

end (while loop)

4. EXIT

Optimization Technique:

1. There are various ways to identify the 'BEST' node for traversal and accordingly there are various flavours of BFS algorithm with different heuristic evaluation functions f(n). We will cover two most popular versions of the algorithm in this blog, namely Greedy Best First Search and A* Best First Search.
2. Even though you would find that both Greedy BFS and A* algorithms find the path equally efficiently, number of steps, you may notice that the A* algorithm is able to come up with is a more optimal path than Greedy BFS.
3. Both Greedy BFS and A* are Best first searches but Greedy BFS is neither complete, nor optimal whereas A* is both complete and optimal. However, A* uses more memory than Greedy BFS, but it guarantees that the path found is optimal.

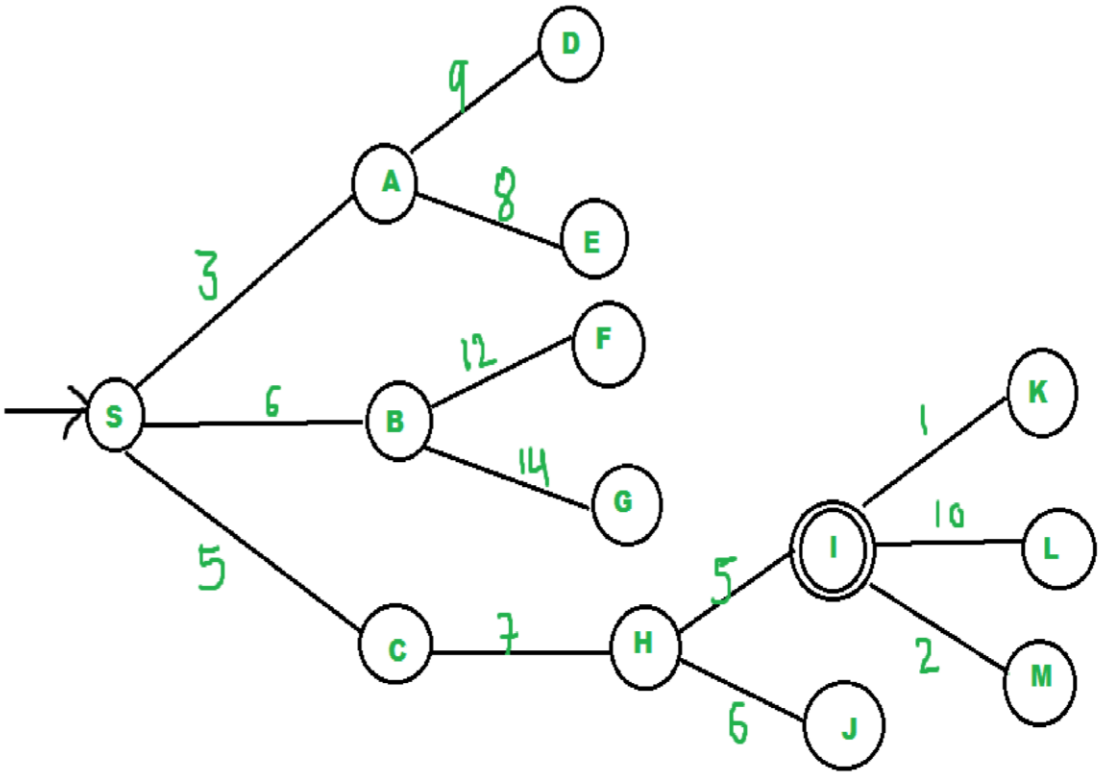


Fig5.1 BFS optimization

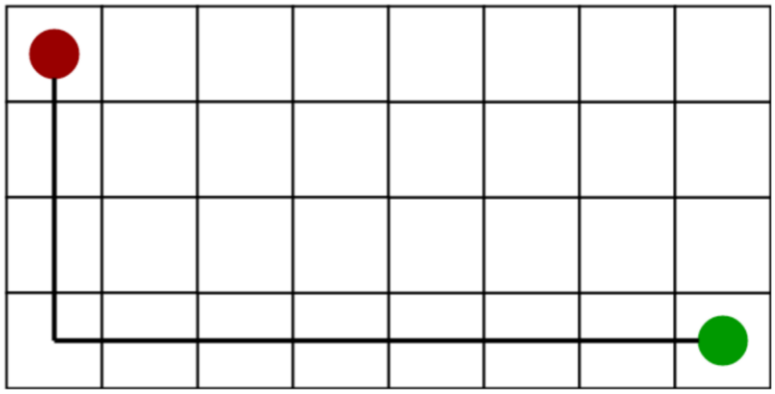


Fig 5.2 A* node path 1

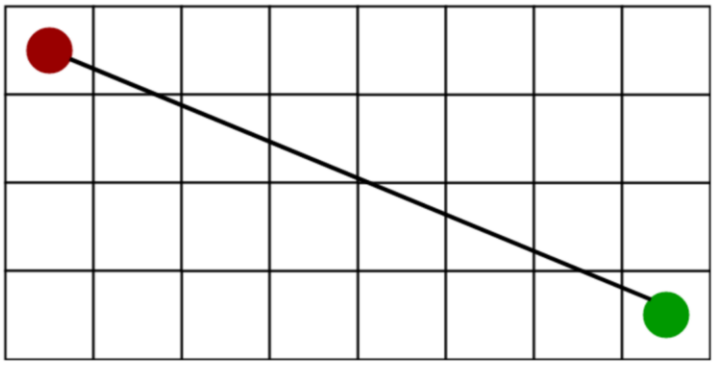


Fig 5.2 A* node path 2

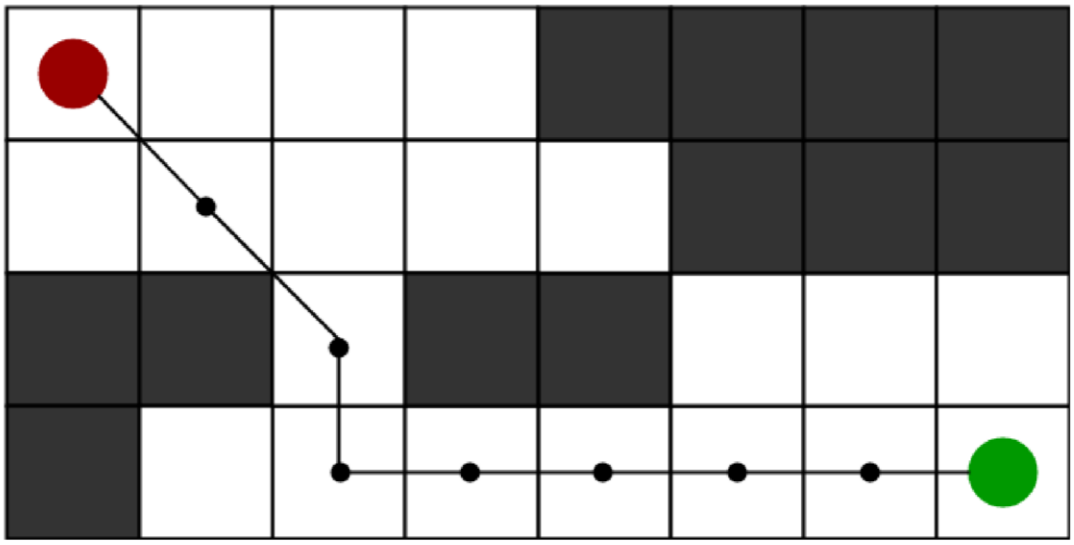


Fig5.3 A* final node path

Programming Code:

