INDEX

NAME: Shashank Patel C.J. STD.: _____ SEC.: 'E' ROLL NO.: 255

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Artificial intelligence		
1.(a)	24/9/24	TPE – TOC – TOE		24-9-24
1.(b)	1/10/24	Vacuum world		
2	8/10/24	8 puzzle problem using DFS, BFS (Non heuristic approach).		8/10/24
3	15/10/24	8 puzzle problem using A* Search with method of 1) Misplace tiles. 2) Manhattan Distance.		15/10/24
4	22/10/24	i) 8 puzzle problem using iterative deepening approach. ii) N-Queen using hill climbing and Backtracking		22-10
5	29/10/24	Simulated Annealing problems.		29/10/24
6	12/11/24	propositional entailment		12/11/24
7	19/11/24	Implement unification in FOL (First Order Logic).		19/11/24
8	25/11/24	Forward reasoning		26/11/24
9	03/12/24	FOL into resolution		
10	03/12/24	Alpha-Beta pruning		3/12/24

Lab - 1.67

Bafna Gold
Date: 29/9/24

Implement tic-tac-toe game

Algorithm

* Create an 2-dimensional array and the all is initialised with Space ' ' and to represent the empty cell.
* Then create a function print_board to create 3x3 dimension board.
* The function check_winner is created to check the winner; then check the row, columns and diagonals to get or to find out the winner. If no winner found return None.
* The function is_board_full to check the board is full or not to evaluate the tie condition.
* The main function tic_tac_toe() is used to call the functions initialize board() and current player is taken at 'x'.
* Now check the input with condition of row<0 or row>2 or col<0 or col>2 or board[row][col] != ' ' for invalid input.
* Therefore, with proper input continue with the game to get the output winner or tie.

24-9-24

01/10/24

Lab-2(b)

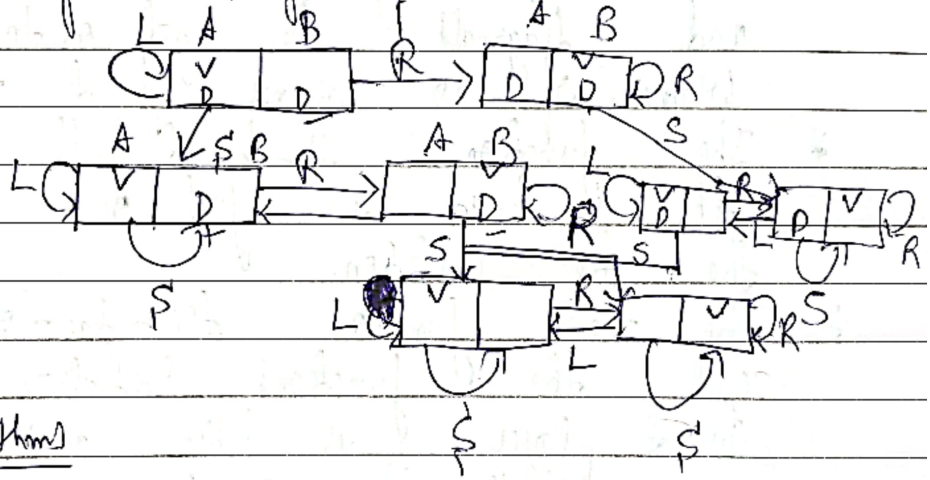
Implement Vacuum world cleaner.

Pseudo code:

function REFLEX-VACUUM-AGENT([Location, status])
returns an action.

if Status = Dirty then return Suck.
else if location = A then return Right.
else if location = B then return Left.

State - Space Diagram of Vacuum world.



* Algorithms

* The function implemented in the cleaner, print_f and the main function.
* The main function contains the floor list where the user enters the row and columns.
* It considers '1' as dirty and '0' as clean.
* In the clean function it contains row, col and the even row Iterate from left to right and add row

iteration through right to left.

* The printf contains or updates each iteration and evaluates the status and each operation or action.
* Also the '1' is changed to '0' and "goal is achieved".

SSS
11/10/24.

03/10/24

Lab - 2

Solve 8 puzzle problems using DFS and BFS

1. Problem representation -

* State: Represent the state of the puzzle as a 3x3 grid or a one-dimensional array.
* Goal State: Define the target arrangement, as:

1	2	3
4	5	6
7	8	-

2. Common Components -

* Function to Generate Successors: Create a function that produces all valid states that can be reached from the current state by moving the blank tile/space tile to left, right, up, down.

3. Successor Generation -

Function: generate_successors(state)

* Find the position of the blank tile/space tile.
* Generate new states by swapping the blank tile with adjacent tiles (up, down, left, right) if the moves are valid.

* Now, the tracing is done by the above approach to generate the goal state from the above "Actions". Where start state example like and goal state like.

