10/11/23

# LAB-0

① Write a python program to check age criteria using else-if ladder.

```
def check_age_criteria(age):
    if age <= 12:
        return "Child"
    elif 12 < age <= 17:
        return "Adolescent"
    elif 17 < age <= 30:
        return "Young Adults"
    elif 30 < age <= 50:
        return "Middle Aged"
    elif 50 < age <= 100:
        return "Old Aged"
    else:
        return "Invalid age"

age = int(input("Enter age:"))
result = check_age_criteria(age)
print(result)
```

=> Output:

Enter your age: 45
Middle Aged

Enter your age: 120
Invalid age

(2) Write a python program to print 1, 2,2, 3,3,3, 4,4,4,4 - -

```
def print_pattern(n):
    for i in range(1, n-1):
        for j in range(i):
            print(i, end=" ")
        print()
num = int(input("Enter an integer: "))
print_pattern(num)
```

→ Output:

```
Enter an integer: 5
1
22
333
4444
55555
```

(5) WAPP to print 1, 12, 123, 1234 ....

```
def print_pattern(n):
    for i in range(1, n+1):
        for j in range(1, i+1):
            print(j, end=" ")
        print()
num = int(input("Enter an integer:"))
print_pattern(num)
```

→ Output:

```
Enter an integer: 5
1
12
123
1234
12345
```

④ WAPP to print Multiplication table

```
def print_table(n):
    for i in range(1,11):
        result = num * i
        print(f"{n} x {i} = {result}")
num = int(input("Enter a number:"))
print_table(num)
```

→ Output:

Enter a number: 3

3x1 = 3

3x2 = 6

⋮

(table of 3)

⑤ WAPP to implement bubble sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
arr = list(map(int, input("Enter elements:").split()))
bubble_sort(arr)
print("Sorted array:", arr)
```

→ Output:

Enter elements: 6 5 4 8 9 2

Sorted array: 2 4 5 6 8 9

⑥ WAPP to print reverse of an integer

```
def reverse_integer(n):
    reversed = int(str(n)[::-1])
    return reversed
num = int(input("Enter an integer:"))
result = reversed_integer(num)
print(result)
```

→ Output:

```
Enter an integer: 359874
478953
```

A 10-11-23

LAB. 7/11/23

17/11/23

LAB - 1

Tic Tac Toe

~~board = [' for x in range(10)']~~

```
def insertletter(letter, pos):
    board[pos] = letter

import random
let = [1,2,3,4,5,6,7,8,9]

def board(let):
    print(" let[0] + "|" + let[1] + "|" + let[2] + "|")
    print("- - -")
    print(" let[3] + "|" + let[4] + "|" + let[5] + "|")
    print("- - -")
    print(" let[6] + "|" + let[7] + "|" + let[7] + "|")
def winner(tic, pos):
    if tic[0] == tic[4] and tic[4] == tic[8] or
        tic[2] == tic[4] and tic[4] == tic[6]:
    return true
    else if
        tic[pos-0] == tic[pos-3] and tic[pos-3] == tic[pos-6]
    return true.
else if tic[pos//3+1] = tic[pos//3+2] and tic[pos//3+2]
                == tic[pos//3+3]:
return true

def update (let):
    num = int(input(" Enter number "))
    while (num not in let):
        num = int(input(" Enter a number "))
    let[num-1] == 'O'
```

```
def update-comp(let):
    for i in let:
        if i!='X' and i!='O':
            let[i-1]='X'
            if(Winner(let, i-1)==true):
                return
    else:
        let[i-1]=i
    for i in let:
        if i!='X' and i!='O':
            let[i-1]='O'
            if(Winner(let, i-1)=='False');
                return
            else:
                let(i-1)=i
    num=random.rand(a)
    while(num not in let);
        num=random.rand(a)
        tic[num-1]=='X'

print Board(let)
    count=0
    if count%2==0:
        print("Computer turn")
        update-comp(let)
        count+=1
    if count>=5:
        if(Winner(tic, pos)==True):
            print("Winner", let[pos-1])
```

LAB-2

Date 24/11/23
Page

# 8-puzzle problem using BFS

Algorithm:

Input: Source & Target to be Solved.

Output: & BFS traversal till target is reached

Step 1: Initializing Queues & Source