BFS:

```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]



def best_first_search(source, target, n):
    visited = [0] * n
    visited = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]

        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()



def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))


addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)
```

```
source = 0
target = 9
best_first_search(source, target, v)
```

A* search:

```
from collections import deque

class Graph:


   def __init__(self, adjacency_list):
       self.adjacency_list = adjacency_list

   def get_neighbors(self, v):
       return self.adjacency_list[v]

   def h(self, n):
       H = {
           'A': 1,
           'B': 1,
           'C': 1,
           'D': 1
       }

       return H[n]

   def a_star_algorithm(self, start_node, stop_node):

       open_list = set([start_node])
       closed_list = set([])

       g = {}

       g[start_node] = 0

       parents = {}
       parents[start_node] = start_node

       while len(open_list) > 0:
           n = None


           for v in open_list:
```

```
                if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                    n = v;

            if n == None:
                print('Path does not exist!')
                return None

            if n == stop_node:
                reconst_path = []

                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]

                reconst_path.append(start_node)

                reconst_path.reverse()

                print('Path found: {}'.format(reconst_path))
                return reconst_path

            for (m, weight) in self.get_neighbors(n):
                if m not in open_list and m not in closed_list:
                    open_list.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_list:
                            closed_list.remove(m)
                            open_list.add(m)

            open_list.remove(n)
            closed_list.add(n)

        print('Path does not exist!')
        return None

adjacency_list = {
   'A': [('B', 1), ('C', 3), ('D', 7)],
   'B': [('D', 5)],
   'C': [('D', 12)]
}
graph1 = Graph(adjacency_list)
```

```
graph1.a_star_algorithm('A', 'D')
```

Output Screenshots:

```
0 1 3 2 8 9
```

Fig 5.1 output for BFS

```
Path found: ['A', 'B', 'D']
['A', 'B', 'D']
```

Fig 5.2 output for A*

Result: For the given scenario Best first search and A* search operation is completed and output in printed in google Collaboratory.

EX NO: 6

DATE: 4/3/21

Implementation of minimax algorithm for an application

Problem Statement:

Implement Min-max Algorithm and create a Program for two-person game of Tic Tac Toe and determine the result between the players through the Algorithm.

Tools Used: Google Collaboratory

Algorithm:

1. Create a list of numbers as keys, which will be used for board.
2. Append the keys in the board as an array.
3. Start a game for the first player and take input, and mark its place as 'X' or 'O'.
4. If the place is already marked, the return place is already filled.
5. Change players after every move.
6. Same procedure for the next player.
7. Well check after five moves if any player won or not.
8. If the same mark is placed in a connection for any player then the player will be declared winner.
9. If neither player wins and the board is full, then it's declared as tie.

Optimization Technique:

1. Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.
2. In Minimax the two players are called maximiser and minimizer. The **maximiser** tries to get the highest score possible while the **minimizer** tries to do the opposite and get the lowest score possible.

3. Every board state has a value associated with it. In a given state if the maximiser has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

4. Create start node as a MAX node with current board configuration

5. Expand nodes down to some depth (i.e., ply) of lookahead in the game.

6. Apply the evaluation function at each of the leaf nodes

7. Obtain the “back up" values for each of the non-leaf nodes from its
8. children by Minimax rule until a value is computed for the root node.

9. Pick the operator associated with the child node whose backed up
10. value determined the value at the root as the move for MAX