1	2	3
4		5
6	7	8

Start State

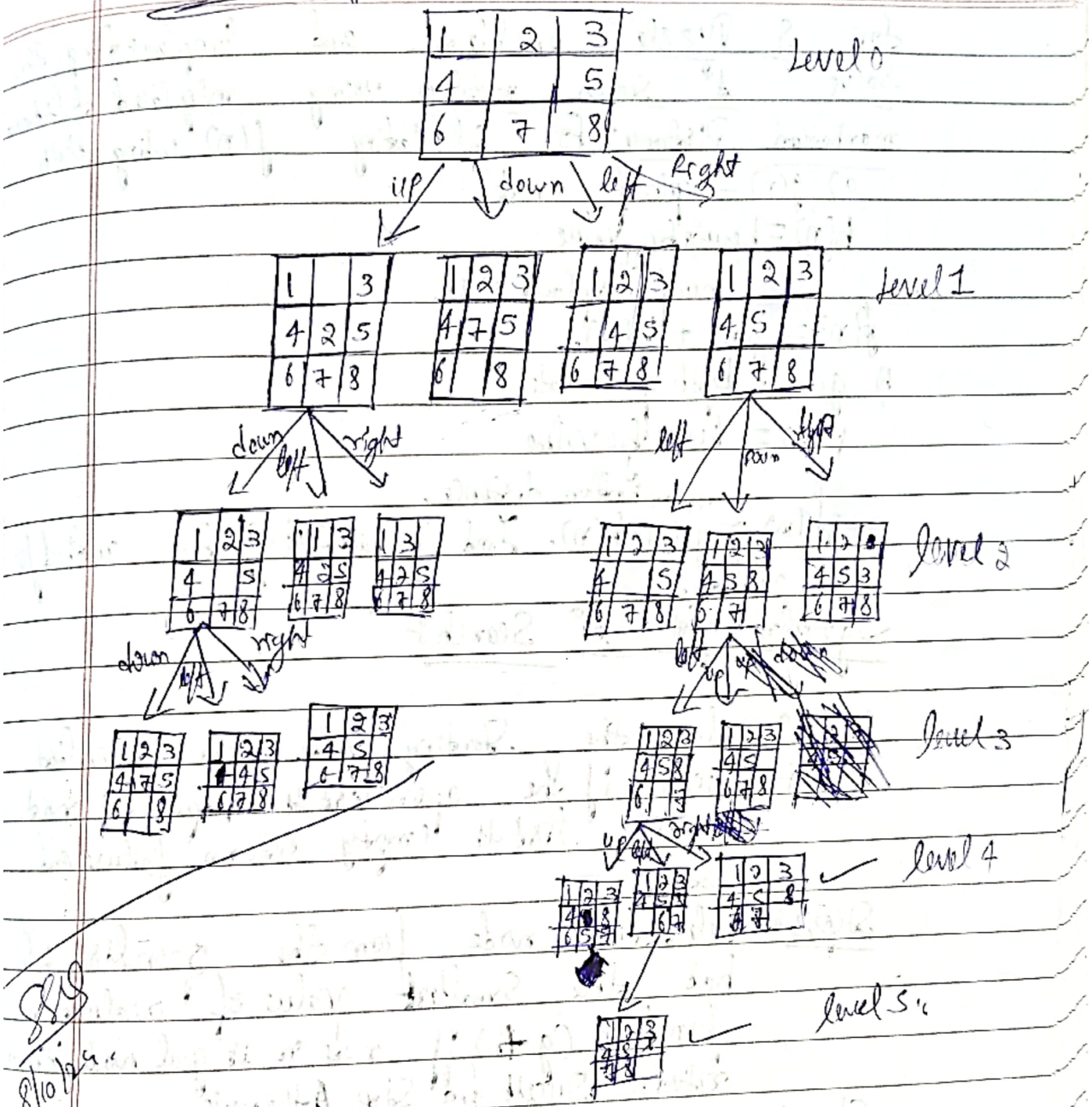
1	2	3
4	5	6
7	8	

Goal State

Using BFS and DFS for this problem.

Bafna Gold
Date: Page:

State Space tree



15/10/24

Lab-3

For 8 Puzzle problem and implementing the Solve A* Search problem using misplaced tiles and manhattan Distance for calculating f(n) using both

a) g(n) = depth of a node
   h(n) = heuristic value
          ↓
          no. of misplaced tiles.
   f(n) = g(n) + h(n).

b) g(n) = depth of a node
   h(n) = heuristic value
          ↓
          manhattan distance.
   f(n) = g(n) + h(n). Find the best cost effective method for that problem.

Algorithm for A* Search:

Step 1: place the Starting node in the openlist.

Step 2: check if the open list is empty or not. If the list is empty return failure and Stop.

Step 3: Select the node from the open list which has the smallest value of evaluation function (g+h), if node n is goal node then return Success and Stop, Otherwise

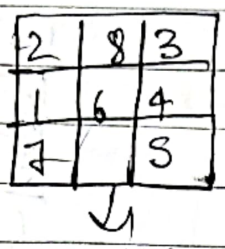
Step 4: Expand node n and generate all of its successors and, put it in the closed list, for each Successor 'n', check whether n is already in open or closed list, if not then compute evaluation function for n and placed into open list.

Step 5: else if node n is already in open and closed, then it should be attached to the back pointer which reflects the lowest g(n) value

Step 6: At the last, after the iteration return to the Step 2.

