INDEX

NAME: Manoj Patil STD.: 5th sem SEC.: B ROLL NO.: CS104

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	17/11/23	Tic-Tac-toe using python		
2	24/11/23	8 puzzle problem (A*)		
3	8/12/23	8 puzzle problem (BFS)		
4	8/12/23	8 puzzle problem (IDDFS)		
5	22/12/23	Vaccum cleaner		
6	28/12/23	Knowledge Base Entailment		
7	28/12/23	Knowledge Base Resolution		
8	12/1/24	Unification		
9	12/1/24	FOL to CNF		
10	23/1/24	Forward chaining		24/1/24

Lab - 1

Python :-

Topics learnt :-

1) Introduction
2) Functions and getting Help
3) Booleans and conditionals
4) Lists
5) Loops and list comprehensions
6) strings and dictionaries
7) working with external libraries

Lab-2

Bafna Gold
Date: Page:

1) Program for age categorisation :-

```
def agecriteria(age):
    if (age <= 2):
        print("Infants")
    elif (age <=12):
        print("kids")
    elif (age <= 20):
        print("Teens")
    elif (age <= 59):
        print("Adults")
    else:
        print("senior citizens")

age =int(input("Enter the age \n"))
agecriteria(age)
```

output
enter the age
75
senior citizens

2) Program for multiplication tables :-

```
def multitable(num, k):
    for i in range(k):
        ans = num*(i+1)
        print("{num}* {i+1}= {ans}\n")
table = int(input('Enter the number\n'))
k = int(input('till which number\n'))
multitable(table, k)
```

output

Enter the number
5

Enter till which numbe
5

5

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

3) Program for bubble sort.

```
def bubble(list_item):
    for i in range(len(list_item)):
        for j in range(i+1, len(list_item)):
            if(list_item[i] > list_item[j]):
                temp = list_item[i]
                list_item[i] = list_item[j]
                list_item[j] = temp
    return list_item
k = int(input("Enter the size"))
```

```
list_item = []
for j in range (k):
        list_item.append (int(input()))
list_item = bubble_sort(list_item)
print(list_item)
```

Output :-

Enter the size : 4
Enter the elements : 3 . 8 1

[1, 3, 8]

4). Reverse of a number :-

```
num = int (input ("Enter the number \n"))
  reversed_num = 0
while num != 0:
    digit = num % 10
    reversed_sum = reversed_sum * 10 + digit
    num // 10
print ("reversed number : %d" %(reversed_num))
```

Output :-
number 4223
reversed number : 8224 :

Tic Tac Toe

1).

for creating a blank board.

```
import numpy as np
import random

def create():
    return(np.array([[0,0,0],
                     [0,0,0],
                     [0,0,0]]))

def posi(board):
    l = []
    for i in range(len(board)):
        for j in range(len(board)):
            if board[i][j] == 0:
                l.append((i,j))
    return(l)

def randomplace(board, player):
    selection = possibilities(board)
    cur = random.choice(selection)
    board[cur] = player
    return board
```

```
def rowwin(board, player)
    for x in range(len(board)):
        win = true
        for y in range(len(board)):
            if board[x,y] != player:
                win = false
                continue
        if win == true:
            return(win)
    return(win)

def colwin(board, player)
    for x in range(len(board)):
        win = true
        for y in range(len(board)):
            if board[y][x] != player
                win = false
                continue
        if win == false
            return(win)
    return(win)
```

8 puzzle problem using bfs

In the problem, the squares will have N+1 tiles where N = 8, 15, 24 and so on

N=8 means the square will have 9 tiles (3 rows and 3 columns)

In this problem, initial state/initial configuration (start state) will be given and we have to reach the goal state

Suppose

Initial state :-

1	2	3
4	6	
7	5	8

goal state :-

1	2	3
4	5	6
7	8	

The puzzle can be ~~reached~~ solved by moving the tiles one by one in the style empty space and this achieve the goal state

rules :-

The empty can only move in 4 directions

1) up 2) down 3) right 4) left.

- It cannot be moved diagonally

can take only one step at a time.

