1-10-24 Lab-1

TIC-TAC-TOE

Initialize board as dictionary with key 1-9, all values set to ' '

function print board (board)
Print board [1] + '|' + board[2] + '|' board [3]
Print '-+-+-'
Print board [4] + '|' + board[5] + '|' + board [6]
Print '-+-+-'
Print board[7] + '|' + board[8] + '|' + board [9]
Print new line.

function space-free(position):
Return board [position] is ' '

function check_win():
Define win_condition as a list of winning conditions
for each combination in win_conditions:
If all positions in combination have the same non-empty value in board:
Return true.

Function check_draw():
Return all position in board are not ' '

function insert_letter(letter, position):
If space_free(position):
Set board [position] to letter.
Call print_board(board)

If check_draw():
Print 'Draw!'
Return True

Else if check_win():
Print letter + 'wins!'
Return True.

Else:
Print 'position taken, please pick a diff. position'
PROMPT User for new position.

function comp_move():
Set best_score to -1000
Set best_move to 0
FOR each key in board:

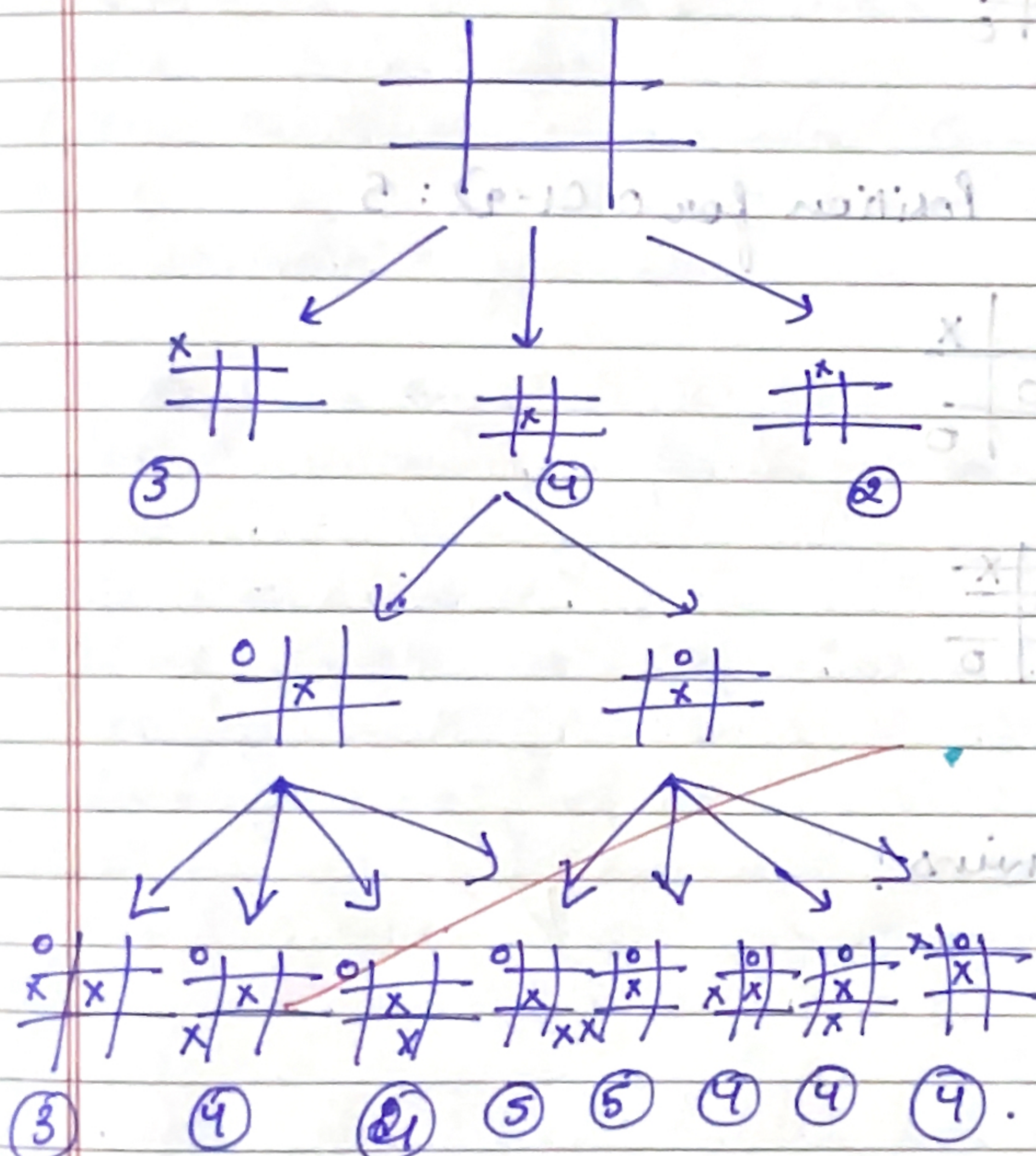
IF space_free(key):