Bafna Gold
Date: Page:

Draw the state space diagram for :-

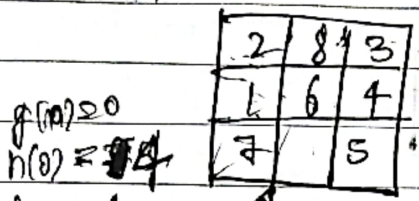


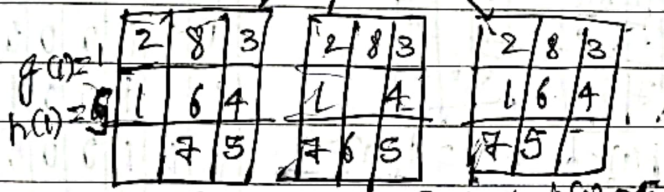
Initial

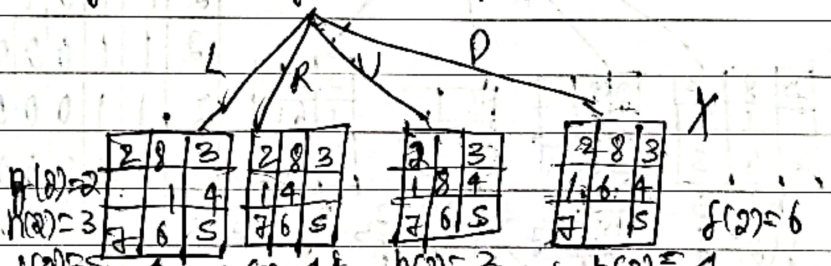
goal state

Final goal state

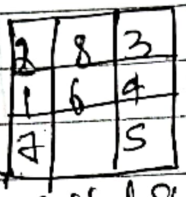
Solution :-



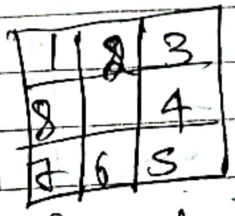




Draw the state space diagram for Manhattan Distance:-

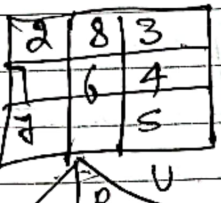


Initial State



final State (goal State)

Solution:-



$g(0)=0$
$h(0)=5$
$f(0)=5$

L R U

$g(1)=1$

$h(1)=6$ $f(1)=7$ | $h(1)=6$ $f(1)=7$ | $h(1)=4$ $f(1)=5$

L R U D

$g(2)=2$

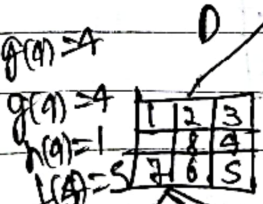
$h(2)=5$ $f(2)=7$ | $h(2)=5$ $f(2)=7$ | $h(2)=3$ $f(2)=5$ | $h(2)=5$ $f(2)=7$

L R D

$g(3)=3$

$h(3)=2$ $f(3)=5$ | $h(3)=4$ $f(3)=7$ | $h(3)=4$ $f(3)=7$

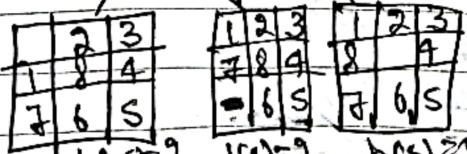
D R



$g(4)=4$
$h(4)=1$
$f(4)=5$

$h(4)=3$
$f(4)=7$

U D R



$g(5)=5$

$h(5)=2$ $f(5)=7$ | $h(5)=2$ $f(5)=7$ | $h(5)=0$ $f(5)=5$

Level no	1	2	3	4	5	6	7	8	h(n)	
1	1	1	0	0	0	1	1	2	6	
1	1	1	0	0	1	1	0	2	6	
1	1	1	0	0	0	0	0	2	4	✓
2	2	1	0	0	0	0	0	2	5	
2	1	1	0	1	0	0	0	2	5	
2	1	1	0	0	0	0	0	1	3	✓
2	1	1	0	0	0	1	0	2	5	
3	1	0	0	0	0	0	0	1	2	✓
3	1	1	1	0	0	0	0	1	4	
3	1	1	0	0	0	0	0	2	4	
4	0	0	0	0	0	0	0	1	1	✓
4	1	1	0	0	0	0	0	1	3	
5	1	0	0	0	0	0	0	1	2	
5	0	0	0	0	0	0	1	1	2	
5	0	0	0	0	0	0	0	0	0	✓

15/10/24

22/10/24

Lab - 4. (a)

Bafna Gold
Date: Page:

8 puzzle problem and solving using Iterative deepening approach. (search).

Algorithm

1. For each child of the current node.
2. If it is the target node, return.
3. If the current maximum depth is reached, return.
4. Set the current node to this node and go back to 1.
5. After having gone through all children, go to the next child of the parent (the next sibling).
6. After having gone through all children of the start node, increase the maximum depth and go back to 1.
7. If we have reached all leaf (bottom) nodes, the goal node doesn't exist.

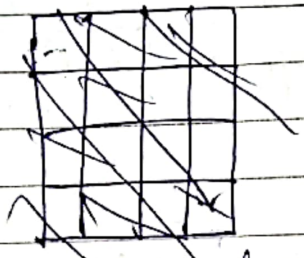
function Iterative_deepening_Search(problem) returns a solution or failure

for depth = 0 to ∞ do

result ← depth-limited-Search(problem, depth)

If result ≠ cutoff then return result.