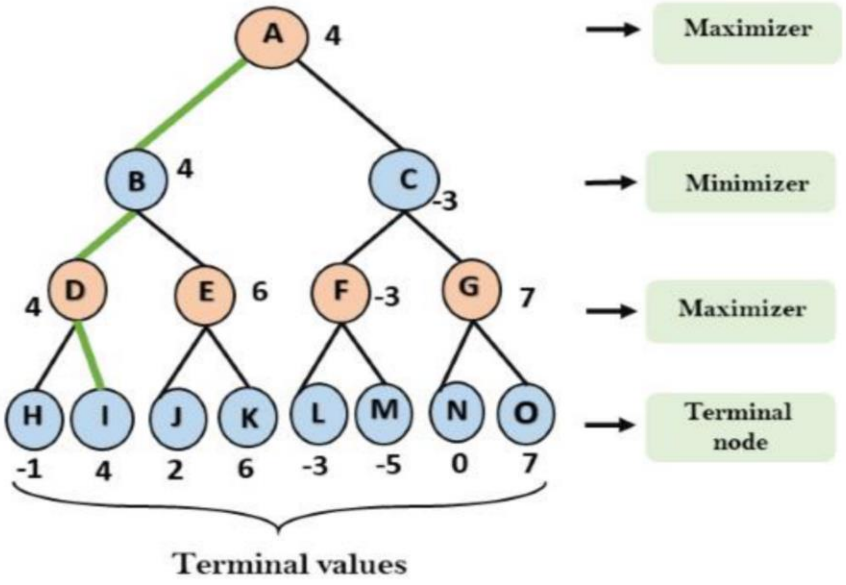


Fig 6.1 Minmax algorithm working

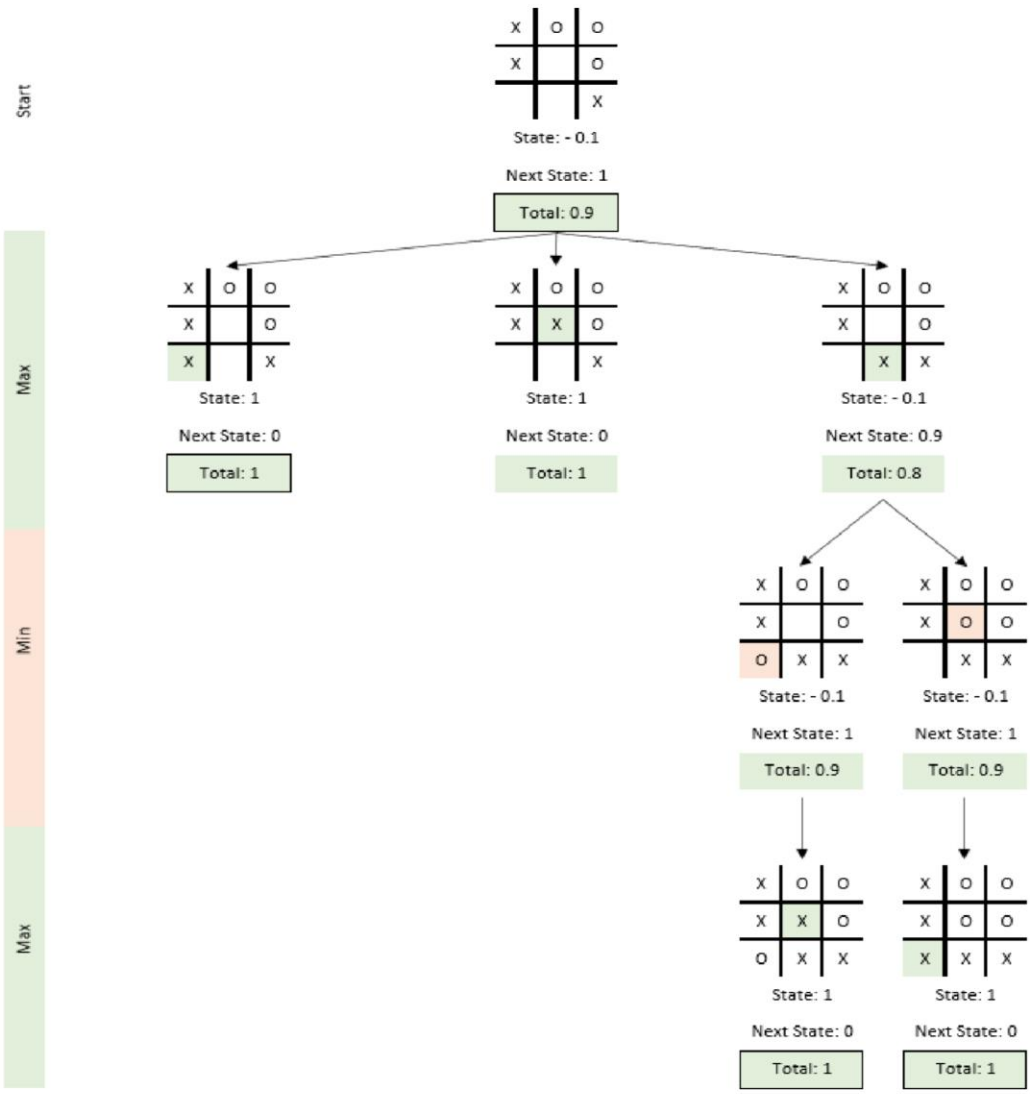


Fig 6.2 Tic tac toe using MinMax

Programming Code:

```
import sys
import random

class TicTacToeGame:

    def __init__(self, rows:int, columns:int, goal:int, max_depth:int=4):


        self.state = []
        self.tiles = {}
        self.inverted_tiles = {}
        tile = 0
        for y in range(rows):
            row = []
            for x in range(columns):
                row += '.'
                tile += 1
                self.tiles[tile] = (y, x)
                self.inverted_tiles[(y, x)] = tile
            self.state.append(row)

        self.goal = goal

        self.vectors = [(1,0), (0,1), (1,1), (-1,1)]

        self.rows = rows
        self.columns = columns
        self.max_row_index = rows - 1
        self.max_columns_index = columns - 1
        self.max_depth = max_depth

        self.winning_positions = []
        self.get_winning_positions()


        self.player = random.choice(['X', 'O'])
```

```
    def get_winning_positions(self):

        for y in range(self.rows):
            for x in range(self.columns):


                for vector in self.vectors:


                    sy, sx = (y, x)

                    dy, dx = vector

                    counter = 0

                    positions = []
                    while True:

                        positions.append(self.inverted_tiles.get((sy, sx)))

                        if (len(positions) == self.goal):

                            self.winning_positions.append(positions)

                            break

                        sy += dy
                        sx += dx


                        if(sy < 0 or abs(sy) > self.max_row_index or sx < 0 or
abs(sx) > self.max_columns_index):
                            break

    def play(self):
        # Variables
```

```
        result = None

        print('Starting board')
        while True:

            self.print_state()

            if (self.player == 'X'):

                print('Player X moving (AI) ...')

                max, py, px, depth = self.max(-sys.maxsize, sys.maxsize)

                print('Depth: {0}'.format(depth))
                if(depth > self.max_depth):
                    py, px = self.get_best_move()

                self.state[py][px] = 'X'

                result = self.game_ended()
                if(result != None):
                    break

                self.player = 'O'
            elif (self.player == 'O'):


                print('Player O moving (Human) ...')

                min, py, px, depth = self.min(-sys.maxsize, sys.maxsize)

                print('Depth: {0}'.format(depth))
                if(depth > self.max_depth):
                    py, px = self.get_best_move()

                print('Recommendation: {0}'.format(self.inverted_tiles.get((py
, px))))
```

```
            number = int(input('Make a move (tile number): '))
            tile = self.tiles.get(number)

            if(tile != None):


                py, px = tile
                self.state[py][px] = 'O'

                result = self.game_ended()
                if(result != None):
                    break

                self.player = 'X'
            else:
                print('Move is not legal, try again.')

    self.print_state()
    print('Result: {0}'.format(result))

def get_best_move(self):

    heuristics = {}

    empty_cells = []
    for y in range(self.rows):
        for x in range(self.columns):
            if (self.state[y][x] == '.'):
                empty_cells.append((y, x))

    for empty in empty_cells:

        number = self.inverted_tiles.get(empty)

        for win in self.winning_positions:
```

```
                if(number in win):

                    player_x = 0
                    player_o = 0
                    start_score = 1
                    for box in win:

                        y, x = self.tiles[box]

                        if(self.state[y][x] == 'X'):
                            player_x += start_score if self.player == 'X' else
start_score * 2
                            start_score *= 10
                        elif (self.state[y][x] == 'O'):
                            player_o += start_score if self.player == 'O' else
start_score * 2
                            start_score *= 10

                    if(player_x == 0 or player_o == 0):

                        score = max(player_x, player_o) + start_score

                        if(heuristics.get(number) != None):
                            heuristics[number] += score
                        else:
                            heuristics[number] = score

        best_move = random.choice(empty_cells)
        best_count = -sys.maxsize
        for key, value in heuristics.items():
            if(value > best_count):
                best_move = self.tiles.get(key)
                best_count = value

        return best_move

    def game_ended(self) -> str:
```

```
        result = self.player_has_won()
        if(result != None):
            return result

        for y in range(self.rows):
            for x in range(self.columns):
                if (self.state[y][x] == '.'):
                    return None

        return 'It is a tie!'


    def player_has_won(self) -> str:


        for y in range(self.rows):
            for x in range(self.columns):


                for vector in self.vectors:


                    sy, sx = (y, x)

                    dy, dx = vector

                    steps = 0
                    player_x = 0
                    player_o = 0

                    while steps < self.goal:

                        steps += 1

                        if(self.state[sy][sx] == 'X'):
                            player_x += 1
```

```
                        elif(self.state[sy][sx] == 'O'):
                            player_o += 1

                        sy += dy
                        sx += dx


                        if(sy < 0 or abs(sy) > self.max_row_index or sx < 0 or
abs(sx) > self.max_columns_index):
                            break

                    if(player_x >= self.goal):
                        return 'X'
                    elif(player_o >= self.goal):
                        return 'O'

        return None

    def min(self, alpha:int=-sys.maxsize, beta:int=sys.maxsize, depth:int=0):


        min_value = sys.maxsize
        by = None
        bx = None


        result = self.game_ended()
        if(result != None):
            if result == 'X':
                return 1, 0, 0, depth
            elif result == 'O':
                return -1, 0, 0, depth
            elif result == 'It is a tie!':
                return 0, 0, 0, depth
        elif(depth > self.max_depth):
            return 0, 0, 0, depth
```

```
        for y in range(self.rows):
            for x in range(self.columns):

                if (self.state[y][x] == '.'):

                    self.state[y][x] = 'O'

                    max, max_y, max_x, depth = self.max(alpha, beta, depth + 1
)



                    if (max < min_value):
                        min_value = max
                        by = y
                        bx = x


                    self.state[y][x] = '.'

                    if (min_value <= alpha):
                        return min_value, bx, by, depth

                    if (min_value < beta):
                        beta = min_value

        return min_value, by, bx, depth

    def max(self, alpha:int=-sys.maxsize, beta:int=sys.maxsize, depth:int=0):
        # Variables
        max_value = -sys.maxsize
        by = None
        bx = None

        result = self.game_ended()
        if(result != None):
            if result == 'X':
                return 1, 0, 0, depth
```

```
            elif result == 'O':
                return -1, 0, 0, depth
            elif result == 'It is a tie!':
                return 0, 0, 0, depth
        elif(depth > self.max_depth):
            return 0, 0, 0, depth

        for y in range(self.rows):
            for x in range(self.columns):

                if (self.state[y][x] == '.'):



                    self.state[y][x] = 'X'

                    min, min_y, min_x, depth = self.min(alpha, beta, depth + 1
)

                    if (min > max_value):
                        max_value = min
                        by = y
                        bx = x

                    self.state[y][x] = '.'

                    if (max_value >= beta):
                        return max_value, bx, by, depth

                    if (max_value > alpha):
                        alpha = max_value

        return max_value, by, bx, depth

    def print_state(self):
        for y in range(self.rows):
            print('| ', end='')
            for x in range(self.columns):
```

```
                if (self.state[y][x] != '.'):
                    print(' {0}  | '.format(self.state[y][x]), end='')
                else:
                    digit = str(self.inverted_tiles.get((y,x))) if len(str(sel
f.inverted_tiles.get((y,x)))) > 1 else ' ' + str(self.inverted_tiles.get((y,x)
))
                    print('{0}  | '.format(digit), end='')
            print()
        print()


if __name__ == "__main__":
    game = TicTacToeGame(3,3,3,1000)
    game.play()
```