Set board[key] to X
Set score to call minimax(board, false)
Set board[key] to
If score > best_score:
Update best_score to score
Update best_move to key

function minimax(board, is_maximizing)
If check_win();
Return -1 If is_maximizing else 1
Else if check_draw:
Return 0.

While true:
call comp_move()
If check_win() OR check_draw():
Break LOOP
Call player1_move()
If check_win OR check_draw():
Break LOOP.

State space tree:-



Output:-

X

Enter Position for O (1-9): 9

X

X O

Enter Position for O (1-9): 5

X X

O

X X

O

X wins!

Vacuum Cleaner :-

function vacuum ():
Set goal state to {'A' : '0' : 'B' : '0'}.
Set cost to 0.

Prompt user for Location of vacuum (A or B)
Store input as Location_input
Prompt user for status of Location_input
(0 for clean, 1 for dirty)
Store input as status_input.
Prompt user for Status of other Location (0 for clean, 1 for dirty)
Store input as store_input_compliment.

Print initial goal state.

If Location input is 'A':
Print "Vacuum is placed in Location A
If status input is 1:
Print "Location A is Dirty."
Set goal_state ['A'] to '0'
Increment cost by 1
Print "Cost of cleaning A: cost"
Print "Location A has been cleaned."

Else:
Print "No action. Location B is already clean".

Else if location input is 'B':
Print "Vacuum is placed in Location B".

If [illegible] status [illegible] '1':
Print "[illegible] B is dirty".
Set goal state ['B'] to 0.
Increment cost by 1
Print "Cost for cleaning B: cost".
Print "Location B has been cleaned."

Else:
Print "No action. Location A is already clean"