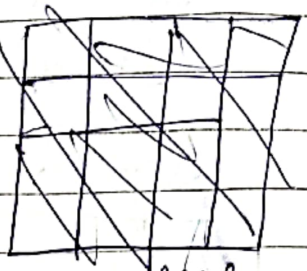
State Space tree Non heuristic approach



Initial State

2	8	3
1	6	4
7		5

Initial State



Goal State

level 0

2	8	3
1	6	4
7		5

Initial State

Level = 0
depth = 0

1	2	3
4	5	6
7	8	

Goal State

level 1

limit = 2

2	8	3
1	6	4
7		5

depth = 0
level = 0

L

2	8	3
1	6	4
	7	5

U

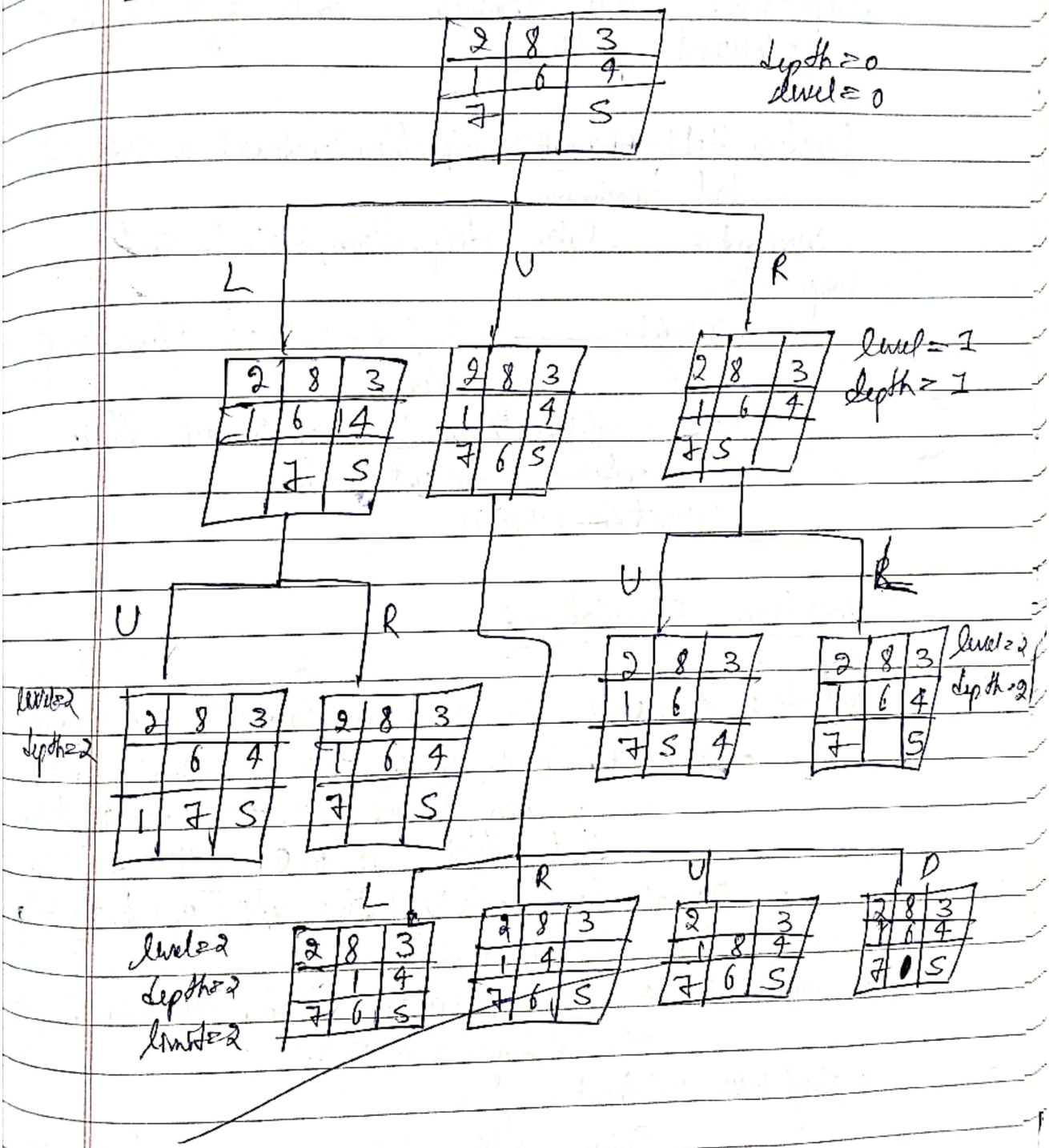
2	8	3
1		4
7	6	5

R

2	8	3
1	6	4
7	5	

level = 1
depth = 1

level 2



22/10/24

Lab - 4(b)