```
queue = []
queue.append (src)
exp = []
```

Step 2: BFS loop & popping queue

```
while len(queue) > 0;
source = queue.pop(0)
exp.append (source)
```

Step 3: Checking if Source reached target

```
if source == target;
print ("Success")
return
```

Step 5: Generate, Explore & check for Possible unexplored Moves

```
poss_moves = possible_moves (source, exp)
for move in poss_moves:
if move not in exp and move not in queue:
queue.append (move)
```

Step 6: Possible Moves, Identifying empty Spot & define directions.

```
def possible_moves (state, visited_states):
b = state.index(0)
d = []
```

Step 7: All possible directions.

```
if b not in [0,1,2]:
    d.append('u')
if b not in [6,7,8]:
    d.append('d')
if b not in [0,3,6]:
    d.append('l')
if b not in [2,5,8]:
    d.append('r')
```

Step 8: Generating & Applying Moves

```
def generate(state, m, b):
    temp = state.copy()
    if m=='d'
    Swap(b+3,b)
    if m=='u':
    Swap(b-3,b)
    if m=='l':
    Swap(b-1,b)
    if m=='r':
    Swap(b+1,b)
    return temp;
```

Step 9: Give Source & Target

Step 10: Run BFS

```
bfs(src, target)
```

classmate
Date 8/12/23
Page

LAB-3

8-puzzle using iterative deepening search algorithm

Algorithm:

Step 1: Initializing start state & goal state.