Complexity :- $O(b^d)$ where

b - branching factor, d - depth factor

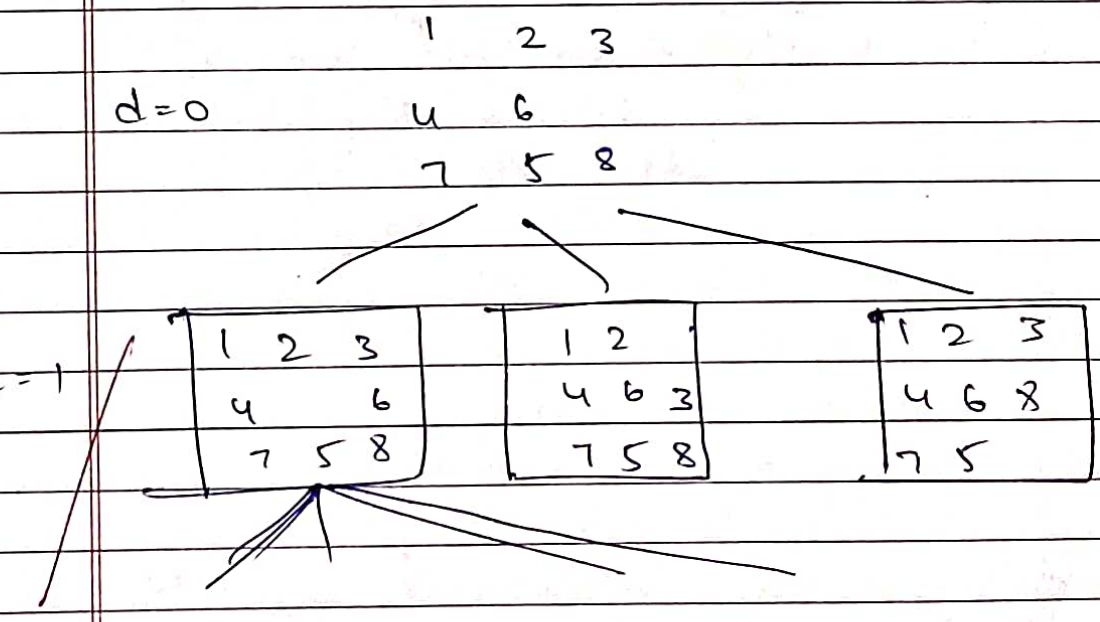
worst case — 3^{20}.

For 8 puzzle problem

Branching factor b = $\frac{\text{all possible moves}}{\text{no. of tiles}}$

$= \frac{24}{9} = 2.69 \sim 3.$

Example



Output :- solution found!

Number of moves : 2

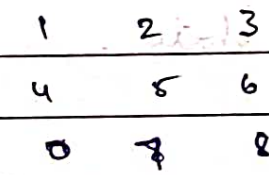
(0,1)

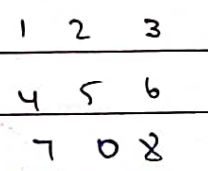
(0,1)

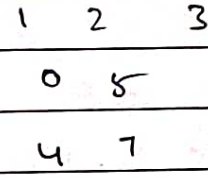
Bafna Gold
Date: Page:

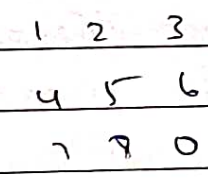
8 puzzle - iterative deepening search algorithms

Algorithm:-









→ goal

Algorithm

1) Initialize the initial state = [] and get goal state
2) set the depth = 1 and expand the initial state
The depth_limit_search(depth) is performed

```
if node.state = goal
    return node
else
    for neighbor in get_neighbors(node.state)
    child = puzzle_node(neighbor, node)
    result = depth_lim(depth+1)
if result = True;
    return true;
```

3) After one iteration where depth = 1, increment the depth by 1 and perform depth-limited search again

4. Here get_neighbour will generate the possible moves by swapping the '0' tile.

5. The path traversed is printed to reach the goal state.

Done
8.12.21

Bafna Gold
Date: Page:

Solving using 8-puzzle using A* algorithm

1	2	3
4	5	6
7	8	0

1	2	3	
0	4	6	h=4
7	5	8	

	1	2	3		0	2	3	1	2	3
(h=3)	4	0	6		1	4	6	7	4	6
	7	5	8		7	5	8	0	5	8

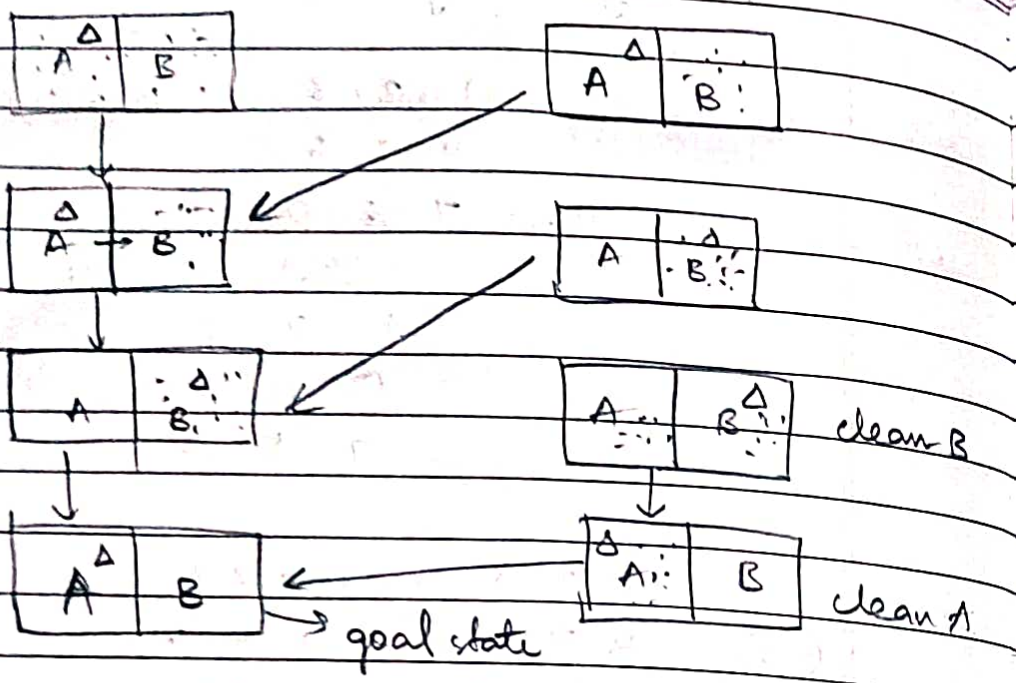
Algorithm

1) Create the initial state & goal state for the problem. In A* the heuristics is considered lower heuristic node is considered in each step
2) Initially expand the node, find the locations of empty tile & generate the node
3) Maintain two lists namely 'open' and 'close' The node (states) generated as stored in the open list, sort using the f(x) value
4) The goal is reached when h(x) = 0 implies that all the tiles are in the correct problem

8/12/24

22-11-23

Vaccum cleaner Agent



Algorithm :-

1) Initialize the starting and goal state, the goal ~~is~~ is to clean both rooms A & B
2) If status = dirty then clean.
   else if location = A & status = clean the return right
   else if location = B & status = clean then return left.
   else exit
3) If both the locations are clean then vaccum cleaner is done with its task

Proceed

code :-

```
def vacuum_world():
    goal_state = {'A': '0', 'B': '0'}
    cost = 0
    location_input = input("enter loc of vaccum")
    status_input = input("enter status")
    statcomp = input("enter status of room")
    if location_input == 'A':
        if status_input == '1':
            print("dirty")
            goalstate['A'] = '0'
            cost += 1
        print("cost for 'A'")

    else:
        print("It is in loc B")
        if status_input == '1':
            print("loc B is dirty")
            goal_state['B'] = '0'
            cost += 6
            print("cost " + str(cost))
    else:
        print(cost)
        print("loc B is already clean")
        if statcomp == '1':
            print("location A is dirty")
            cost += 1
    else:
        print("no action")
```

```
print ("goal state ")
print ("",(goal_state))
vaccum_world()
```

Output :

```
Enter location of vaccum D
Enter status of D
enter status of other room 1
initial location condition {'A': '0', 'B': '0'}
performance
    measure : 2
```

29/12/23

Entailment

Inputs:

knowledge base (set of logical rules)

steps:-

1. Negate the query: obtain the negation.
2. Combine with knowledge base:
3. Check satisfiability
   to check if the negation with KB is satisfying the rules
4. Determine entailment
   if conjunction is not satisfiable → True
   if conjunction is satisfiable → False.

Code:-

```
from sympy import symbols, And, Not, Implies,
                  satisfiable.

def createknbase():
    p = symbols('p')
    q = symbols('q')
    r = symbols('r')

    knbase = And(
      Implies(p,q)
      Implies(q,r)
       Not(r)
    )
    return knbase
```

```
def query_ent(kn_base, query):
    entail = satisfiable(And(kn_base, Not(query)))
    return not entailment

if __name__ == "__main__":
    KB = create_kn_base()
    query = symbols('not p')

    results = query.entails(KB, query)

    print("Knowledge base", kb)
    print("Query", query)
    print("Query entails Knowledge Base:", res_alf)
```

Output :-

Knowledge base : ~r & (Implies(p,q)) & (implies(q,r))
Query : (not, p)
Query entails knowledge base : False.

```
def negate_literal(literal)
    if literal[0] == 'n':
        return literal[1:]
    else:
        return 'n' + literal

def resolve(C1, C2)
    resolved_clause = set(C1) | set(C2)
    for literal in C1:
        if negate_literal(literal) in C2:
            resolved_clause.remove(literal)
            resolved_clause.remove(negate_literal)
    return tuple(resolved_clause)

def resolution(knowledge_base):
    while True:
        new clauses : set()
        for i, C1 in enumerate(knowledge_base)
            for j, C2 in enumerate(knowledge)
                if i != j:
                    new clause = resolve(C1, C2)

        if len(new_clause) > 0 & new clause not
                                    knowledge_base
            new_clause.add(new_clause)
        if not new_clause
            break
        knowledge_base = new_clause
    return knowledge_base
```

```
if __name__ == "__main__":
    kb = {('p', 'q'), ('¬p', 'r'), {'¬q', '¬r'}}
    result = resolution(kb)
    print("original kb", kb)
    print("resolved kb", result)
```