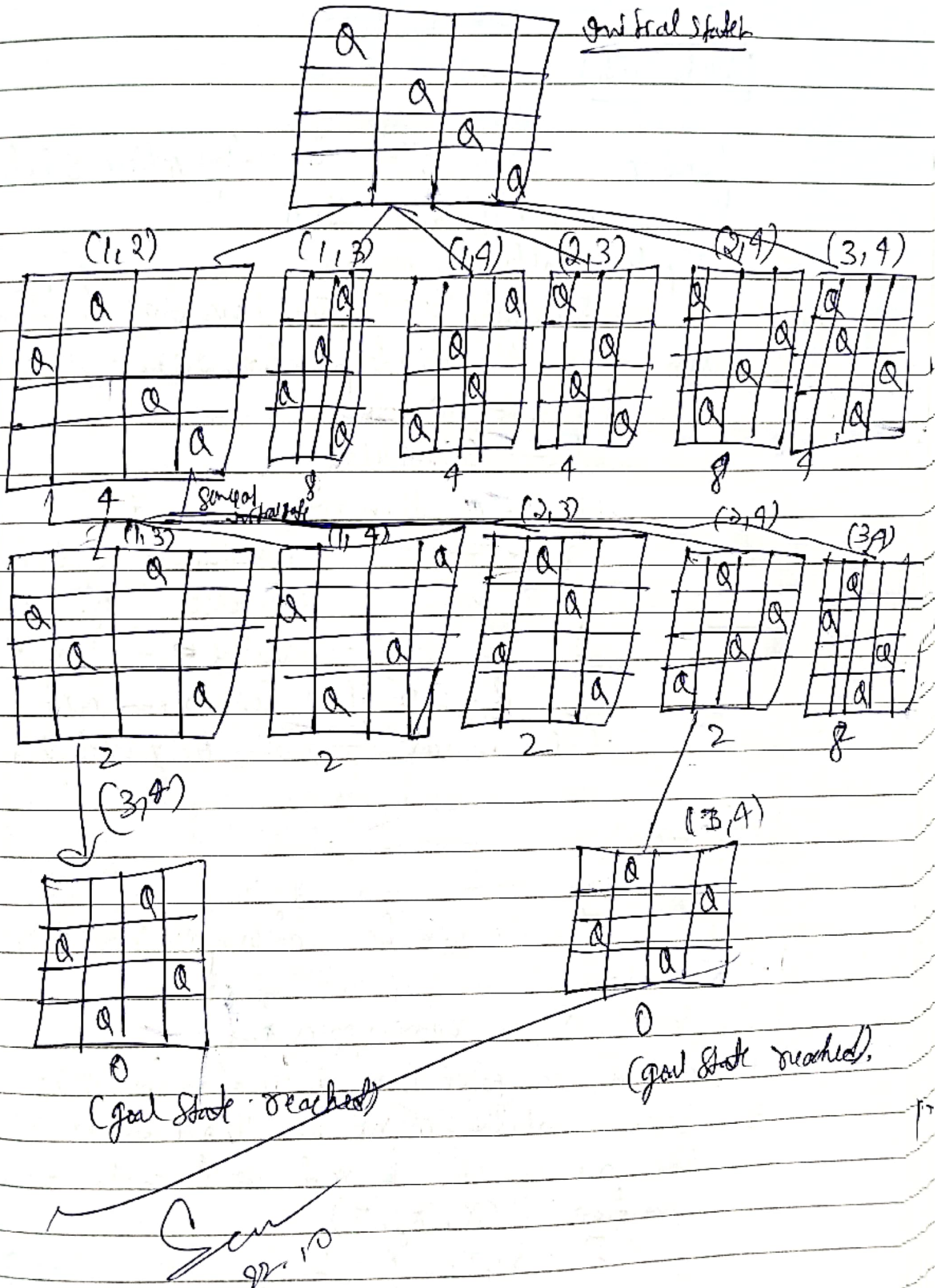
Implement Hill climbing Search algorithm to solve N-Queens problem.

function HILL-CLIMBING(problem) returns a state that is a local maximum.
current ← Make-Node(problem, Initial-State).
loop do
neighbor ← a highest-valued successor of current.
if neighbor.Value ≤ Current.Value then
return Current.State
Current ← neighbor

Problem Formulation

* State : 4 queens on the board. One queen per column.
— Variables: x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i. Assume that there is one queen per column.
— Domain for each variable: $x_i \in \{0, 1, 2, 3\}$ for all i.
* Initial State: A random state.
* Goal State: 4 queens on the board. No pair of queens are attacking each other.
* Neighbour relation
Swap the row positions of two queens.
* Cost function: The number of pairs of queens attacking each other, directly or indirectly.

State Space Tree



Lab-5:- 29/10/24

Write a program to implement Simulated Annealing Algorithm.

Pseudo code:

function Simulated-Annealing(problem, Schedule) returns a solution state
inputs: problem, a problem
Schedule, a mapping from time to "temperature"
Current ← Make-Node(problem.Initial-State)
for t = 1 to ∞ do
T ← Schedule(t)
if T = 0 then return current
next ← a randomly selected successor of current
ΔE ← next.Value – Current.Value
If ΔE > 0 then Current ← next
else Current ← next only with probability $e^{\Delta E/T}$

Algorithm:

The algorithm can be decomposed in 4 simple steps:

1. Start at a random point x_i.
2. Choose a new point x_j, on a neighbourhood $N(x)$
3. Decide whether or not to move to the new point x_j. The decision will be made based on the probability function $P(x_i, x_j, T)$

$$P(x_i, x_j, T) = \begin{cases} 1 & \text{si } f(x_j) \geq f(x_i) \\ e^{\frac{F(x_j)-F(x_i)}{T}} & \text{si } F(x_j) < F(x_i) \end{cases}$$

4. Reduce T.

1) Output: 8 Queen's problem

The best position found is: [5 3 0 4 7 1 6 2]
The number of queens that are not attacking each other is: 8.0
Chess board configuration:

. . Q . . . . .
. . . . . Q . .
. . . . Q
. Q . . . . . .
. . . Q . . . .
Q . . . .
. . . . . Q . .
. . . . Q . . .