Start state = [ ]
goal state = [1, 2, 3, 4, 5, 6, 7, 8, 0]

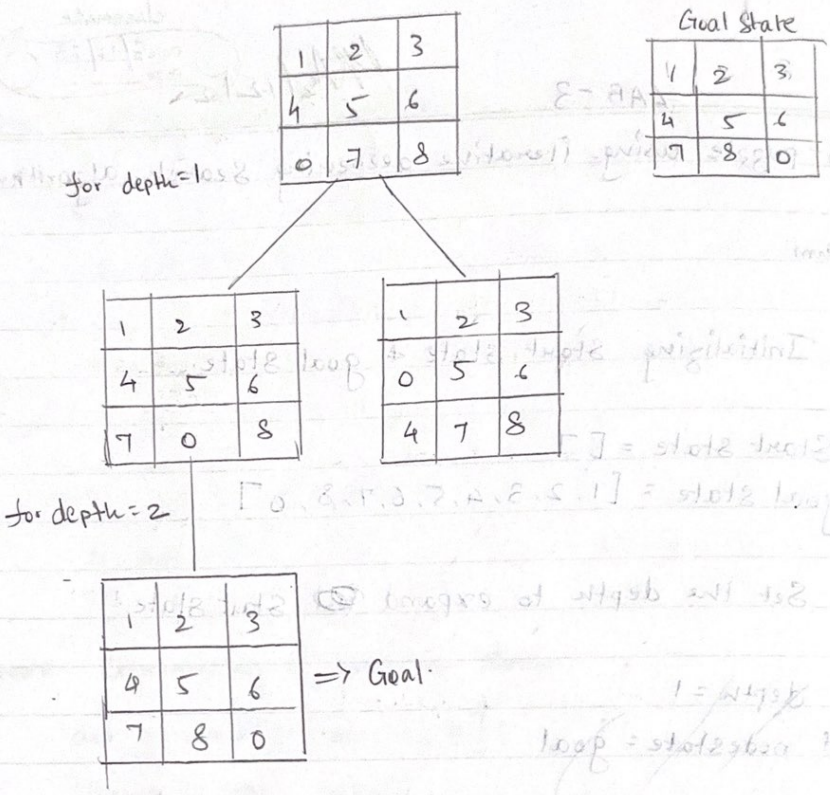
Step 2: Set the depth to expand start state.

~~depth = 1~~
~~if nodestate = goal~~

depth = 1
DFS(depth)
if nodestate = goal
return node
else
for neighbour in neighbour(node state)
child = puzzlenode(neighbour, node)
result = DFS(dept-1)
if result = True;
return result.

Step 3: After each iteration do depth += 1 and doing DFS()

Step 4: neighbours will generate possible moves by swapping 0


Goal State
1 2 3
4 5 6
7 8 0
for depth=1
1 2 3
4 5 6
7 0 8
1 2 3
0 5 6
4 7 8
for depth=2
1 2 3
4 5 6
7 8 0
=> Goal.

LAB - 4.

8 puzzle using A*

Algorithm

Step 1: Initialising Start State & goal state.

Start state = [ ]

goal state = [1, 2, 3, 4, 5, 6, 7, 8, 0]

Step 2: Defining heuristic value such that lower heuristic value is considered.

$f$value = heuristic + pathcost

Step 3: Calculate the heuristic function value

$f(x) = h(x) + g(x)$ ← pathcost

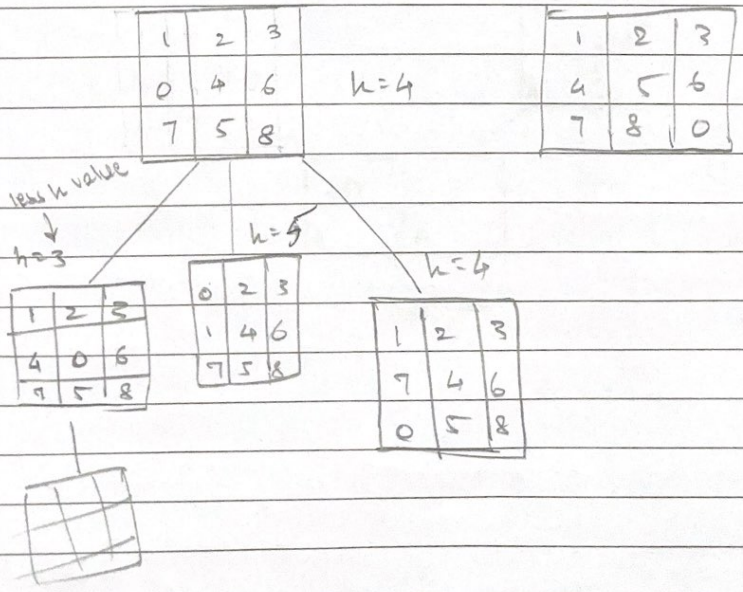
↑ otite

Step 4: Maintain two lists 'open' & 'close'

node(states) → open list

visited nodes → close list & removed from open

Step 4: When $h(x) = 0$ then goal state is reached.



22/12/23

URA22-12-23

classmate
Date
Page

LAB-5

Vaccum cleaner agent