Output Screenshots:

```
Depth: 104
| O  |  X  |  3  |
| X  |  X  |  6  |
| 7  |  O  |  9  |

Player O moving (Human) ...
Depth: 32
Recommendation: 6
Make a move (tile number): 6
| O  |  X  |  3  |
| X  |  X  |  O  |
| 7  |  O  |  9  |

Player X moving (AI) ...
Depth: 11
| O  |  X  |  X  |
| X  |  X  |  O  |
| 7  |  O  |  9  |

Player O moving (Human) ...
Depth: 4
Recommendation: 7
Make a move (tile number): 7
| O  |  X  |  X  |
| X  |  X  |  O  |
| O  |  O  |  9  |

Player X moving (AI) ...
Depth: 1
| O  |  X  |  X  |
| X  |  X  |  O  |
| O  |  O  |  X  |

Result: It is a tie!
```

Result: We have done tic tac toe using min max algorithm technique and output is printed in Collaboratory.

EX NO: 7

DATE: 12/4/21

Implementation of unification and resolution for real world problems

Problem Statement:

To implement the resolution and unification using aws console and print the result for the same.

Tools Used: AWS

Algorithm:

1. If Ψ_1 or Ψ_2 is a variable or constant, then:

    a) If Ψ_1 or Ψ_2 are identical, then return NIL.

    b) Else if $\Psi_1$is a variable,

        a. then if Ψ_1 occurs in Ψ_2, then return FAILURE

        b. Else return { (Ψ_2/ Ψ_1)}.

    c) Else if Ψ_2 is a variable,

        a. If Ψ_2 occurs in Ψ_1 then return FAILURE,

        b. Else return {(Ψ_1/ Ψ_2)}.

    d) Else return FAILURE.

2: If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, then return FAILURE.

3: IF Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE.

4: Set Substitution set(SUBST) to NIL.

5: For i=1 to the number of elements in Ψ_1.

a) Call Unify function with the ith element of Ψ_1 and ith element of Ψ_2, and put the result into S.

b) If S = failure then returns Failure

c) If S ≠ NIL then do,

a. Apply S to the remainder of both L1 and L2.

b. SUBST= APPEND(S, SUBST).

6: Return SUBST.

7.Conversion of facts into first-order logic.

8.Convert FOL statements into CNF

9.Negate the statement which needs to prove (proof by contradiction)

10.Print resolution (unification).

Optimization Technique:

1.Initialize the substitution set to be empty.

2. Recursively unify atomic sentences:

a. Check for Identical expression match.

b. If one expression is a variable v_i, and the other is a term t_i which does not contain variable v_i, then:

   a. Substitute t_i / v_i in the existing substitutions

   b. Add t_i /v_i to the substitution setlist.

   c. If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

Programming code:

Unification:

```
def get_index_comma(string):
    """
    Return index of commas in string
    """

    index_list = list()
    # Count open parentheses
    par_count = 0

    for i in range(len(string)):
        if string[i] == ',' and par_count == 0:
```

```
            index_list.append(i)
        elif string[i] == '(':
            par_count += 1
        elif string[i] == ')':
            par_count -= 1

    return index_list


def is_variable(expr):
    """
    Check if expression is variable
    """

    for i in expr:
        if i == '(' or i == ')':
            return False

    return True


def process_expression(expr):
    """
    input:  - expression:
            'Q(a, g(x, b), f(y))'
    return: - predicate symbol:
            Q
            - list of arguments
            ['a', 'g(x, b)', 'f(y)']
    """

    # Remove space in expression
    expr = expr.replace(' ', '')

    # Find the first index == '('
    index = None
    for i in range(len(expr)):
        if expr[i] == '(':
            index = i
            break

    # Return predicate symbol and remove predicate symbol in expression
    predicate_symbol = expr[:index]
    expr = expr.replace(predicate_symbol, '')

    # Remove '(' in the first index and ')' in the last index
    expr = expr[1:len(expr) - 1]
```

```
    # List of arguments
    arg_list = list()

    # Split string with commas, return list of arguments
    indices = get_index_comma(expr)

    if len(indices) == 0:
        arg_list.append(expr)
    else:
        arg_list.append(expr[:indices[0]])
        for i, j in zip(indices, indices[1:]):
            arg_list.append(expr[i + 1:j])
        arg_list.append(expr[indices[len(indices) - 1] + 1:])

    return predicate_symbol, arg_list


def get_arg_list(expr):
    """
    input:  expression:
            'Q(a, g(x, b), f(y))'
    return: full list of arguments:
            ['a', 'x', 'b', 'y']
    """

    _, arg_list = process_expression(expr)

    flag = True
    while flag:
        flag = False

        for i in arg_list:
            if not is_variable(i):
                flag = True
                _, tmp = process_expression(i)
                for j in tmp:
                    if j not in arg_list:
                        arg_list.append(j)
                arg_list.remove(i)

    return arg_list


def check_occurs(var, expr):
    """
    Check if var occurs in expr
    """
```

```
    arg_list = get_arg_list(expr)
    if var in arg_list:
        return True

    return False


def unify(expr1, expr2):

    # Step 1:
    if is_variable(expr1) and is_variable(expr2):
        if expr1 == expr2:
            return 'Null'
        else:
            return False
    elif is_variable(expr1) and not is_variable(expr2):
        if check_occurs(expr1, expr2):
            return False
        else:
            tmp = str(expr2) + '/' + str(expr1)
            return tmp
    elif not is_variable(expr1) and is_variable(expr2):
        if check_occurs(expr2, expr1):
            return False
        else:
            tmp = str(expr1) + '/' + str(expr2)
            return tmp
    else:
        predicate_symbol_1, arg_list_1 = process_expression(expr1)
        predicate_symbol_2, arg_list_2 = process_expression(expr2)

        # Step 2
        if predicate_symbol_1 != predicate_symbol_2:
            return False
        # Step 3
        elif len(arg_list_1) != len(arg_list_2):
            return False
        else:
            # Step 4: Create substitution list
            sub_list = list()

            # Step 5:
            for i in range(len(arg_list_1)):
                tmp = unify(arg_list_1[i], arg_list_2[i])

                if not tmp:
                    return False
                elif tmp == 'Null':
```

```
                    pass
                else:
                    if type(tmp) == list:
                        for j in tmp:
                            sub_list.append(j)
                    else:
                        sub_list.append(tmp)

            # Step 6
            return sub_list


if __name__ == '__main__':

    f1 = 'Q(a, g(x, a), f(y))'
    f2 = 'Q(a, g(f(b), a), x)'

    result = unify(f1, f2)
    if not result:
        print('Unification failed!')
    else:
        print('Unification successful!')
        print(result)
```