Output :-

Enter statement : negation
The statement not entiled by knowledge-base

Unification

Eg:- Knows (John, x) Knows (John, Jane)
{x/Jane}

Algorithm:-

step 1: If term1 or term2 is available or constant then:
a) term1 or term2 are identical
return NIL
b) Else if term1 is a variable if term1 occurs in term2
return FAIL
c) else if term2 is a variable if term2 occurs in term1
return FAIL
else
return < (term1/term2) }
d) else return FAIL
step 2:- if predicate(term1) ≠ predicate(term2)
return FAIL
step 3: number of argument ≠ return FAIL
step 4: set (SUB ST) to NIL
step 5: for i=1 to the number of elements in term 1
a) call unify (ith term1, ith term2)
put result into S
S = FAIL
return FAIL
b) if S ≠ NIL
a) apply S to the remainder of both C_1 & C_2

b) SUBST. append (s, subset)

step 6 :: return SUBST

19/1/24

Bafna Gold
Date: Page:

Program - 2

```
def getAttributes(string):
    expr = '\([^)]+\)'

def getPredicates(string):
    expr = '[a-z~]+\([A-z a-z,]+\)'

def DeMorgan(sentence):
    string = string.replace('~~', '')
    flag = '(' in string
    string = string.replace('~[', '(')
    string = string.strip(']')
    for predic in getpredicates(string):
        s = list(string)

def skolemization(sentence):
    skolem constants = [f'{chr(c)}' for
        c in range(ord('A'), ord('Z')+1)]
    statement = ''.join list(sentence).copy()
    matches = refindall('\[\[^]]+\]')
        statement
        for s in statements:
            statements = statement.replace(s, s[1:-1])
    for predicate in getpredicates(statement)
        attributes = getattributes(predicate)
```

```
if "".join(attributes).islower():
    statement = statement.replace(match(1)
                SOKLEM-CONSTANTS.pop(1))
else:
    aL = [a for a in attributes if a.islower()]
    aU = [a for a in attributes if not a.lower()][0]
    statement = statement.replace(match(1), SKOLEM
return statement
```

Output :-

[~american(x) | ~weapon(y) | ~sells(x,y,z) | ~hostile(z) | criminal(x).]

Output :-

[~animal(y) | loves(x,y)] & [~loves(x,y) | animal(y)].

```
print ( skolemization (fol_lof("
    [american(x) & weapon(y) & sells(x,y,z) &
        hostile(z)] ⇒ criminal(x)"))
```

Output :-

[~american(x) | ~weapon(y) | ~sells(x,y,z) | ~hostile(z) | criminal(x)]

Create a knowledge base consisting of first order logic statements & prove the given query using forward reasoning

Bafna Gold
Date: Page:

```
import re

def isVariable(x)
    return len(x)==0 and x.islower() and (x)not
        x.isalpha()

def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return matches

def getpredicates(string):
    expr = '[a-zA-Z]+)\([^&|]+)'
    return re.findall(expr, string)

class Fact:
    def __init__(self, expression):
        self.expression = expression
        predicate, params = self.split(expression)

    def split(self, expression):
        predicate = getPredicates(expression)[0]

    def getResult(self):
        return self.result
```

```
def getvariables(self):
    return (v) if isVariables(v)
    else None

class Implication:

    def evaluate(self, facts):
        constants = {}
        new_lhs = []
    for fact in facts:
        for val in self.lhs:
            if val.predicate == fact.predicate:
            for i, v in enumerate(val.getvariables()):
                if v:
                    constants[v] = fact.getconstants()[i]
        new_lhs.append(fact)

Class Kb:
    def __init__(self):
        self.facts = set()
        self.implications = set()

    def tell(self, e):
        if '⇒' in e
            self.implications.add('implication(e)')
```

```
def display(self):
    for i, f in enumerate(set([f.expression
                               for f in self.facts])):
        print(f'\t {i}. {f}')
```

```
kb = KB()
kb.tell('missile(x) => weapon(x)')
kb.tell('missile(M1)')
```

```
kb = KB()
kb.tell('king(x) & greedy(x) => evil(x)')
kb.tell('king(John)')
kb.tell('greedy(John)')
kb.tell('king(Richard)')
kb.query('evil(x)')
```

Output :-

Querying evil(x):

1. evil(Richard)
2. evil(John)

24/1/24