(i) For two rooms

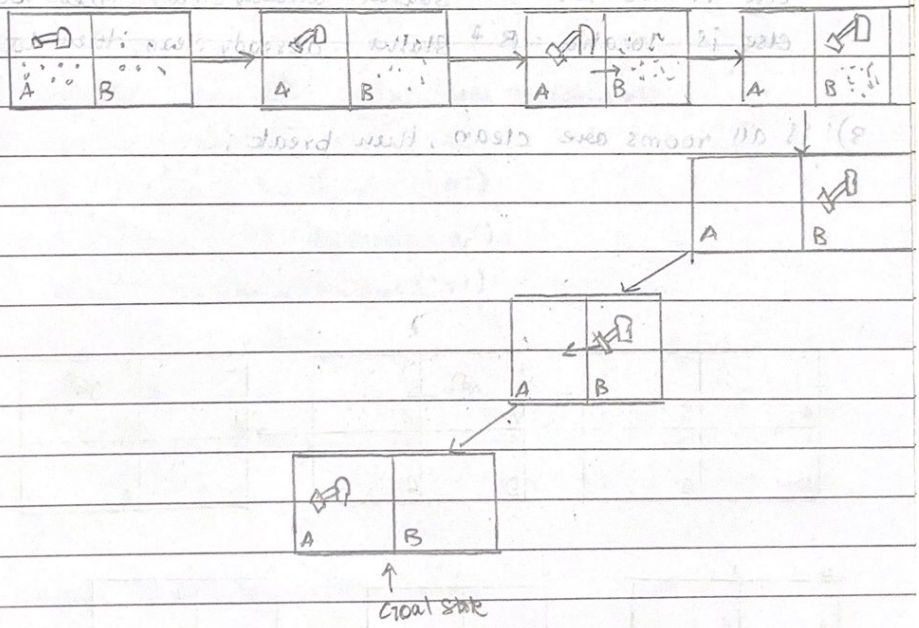
→ Algorithm:

1) Initalize the Starting & goal State
↓
to clear both A&B rooms

goalstate = {'A':'0', 'B':'0'}

2) If status = Dirty then clean
else if location = A & Status = clean, then return right
else if location = B & status = clean, then return left
else exit

3) if both rooms are clean, then break; task done.



(2) For 4 rooms

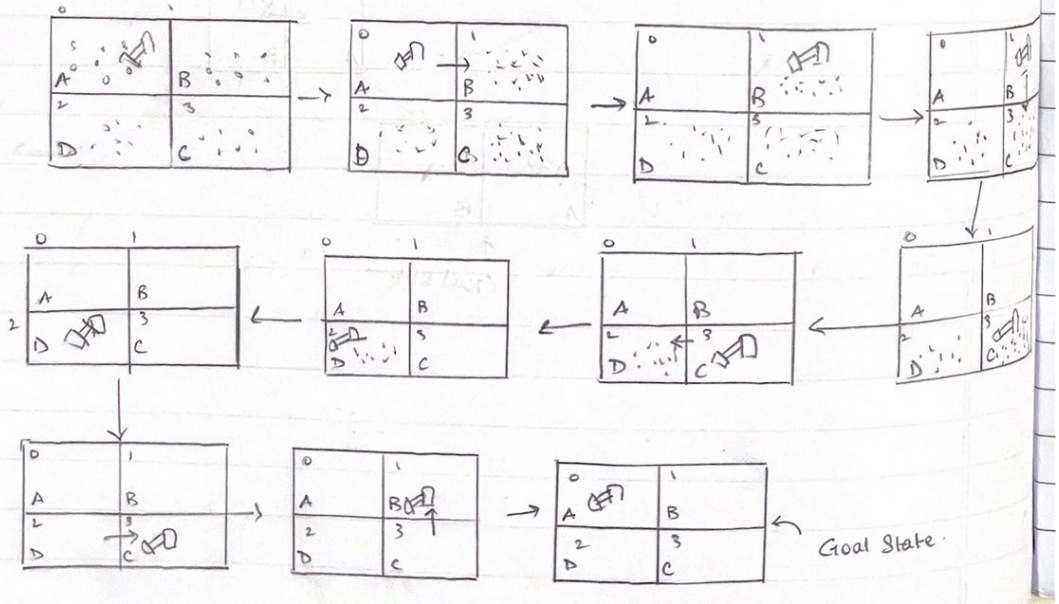
→ Algorithm:

1) Initialise starting and goal state
   ↳ clean all rooms A, B, C, D

2) index = [0, 1, 2, 3]
   goal state = ['A': '0', 'B': '0', 'C': '0', 'D': '0']
   Cost = 0

2) If Status = Dirty, then clean
   else if location = A & Status = clean, then location = index+1
   else if location = B & Status = clean, then location = index+2
   else if location = C & Status = clean, then location = index-1
   else if location = D & Status = clean, then location = index-2
   ~~else if location = C & Status = already clean, then location = index;~~
   ~~else if location = B & Status = already clean, then location = index~~

3) if all rooms are clean, then break;



29/12/23

URA 9-12-23

classmate
Date
Page

# LAB - 6
## Knowledge base Entailment

→ Algorithm:

Step 1: Define propositional symbols