Resolution:

```
import copy
import time

class Parameter:
    variable_count = 1

    def __init__(self, name=None):
        if name:
            self.type = "Constant"
            self.name = name
        else:
            self.type = "Variable"
            self.name = "v" + str(Parameter.variable_count)
            Parameter.variable_count += 1

    def isConstant(self):
        return self.type == "Constant"

    def unify(self, type_, name):
        self.type = type_
        self.name = name
```

```
    def __eq__(self, other):
        return self.name == other.name

    def __str__(self):
        return self.name

class Predicate:
    def __init__(self, name, params):
        self.name = name
        self.params = params

    def __eq__(self, other):
        return self.name == other.name and all(a == b for a, b in zip(self.par
ams, other.params))

    def __str__(self):
        return self.name + "(" + ",".join(str(x) for x in self.params) + ")"

    def getNegatedPredicate(self):
        return Predicate(negatePredicate(self.name), self.params)


class Sentence:
    sentence_count = 0

    def __init__(self, string):
        self.sentence_index = Sentence.sentence_count
        Sentence.sentence_count += 1
        self.predicates = []
        self.variable_map = {}
        local = {}

        for predicate in string.split("|"):
            name = predicate[:predicate.find("(")]
            params = []

            for param in predicate[predicate.find("(") + 1: predicate.find(")"
)].split(","):
                if param[0].islower():
                    if param not in local:  # Variable
                        local[param] = Parameter()
                        self.variable_map[local[param].name] = local[param]
                    new_param = local[param]
                else:
                    new_param = Parameter(param)
                    self.variable_map[param] = new_param
```

```
                params.append(new_param)

            self.predicates.append(Predicate(name, params))

    def getPredicates(self):
        return [predicate.name for predicate in self.predicates]

    def findPredicates(self, name):
        return [predicate for predicate in self.predicates if predicate.name =
= name]

    def removePredicate(self, predicate):
        self.predicates.remove(predicate)
        for key, val in self.variable_map.items():
            if not val:
                self.variable_map.pop(key)

    def containsVariable(self):
        return any(not param.isConstant() for param in self.variable_map.value
s())

    def __eq__(self, other):
        if len(self.predicates) == 1 and self.predicates[0] == other:
            return True
        return False

    def __str__(self):
        return "".join([str(predicate) for predicate in self.predicates])

class KB:
    def __init__(self, inputSentences):
        self.inputSentences = [x.replace(" ", "") for x in inputSentences]
        self.sentences = []
        self.sentence_map = {}

    def prepareKB(self):
        self.convertSentencesToCNF()
        for sentence_string in self.inputSentences:
            sentence = Sentence(sentence_string)
            for predicate in sentence.getPredicates():
                self.sentence_map[predicate] = self.sentence_map.get(predicate
, []) + [sentence]

    def convertSentencesToCNF(self):
        for sentenceIdx in range(len(self.inputSentences)):
            if "=>" in self.inputSentences[sentenceIdx]:  # Do negation of the
 Premise and add them as literal
```

```
                self.inputSentences[sentenceIdx] = negateAntecedent(self.input
Sentences[sentenceIdx])

    def askQueries(self, queryList):
        results = []

        for query in queryList:
            negatedQuery = Sentence(negatePredicate(query.replace(" ", "")))
            negatedPredicate = negatedQuery.predicates[0]
            prev_sentence_map = copy.deepcopy(self.sentence_map)
            self.sentence_map[negatedPredicate.name] = self.sentence_map.get(n
egatedPredicate.name, []) + [negatedQuery]
            self.timeLimit = time.time() + 40

            try:
                result = self.resolve([negatedPredicate], [False]*(len(self.in
putSentences) + 1))
            except:
                result = False

            self.sentence_map = prev_sentence_map

            if result:
                results.append("TRUE")
            else:
                results.append("FALSE")

        return results

    def resolve(self, queryStack, visited, depth=0):
        if time.time() > self.timeLimit:
            raise Exception
        if queryStack:
            query = queryStack.pop(-1)
            negatedQuery = query.getNegatedPredicate()
            queryPredicateName = negatedQuery.name
            if queryPredicateName not in self.sentence_map:
                return False
            else:
                queryPredicate = negatedQuery
                for kb_sentence in self.sentence_map[queryPredicateName]:
                    if not visited[kb_sentence.sentence_index]:
                        for kbPredicate in kb_sentence.findPredicates(queryPre
dicateName):

                            canUnify, substitution = performUnification(copy.d
eepcopy(queryPredicate), copy.deepcopy(kbPredicate))
```

```
                            if canUnify:
                                newSentence = copy.deepcopy(kb_sentence)
                                newSentence.removePredicate(kbPredicate)
                                newQueryStack = copy.deepcopy(queryStack)

                                if substitution:
                                    for old, new in substitution.items():
                                        if old in newSentence.variable_map:
                                            parameter = newSentence.variable_m
ap[old]
                                            newSentence.variable_map.pop(old)
                                            parameter.unify("Variable" if new[
0].islower() else "Constant", new)
                                            newSentence.variable_map[new] = pa
rameter

                                    for predicate in newQueryStack:
                                        for index, param in enumerate(predicat
e.params):
                                            if param.name in substitution:
                                                new = substitution[param.name]
                                                predicate.params[index].unify(
"Variable" if new[0].islower() else "Constant", new)

                                for predicate in newSentence.predicates:
                                    newQueryStack.append(predicate)

                                new_visited = copy.deepcopy(visited)
                                if kb_sentence.containsVariable() and len(kb_s
entence.predicates) > 1:
                                    new_visited[kb_sentence.sentence_index] =
True

                                if self.resolve(newQueryStack, new_visited, de
pth + 1):
                                    return True
                return False
        return True


def performUnification(queryPredicate, kbPredicate):
    substitution = {}
    if queryPredicate == kbPredicate:
        return True, {}
    else:
        for query, kb in zip(queryPredicate.params, kbPredicate.params):
            if query == kb:
                continue
```

```
            if kb.isConstant():
                if not query.isConstant():
                    if query.name not in substitution:
                        substitution[query.name] = kb.name
                    elif substitution[query.name] != kb.name:
                        return False, {}
                    query.unify("Constant", kb.name)
                else:
                    return False, {}
            else:
                if not query.isConstant():
                    if kb.name not in substitution:
                        substitution[kb.name] = query.name
                    elif substitution[kb.name] != query.name:
                        return False, {}
                    kb.unify("Variable", query.name)
                else:
                    if kb.name not in substitution:
                        substitution[kb.name] = query.name
                    elif substitution[kb.name] != query.name:
                        return False, {}
    return True, substitution


def negatePredicate(predicate):
    return predicate[1:] if predicate[0] == "~" else "~" + predicate


def negateAntecedent(sentence):
    antecedent = sentence[:sentence.find("=>")]
    premise = []

    for predicate in antecedent.split("&"):
        premise.append(negatePredicate(predicate))

    premise.append(sentence[sentence.find("=>") + 2:])
    return "|".join(premise)


def getInput(filename):
    with open(filename, "r") as file:
        noOfQueries = int(file.readline().strip())
        inputQueries = [file.readline().strip() for _ in range(noOfQueries)]
        noOfSentences = int(file.readline().strip())
        inputSentences = [file.readline().strip() for _ in range(noOfSentences
)]
        return inputQueries, inputSentences
```

```
def printOutput(filename, results):
    print(results)
    with open(filename, "w") as file:
        for line in results:
            file.write(line)
            file.write("\n")
    file.close()


if __name__ == '__main__':
    inputQueries_, inputSentences_ = getInput("input.txt")
    knowledgeBase = KB(inputSentences_)
    knowledgeBase.prepareKB()
    results_ = knowledgeBase.askQueries(inputQueries_)
    printOutput("output.txt", results_)
```

Input:

```
3
Sick(Alice,Flu)
Sick(Bob,Shingles)
Sick(Cate,Shingles)
7
Sick(x,a) & Contagious(a) & Contact(x,y) => Sick(y,a)
Contagious(Flu)
~Contagious(Shingles)
Sick(Bob,Flu)
Sick(Alice,Shingles)
~Contact(Bob,Alice)
Contact(Alice,Cate)
```

Output:

Unification:

```
Vaishnavimoorthy:~/environment $ cd RA1811028010041
Vaishnavimoorthy:~/environment/RA1811028010041 $ python3 unification.py
Unification successful!
['f(b)/x', 'f(y)/x']
Vaishnavimoorthy:~/environment/RA1811028010041 $
```

Resolution:

```
Vaishnavimoorthy:~/environment/RA1811028010041 $ python3 resolution.py
['FALSE', 'FALSE', 'FALSE']
Vaishnavimoorthy:~/environment/RA1811028010041 $
```

Result: We have successfully implemented resolution and unification problem.