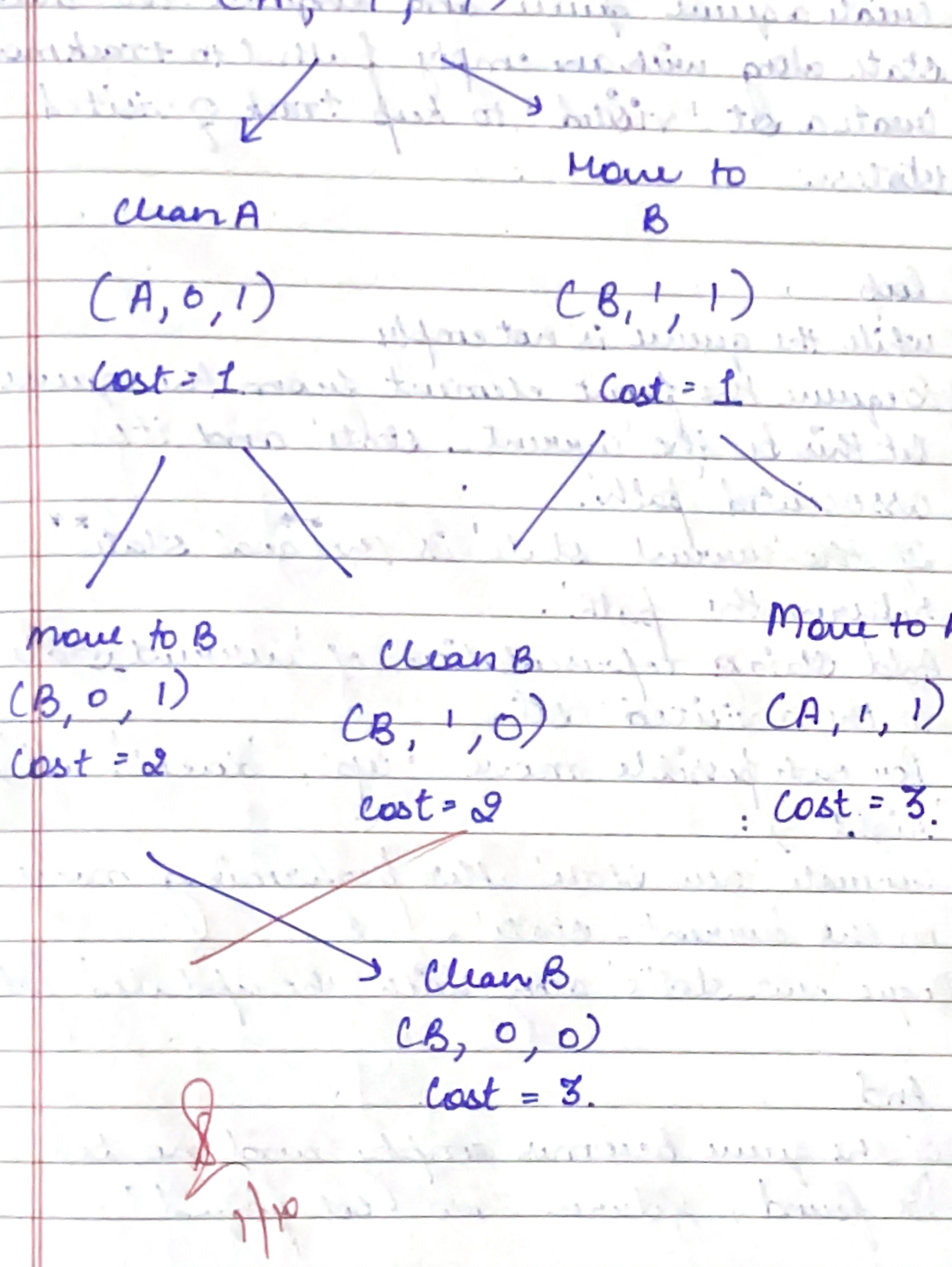
Print "Goal state: goal state"
Print "Performance Measurement: cost"

CALL Vaccum()

Enter Location of Vacuum (A or B): A
Enter status of A (0 for clean : 1
1 for Dirty)

Initial Location condition: {'A': '0': 'B': '0'}
Vacuum is placed in Loc A.
Location A is Dirty.
Cost of cleaning A: 1
Location A has been cleaned.
No action. Loc B is already clean
Goal State: {A : 0 : B : 0}
Performance Measurement: 1

State - Space Diagram:



n/p

8-10-24

8 Square Puzzle Using BFS:

1.) Initialize
→ Create a queue 'queue' and enqueue the initial state along with an empty path (to track moves)
→ Create a set 'visited' to keep track of visited states.

2.) loop
while the queue is not empty
→ Dequeue the front element from the queue. Let this be the 'current_state' and its associated 'path'.
→ If the 'current state' is the **goal state** return the 'path'.
→ Add string representation of 'current_state' to the 'visited' set.
→ for each possible move ('up', 'down', 'left', 'right'):
• Generate 'new state' after performing move on the 'current_state'.
Enque 'new_state' along with the updated 'path'

3.) End
If the queue becomes empty and no Sol is found, Return "no Sol found".

DFS :-

• Initialize
→ Create a stack 'stack' and p
along with an empty
→ Create a set 'visited' to ke

• Loop
while the stack is not emp
→ Pop top element from t
be current state.
→ If the current state is g
'path'.
→ Add string representatio
to the visited set.
→ for each possible move (
• Generate new state is va
visited.
• Push new state onto the s
updated 'path' (including

• End
If stack becomes empty
return "No Sol found"

DFS :-

- Initialize
→ Create a stack 'stack' and push the initial state along with an empty path.
→ Create a set 'visited' to keep track of visited states.

- Loop
while the stack is not empty:
→ Pop top element from the stack. Let this be current state.
→ If the current state is goal state return the path.
→ Add string representation of the current state to the visited set.
→ for each possible move ('up', 'down', 'left', 'right')
  - Generate new state is valid and hasn't been visited.
  - Push new state onto the stack along with the updated path (including current move).

- End
If stack becomes empty and no sol is found return "No Sol found".

State - Space Tree (BFS)