```
class PropositionalSymbol:
    def __init__(self, name):
def Implies(p, q):
    return (notp) or q
def Not(p)
    return notp
def And(*args)
    return all(args)
def satisfiable(formula):
    return True
```

Step 2: Define logic base using logic statements

```
def create-knowledge-base():
    p = PropositionalSymbol('P')
    q = PropositionalSymbol('q')
    r = PropositionalSymbol('r')
knowledge-base = And(
    Implies(p, q),
    Implies(q, r),
    Not(r)
)
return knowledge-base
```

Step 3: Check if knowledge base entails the query

```
def query-entail(knowledge-base, query):
entailment = Satisfiable(And(knowledge-base, Not(query)))
    return not entailment
```

Step 4: Create knowledge base

```
kb = create_knowledge_base()
```

Step 5: Define query

```
query = PropositionalSymbol('p')
result = query_entails(kb, query)
```

Step 6: Display result.

```
print("knowledge Base:", kb)
print("Query :" query.name)
print("Query entails knowledge Base:", result)
```

29/12/23

LAB-6

## Resolution

→ Algorithm:

Step1: Initialize an empty list for knowledge base ('kb')

```
def tell(kb, rule)
    kb.append(rule)
```

Step2: Define a list of all possible combinations

```
Combinations = [(True, True, True), (False, True, True) ....
    , (False, False, False)]
```

Step3: Define a function that takes knowledge base and a query function for each combination

```
def ask(kb, q):
    for c in combinations:
        s = all(rule(c) for rule in kb)
        f = q(c)
        print(s, f)
        if s != f and s != False:
            return "Does not Entail"
        return 'Entails'
kb = []
```

Step 4: Get user input

```
rule = input("Enter Rule 1 as a lambda function:")
r1 = eval(rule)
tell(kb, r1)
```

Step 5: Get user input Query

```
query = input("Enter Query as a lambda function:")
q = eval(query
result = ask(kb, q)
print(result)
```

RA1911-23

19/1/24

LAB-7

## Unification.

→ Algorithm:

Step 1: If A & B is a variable or constant, then
- → A & or B are identical return NIL
- → Else if A is available if A occurs in B
  return FAIL
  else
  return {A/B}
- → else if B is a variable if B occurs in A
  return FAIL
  else
  return {A/B}
- → else return FAIL

Step 2: If predicate (A) ≠ predicate (B)
return FAIL

Step 3: Number of arguments ≠ return FAIL

Step 4: Set (SUBST) to NIL

Step 5: For i=1 to the number of elements in A
- → call unify ($i^{th}$ A, $i^{th}$ B)
  put result into S
- → S = FAIL return FAIL
- → If S ≠ NIL
  - Apply S to the remainder of both L1 & L2
  - SUBST = APPEND (S, SUBST)

Step 6: Return SUBST

19/1/24

LAB-7

FOL to CNF

→ Algorithm:

Step 1: Make a list of SKOLEM_CONSTANTS

Step 2: Find ∀, ∃

If attributes are lower case → replace with SKOLEM_CONSTANT

Remove used Skolem constant or function from the list

If attributes are both lowercase & uppercase replace the uppercase attribute with a Skolem function.

Step 3: replace ⟷ with '_'

transform _ as $Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

Step 4: replace ↔ with '_'

Step 5: DeMorgan's law

replace ~[

as ~P|~Q if( was present)

replace ~[

as ~P|~Q if( & was present)

replace ~~ with ''

Step 6: Return result

19/1/24

LAB 7

# Forward Chaining

→ Algorithm:

Step 1: Input knowledge base and the query

Step 2: for i in kb:
    if i == query return True
    if '=>' in i:
        Split LHS and RHS part
        if LHS in kb:
            add RHS to kb
return false

Step 3: To remove variables
    if i.lower():
        replace the variable with constants

→ Example:

john(x) & greedy(x) => evil(x)
king(john)
greedy(john)
king(Richard)

Query
evil(x)