EX NO: 8

DATE: 21/4/21

Implementation of knowledge representation schemes – use cases

Problem Statement:

To implement the knowledge representation schemes using test cases.

Tools Used: AWS

Algorithm:

1. In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, so generating the conclusions from evidence and facts is termed as Inference.
2. Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.
3. In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:
4. Implication: It is one of the logical connectives which can be represented as $P \rightarrow Q$. It is a Boolean expression.
5. Converse: The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as $Q \rightarrow P$.
6. Contrapositive: The negation of converse is termed as contrapositive, and it can be represented as $\neg Q \rightarrow \neg P$.
7. Inverse: The negation of implication is called inverse. It can be represented as $\neg P \rightarrow \neg Q$.

Optimization Technique:

1. Logical Representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. This representation lays down some important communication rules. It consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

2. Semantic Network Representation

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

3. Frame Representation

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.

4. Production Rules

Production rules system consist of (condition, action) pairs which mean, "If condition then action". It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

Programming Code:

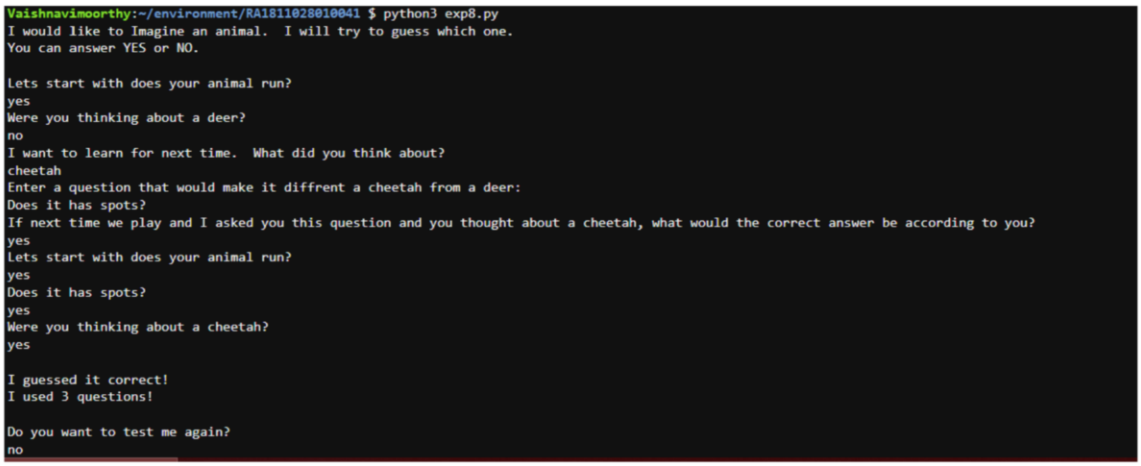
```
import sys

def definiteNoun(s):
  "Add definite form to a noune, for instance 'whale' becomes 'a whale'"
  s = s.lower().strip()
  if s in ['a', 'e', 'i', 'o', 'u', 'y']:
    return "an " + s
  else:
    return "a " + s

def removeArticle(s):
  "Remove the definite article 'a' or 'an' from a noun."
  s = s.lower().strip()
  if s[0:3] == "an ": return s[3:]
  if s[0:2] == "a ": return s[2:]
  return s

def makeQuestion(question, yes, no):
  return [question, yes, no]

def isQuestion(p):
  "Check if node is a question (with answers), or a plain answer."
  return type(p).__name__ == "list"

def askQuestion(question):
  print ("\r%s " % question,)
  return sys.stdin.readline().strip().lower()

def getAnswer(question):
  if isQuestion(question):
    return askQuestion(question[0])
  else:
    return askQuestion("Were you thinking about %s?" % definiteNoun(question))

def answeredYes(answer):
  if len(answer) > 0:
    return answer.lower()[0] == "y"
  return False

def gameOver(message):
  global tries
  print ("")
  print ("\r%s" % message)
  print ("I used", tries, "questions!")
  print ("")
```

```
def playAgain():
  return answeredYes(askQuestion("Do you want to test me again?"))

def correctGuess(message):
  global tries
  gameOver(message)

  if playAgain():
    print ("")
    tries = 0
    return Q
  else:
    sys.exit(0)

def nextQuestion(question, answer):
  global tries
  tries += 1

  if isQuestion(question):
    if answer:
      return question[1]
    else:
      return question[2]
  else:
    if answer:
      return correctGuess("I guessed it correct!")
    else:
      return makeNewQuestion(question)

def replaceAnswer(tree, find, replace):
  if not isQuestion(tree):
    if tree == find:
      return replace
    else:
      return tree
  else:
    return makeQuestion(tree[0],
      replaceAnswer(tree[1], find, replace),
      replaceAnswer(tree[2], find, replace))

def makeNewQuestion(wrongAnimal):
  global Q, tries

  correctAnimal = removeArticle(askQuestion("I want to learn for next time.  W
hat did you think about?"))

  newQuestion = askQuestion("Enter a question that would make it diffrent %s f
rom %s:"
```

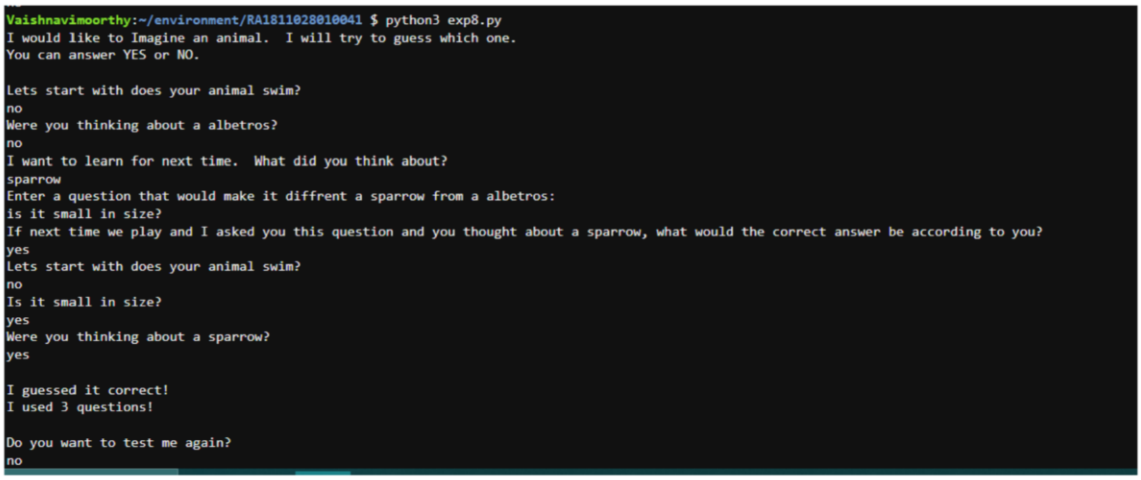
```
      % (definiteNoun(correctAnimal), definiteNoun(wrongAnimal))).capitalize()

  yesAnswer = answeredYes(askQuestion("If next time we play and I asked you th
is question " +
    "and you thought about %s, what would the correct answer be according to y
ou?" % definiteNoun(correctAnimal)))

  # Create new question node
  if yesAnswer:
    q = makeQuestion(newQuestion, correctAnimal, wrongAnimal)
  else:
    q = makeQuestion(newQuestion, wrongAnimal, correctAnimal)

  # Create new question tree and start over again
  Q = replaceAnswer(Q, wrongAnimal, q)
  tries = 0 # reset since we'll play again
  return Q

tries = 0
Q = (makeQuestion('Lets start with does your animal run?', 'deer', 'albetros')
)
q = Q

print ("I would like to Imagine an animal.  I will try to guess which one.")
print ("You can answer YES or NO.")
print ("")

try:
  while True:
    ans = answeredYes(getAnswer(q))
    q = nextQuestion(q, ans)
except KeyboardInterrupt:
  sys.exit(0)
except Exception:
  print (e)
  sys.exit(1)
```