2) output: Tower of Hanoi problem

Best State (final configuration): [2 2 2]
Number of correct disks on destination peg: 3.0
Tower of Hanoi Configuration:
peg 0: []
peg 1: []
peg 2: [0, 1, 2]

S.S
29/10/24

Lab - 6

12/11/24

Truth table enumeration algorithm for deciding propositional entailment

A truth-table enumeration algorithm for deciding propositional entailment. (TT stands for truth table). PL-TRUE? returns true if a sentence holds within a model. The variable model represents a partial model - an assignment to some of the symbols. The keyword "and" is used here as a logical operation on its two arguments, returning true or false.

pseudo code for TT-ENTAILS function.

```
function TT-ENTAILS?(KB, α) return true or false:
    inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic.
    Symbols ← a list of the proposition symbols in KB and α.
    return TT-CHECK-ALL(KB, α, Symbols, { })
```

pseudo code for TT-Check-ALL function.

```
function TT-CHECK-ALL(KB, α, Symbols, model)
    return true or false
    if EMPTY?(symbols) then
        if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
        else return true // when KB is false always return true.
```

else do

P ← FIRST(Symbols)

rest ← REST(Symbols)

return (TT-CHECK-ALL(KB, α, rest, model ∪ {P = true}))

and

TT-CHECK-ALL(KB, α, rest, model ∪ {P = false}))

Truth table:

Propositional Inference: Enumeration method:

example: $\alpha = A \vee B$ $KB = (A \vee C) \wedge (B \vee \neg C)$

Checking that $KB \models \alpha$

A	B	C	A∨C	B∨¬C	KB	α
false	false	false	false	true	false	false
false	false	true	true	false	false	false
false	true	false	false	true	false	true
false	true	true	true	true	true	true
true	false	false	true	true	true	true
true	false	true	true	false	false	true
true	true	false	true	true	true	true
true	true	true	true	true	true	true

KB entails α : True.

12/11/24

Lab-7 19/11/24

Implement unification in FOL (First order Logic).

Algorithm

Unify(ψ_1, ψ_2):

Step 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 ψ_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

Step 2: If the initial predicate symbol in ψ_1 and ψ_2 are not same, then return FAILURE.

Step 3: If ψ_1 and ψ_2 have a different number of arguments, then return FAILURE.

Step 4: Set Substitution Set (SUBST) to NIL.

Step 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 returnt FAILURE.
c) If S≠NIL then do,

a. Apply S to the remainder of both L1 and L2
b. SUBST = APPEND(S, SUBST).

Step 6: Return SUBST.

a) P(x, F(y)) → ①
P(a, F(g(x)) → ②
① and ② are identical. If x is replaced with a in ①
P(a, F(y)) → ①
P(a, F(g(x)) → ②
If y is replaced with g(x) in ①
P(a, F(g(x))) → ①
P(a, F(g(x)) → ②
So finally both ① & ② are identical.

a) Q(a, g(x, a), f(y)) → ①
Q(a, g(f(b), a), x) → ②
① & ② are identical. If x is replaced with f(b).
Q(a, g(f(b), a), f(y)) → ①
Q(a, g(f(b), a), x) → ②
If f(y) is replaced with x in ①
Q(a, g(f(b), a), x) → ①
Q(a, g(f(b), a), x) → ②
So finally both ① & ② are identical.

Lab-8

26/11/24

Create a Knowledge Base consisting of first order logic statements and prove the given query using forward reasoning.

A conceptually straightforward, but very inefficient, forward-chaining algorithm. On each iteration, it adds to KB all the atomic sentences that can be inferred in one step from the implication sentences and the atomic sentences already in KB. The function STANDARDIZE-VARIABLES replaces all variables in its arguments with new ones that have not been used before.

Algorithm FOL-FC-ASK(KB, α) returns a substitution or false.

Inputs: KB, the knowledge base, a set of first-order definite clauses
α, the query, an atomic sentence
local variables: new, the new sentences inferred on each iteration.

repeat until new is empty
  new ← { }
  for each rule in KB do
    $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(rule)
    for each θ such that SUBST(θ, $p_1 \wedge \dots \wedge p_n$) = SUBST(θ, $p'_1 \wedge \dots \wedge p'_n$) for some $p'_1, \dots, p'_n$ in KB
      $q' \leftarrow$ SUBST(θ, q)
      if q' does not unify with some sentence already in KB or new then

add q' to new
$\theta \leftarrow$ UNIFY(q', α)
if θ is not fail then return θ
add new to KB
return false