Output:

TEST CASE 1:



TEST CASE 2:



Result: We have successfully implemented knowledge representation schemes – use cases and output is received in AWS.

EX NO: 9

DATE: 27/4/21

Implementation of uncertain methods for an application

Problem Statement:

To implement Fuzzy logic using matplotlib in python and find the graph of temperature, humidity and speed in different conditions.

Algorithm:

1. Locate the input, output, and state variables of the plane under consideration. I
2. Split the complete universe of discourse spanned by each variable into a number of fuzzy subsets, assigning each with a linguistic label. The subsets include all the elements in the universe.
3. Obtain the membership function for each fuzzy subset.
4. Assign the fuzzy relationships between the inputs or states of fuzzy subsets on one side and the output of fuzzy subsets on the other side, thereby forming the rule base.
5. Choose appropriate scaling factors for the input and output variables for normalizing the variables between [0, 1] and [-1, I] interval.
6. Carry out the fuzzification process.
7. Identify the output contributed from each rule using fuzzy approximate reasoning.
8. Combine the fuzzy outputs obtained from each rule.
9. Finally, apply defuzzification to form a crisp output.

Optimization Technique:

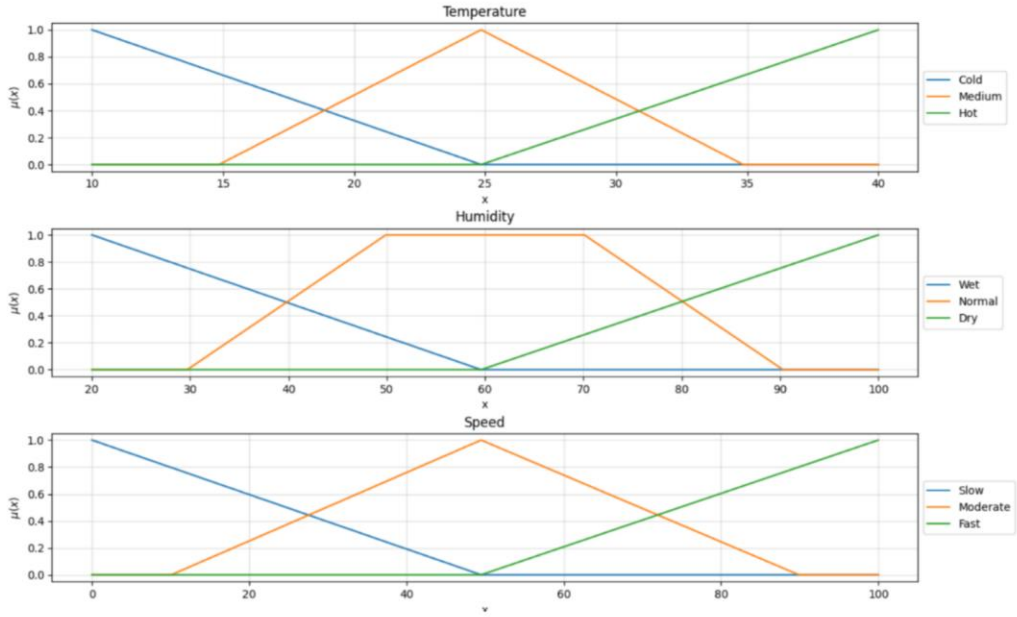
1. Decomposing the large-scale system into a collection of various subsystems.
2. Varying the plant dynamics slowly and linearizing the nonlinear plane dynamics about a set of operating points.
3. Organizing a set of state variables, control variables, or output features for the system under consideration.
4. Designing simple P, PD, PID controllers for the subsystems. Optimal controllers can also be designed.

Programming Code:

```
from fuzzy_system.fuzzy_variable_output import FuzzyOutputVariable
from fuzzy_system.fuzzy_variable_input import FuzzyInputVariable
# from fuzzy_system.fuzzy_variable import FuzzyVariable
from fuzzy_system.fuzzy_system import FuzzySystem

temp = FuzzyInputVariable('Temperature', 10, 40, 100)
temp.add_triangular('Cold', 10, 10, 25)
temp.add_triangular('Medium', 15, 25, 35)
temp.add_triangular('Hot', 25, 40, 40)

humidity = FuzzyInputVariable('Humidity', 20, 100, 100)
humidity.add_triangular('Wet', 20, 20, 60)
humidity.add_trapezoidal('Normal', 30, 50, 70, 90)
humidity.add_triangular('Dry', 60, 100, 100)

motor_speed = FuzzyOutputVariable('Speed', 0, 100, 100)
motor_speed.add_triangular('Slow', 0, 0, 50)
motor_speed.add_triangular('Moderate', 10, 50, 90)
motor_speed.add_triangular('Fast', 50, 100, 100)

system = FuzzySystem()
system.add_input_variable(temp)
system.add_input_variable(humidity)
system.add_output_variable(motor_speed)

system.add_rule(
        { 'Temperature':'Cold',
            'Humidity':'Wet' },
        { 'Speed':'Slow'})

system.add_rule(
        { 'Temperature':'Cold',
            'Humidity':'Normal' },
        { 'Speed':'Slow'})

system.add_rule(
        { 'Temperature':'Medium',
            'Humidity':'Wet' },
        { 'Speed':'Slow'})

system.add_rule(
        { 'Temperature':'Medium',
            'Humidity':'Normal' },
        { 'Speed':'Moderate'})

system.add_rule(
```

```
        { 'Temperature':'Cold',
            'Humidity':'Dry' },
        { 'Speed':'Moderate'})

system.add_rule(
        { 'Temperature':'Hot',
            'Humidity':'Wet' },
        { 'Speed':'Moderate'})

system.add_rule(
        { 'Temperature':'Hot',
            'Humidity':'Normal' },
        { 'Speed':'Fast'})

system.add_rule(
        { 'Temperature':'Hot',
            'Humidity':'Dry' },
        { 'Speed':'Fast'})

system.add_rule(
        { 'Temperature':'Medium',
            'Humidity':'Dry' },
        { 'Speed':'Fast'})

output = system.evaluate_output({
                'Temperature':18,
                'Humidity':60
        })

print(output)
# print('fuzzification\n-------------\n', info['fuzzification'])
# print('rules\n-----\n', info['rules'])

system.plot_system()
```

Output:



Result: We have successfully implemented fuzzy uncertainty problem using matplotlib and output is received.

EX NO: 10

DATE: 11/3/21

Implementation of block world problem

Problem Statement:

To implement the block world problem using correct artificial intelligence optimization techniques.

Tools Used: Google Collaboratory

Algorithm:

1. Initialise a stack to store the blocks.
2. Make sure the stack is empty when HEAD NODE.NEXT = NULL
3. Read the pattern of blocks given label it START STATE
4. Compare the given pattern to the given final pattern label it GOAL STATE
5. Now start the movement of the blocks one by on either one on top or to the floor according to the need.
6. Keep recording these movements in the empty stack created by STACK.PUSH and STACK.POP methods.
7. Stop the block manipulation when goal state is reached.

Optimization Technique:

1. We will move block 1 to next row or stack at initial state 0
2. Block 5 will move to 3rd row
3. Block 2 will move to top of 1 and block 4 will move to new row then 3 will move to top of 2 making row in order of 3->2->1
4. Finally block will move to top of 4 and both will move to top of 3 reaching final state of 5->4->3->2->1

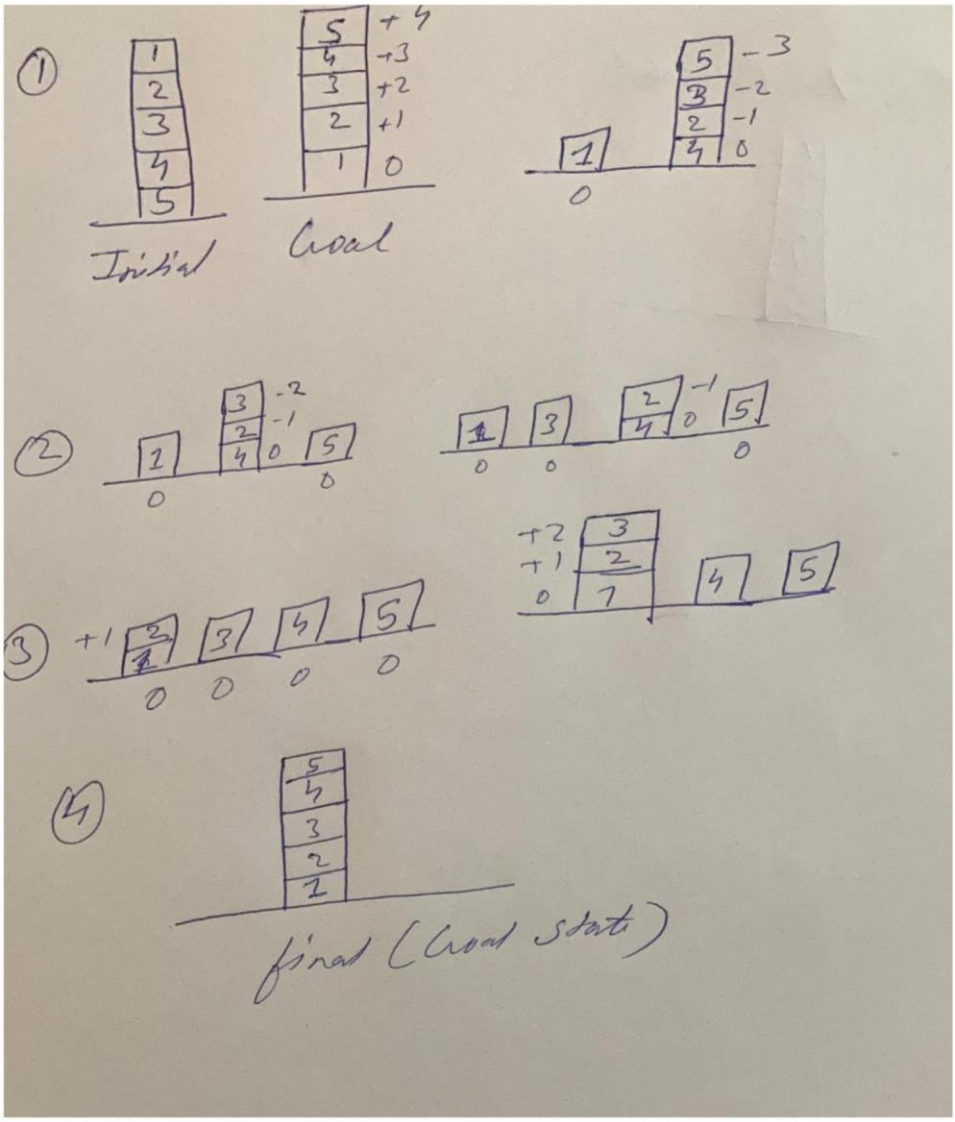


Fig 10.1 Optimization

Programming Code:

```
import time
class Node:

    def __init__(self, data=None):
        self.data = data
        self.next = None

class Stack:

    def __init__(self):

        print("Blocks created")
        self.stack_pointer = None

    def push(self, x):
        if not isinstance(x, Node):
            x = Node(x)
        print(f"Adding {x.data} to the row")
        if self.is_empty():
            self.stack_pointer = x
        else:
            x.next = self.stack_pointer
            self.stack_pointer = x

    def pop(self):

        if not self.is_empty():
            print(f"Removing Block on top of Block")
            curr = self.stack_pointer
            self.stack_pointer = self.stack_pointer.next
            curr.next = None
            return curr.data
        else:
            return "Block pos is empty"

    def is_empty(self):
```

```
        return self.stack_pointer == None

    def peek(self):

        if not self.is_empty():
            return self.stack_pointer.data

    def __str__(self):
        print("Printing Block state...")
        to_print = ""
        curr = self.stack_pointer
        while curr is not None:
            to_print += str(curr.data) + "->"
            curr = curr.next
        if to_print:
            print("Block Pointer")
            print(" |")
            print(" V")
            return "[" + to_print[:-2] + "]"
        return "[]"

print ("INITIAL STATE : {[1], [2], [3], [4], [5]}")
print("-"*70)
print ("FINAL STATE :[4->3->2->1]")
my_stack = Stack()
print("Checking if Block pos is empty:", my_stack.is_empty())
my_stack.push(1)
time.sleep(1)
my_stack.push(2)
print(my_stack)
time.sleep(1)
my_stack.push(3)
time.sleep(1)
my_stack.push(4)
time.sleep(1)
print("Checking Blocks:", my_stack.peek())
time.sleep(1)
```

```
my_stack.push(5)
print(my_stack)
time.sleep(1)
print(my_stack.pop())
time.sleep(1)
print(my_stack.pop())
print(my_stack)
time.sleep(1)
my_stack.push(4)
print(my_stack)
time.sleep(1)
```

Output:

```
INITIAL STATE : {1->2->3->4->5}
--------------------------------------------------------------------------------
FINAL STATE :[5->4->3->2->1]
Block created
Checking if stack is empty: True
Adding 1 to the row
Adding 2 to the row
Printing block state...
block Pointer
 |
 V
[2->1]
Adding 3 to the row
Adding 4 to the row
Checking item on row: 4
Adding 5 to the row
Printing block state...
block Pointer
 |
 V
[5->4->3->2->1]
Removing Block on top of Block
5
Removing Block on top of Block
4
Printing block state...
block Pointer
 |
 V
[3->2->1]
Adding 4 to the row
Printing block state...
block Pointer
 |
 V
[4->3->2->1]
Adding 5 to the top of block
block Pointer
 |
 V
 5
FINAL STATE :[5->4->3->2->1]
```

Result: We have successfully implemented block world problem using stacks and searching algorithm and output is received in google Collaboratory.