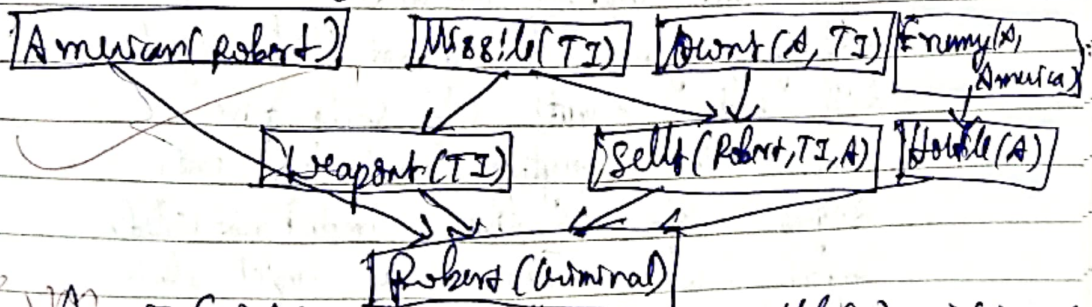
Example:

It is a crime for an American to sell weapons to hostile nations. Let's say p, q, and r are variables.
$American(p) \wedge Weapon(q) \wedge Sells(p, q, r) \wedge Hostile(r) \Rightarrow Criminal(p)$

Country A has some missiles
$\exists x\, Owns(A, x) \wedge Missile(x)$.
Existential instantiation: Introducing a new constant T1:
$Owns(A, T1)$
$Missile(T1)$
All of the missiles were sold to country A by Robert
$\forall x\, Missile(x) \wedge Owns(A, x) \Rightarrow Sells(Robert, x, A)$
Missiles are weapons.
$Missile(x) \Rightarrow Weapon(x)$
Enemy of America is known as hostile.
$\forall x\, Enemy(x, America) \Rightarrow Hostile(x)$

Robert is an American
American(Robert)
The country A, an enemy of America
Enemy(A, America).

To Prove: Robert is Criminal
Criminal(Robert)



$American(p) \wedge Weapon(q) \wedge Sells(p, q, r) \wedge Hostile(r) \Rightarrow Criminal(p)$

Lab-8

Convert a given first order logic statement into resolution.

A Basic steps for proving a conclusion S given premises premise 1 —— premise n

1. Convert all sentences to CNF.
2. Negate conclusion S & convert result to CNF.
3. Add negated conclusion S to the premise clauses.
4. Repeat until contradiction or no progress is made
   a. Select 2 clauses (call them parent clauses).
   b. Resolve them together, performing all required unification.
   c. If resolvent is the empty clause, a contradiction has been found (i.e. S follows from the premises).
   d. If not, add resolvent to the premises.

If we succeed in Step 4, we have proved the conclusion.

Proof by Resolution:

Given the KB of premises

John likes all kind of food.
Apple and vegetables are food.
Anything anyone eats and not killed is food.
Anil eats peanuts and still alive.
Harry eats everything that Anil eats.
Anyone who is alive implies not killed.
Anyone who is not killed implies alive.

prove by resolution that:

John likes peanuts.

Bafna Gold
Date: Page:

Representation in FOL :

a. John likes all kind of food
b. Apple and vegetables are food.
c. Anything anyone eats and not killed is food.
d. Anil eats peanuts and still alive.
e. Harry eats everything that Anil eats.
f. Anyone who is alive implies not killed.
g. Anyone who is not killed implies alive.

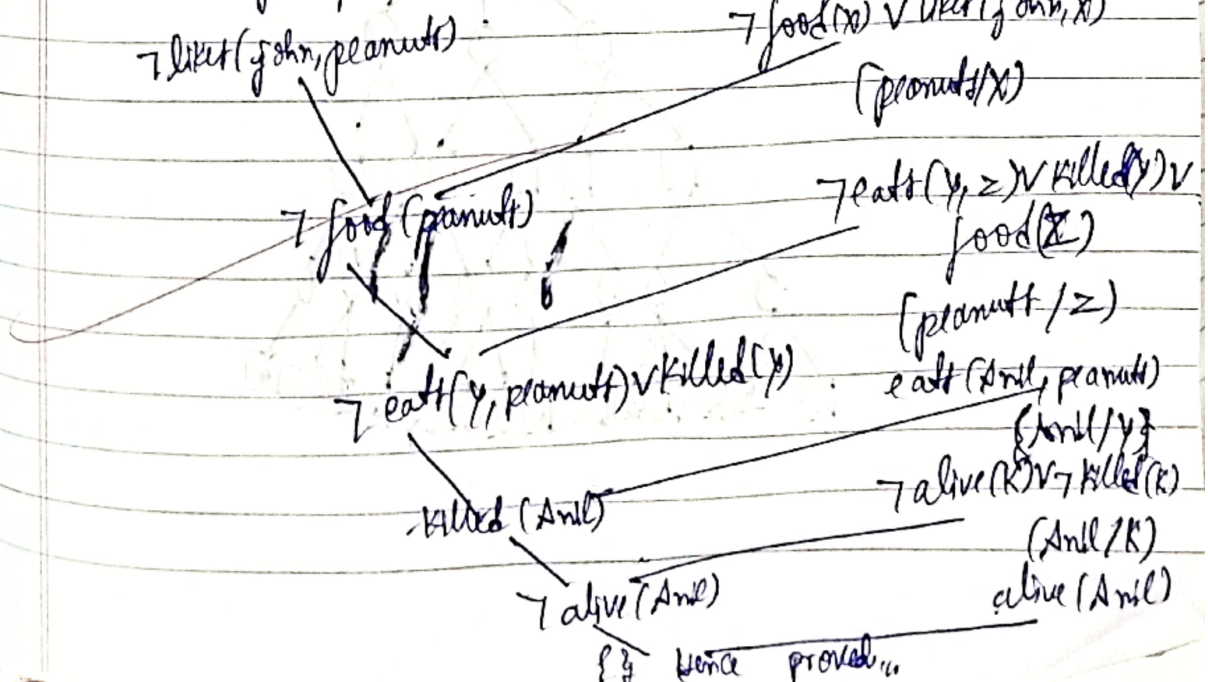
Prove by resolution that :

h. John likes peanuts

a. $\forall x : food(x) \rightarrow likes(John, x)$
b. $food(apple) \wedge food(vegetables)$
c. $\forall x \forall y : eats(x, y) \wedge \neg killed(x) \rightarrow food(y)$
d. $eats(Anil, peanuts) \wedge alive(Anil)$
e. $\forall x : eats(Anil, x) \rightarrow eats(Harry, x)$
f. $\forall x : \neg killed(x) \rightarrow alive(x)$
g. $\forall x : alive(x) \rightarrow killed(x)$
h. $likes(John, peanuts)$

Proof by Resolution :

a. $\neg food(x) \vee likes(John, x)$
b. $food(apple)$
c. $food(vegetables)$
d. $\neg eats(y, z) \vee killed(y) \vee food(z)$
e. $eats(Anil, peanuts)$
f. $alive(Anil)$
g. $\neg eats(Anil, w) \vee eats(Harry, w)$
h. $killed(g) \vee alive(g)$
i. $\neg alive(k) \vee \neg killed(k)$
j. $likes(John, peanuts)$



Lab-10

Implement Alpha-Beta pruning Algorithm:

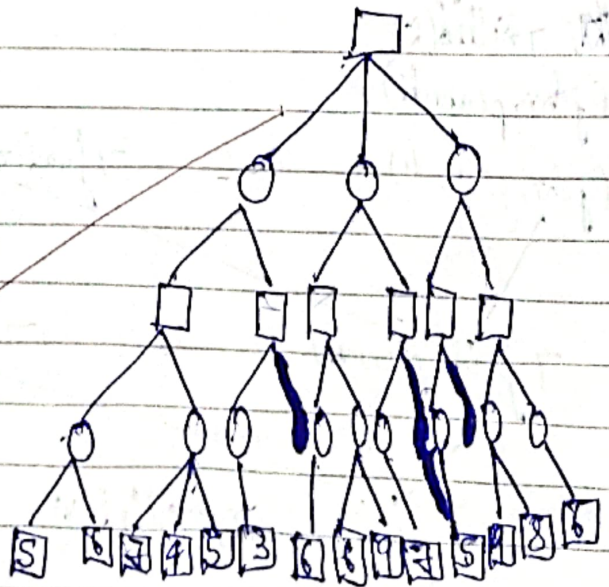
Algorithm:

- Alpha(α) – Beta(β) property to compute find the optimal path without looking at every node in the game tree.
- Max contains Alpha(α) and Min contains Beta(β) bound during the calculation.
- In both MIN and MAX node, we return when α > β which Compare with its parent node only.
- Both minimax and Alpha(α) – Beta(β) algo give same path.
- Alpha(α)-Beta(β) gives optimal solution at its fastest best time to get the value for the root node.

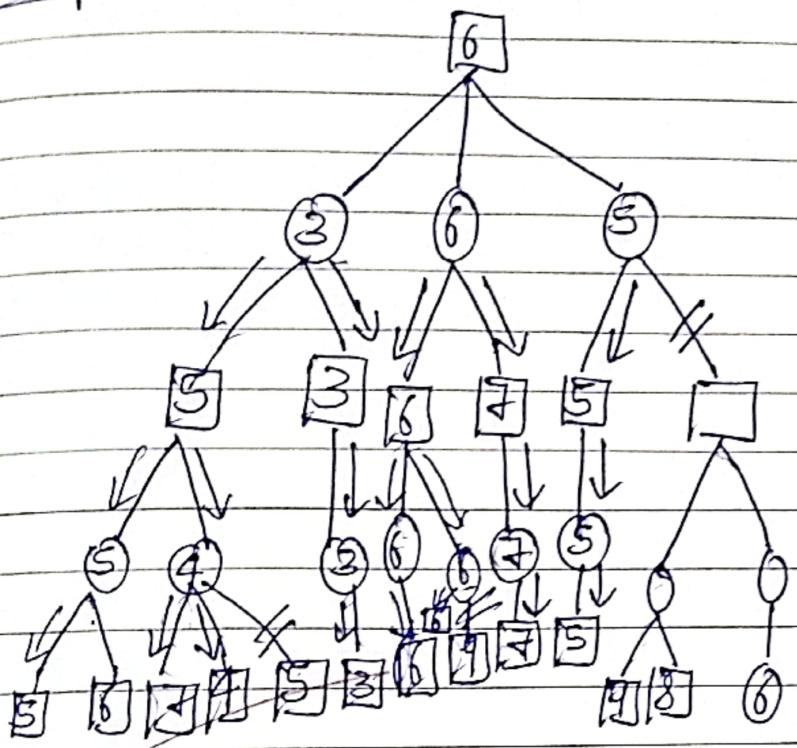
Input:-

□ max

O min



Output



880

3/12/24