22-10-24

Implement Iterative Deepening search algorithm

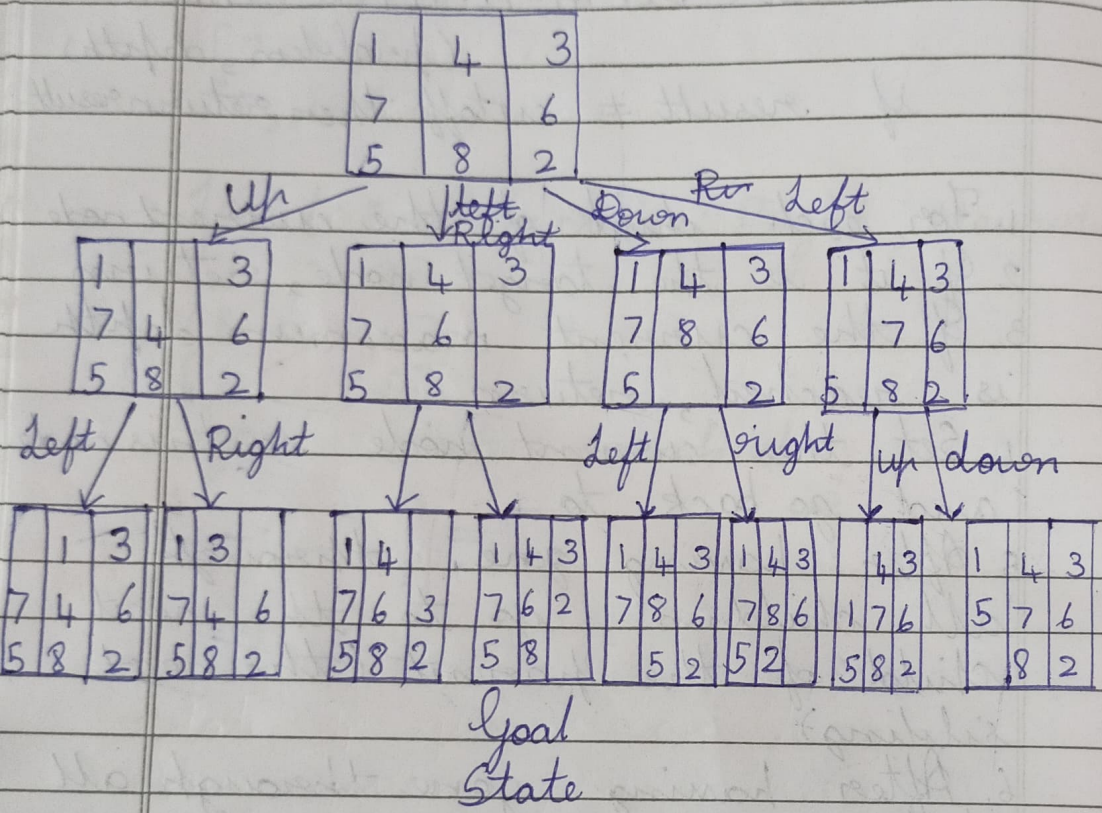
function ITERATIVE-DEEPENING-SEARCH (problem) returns a solution, or failure
for depth = 0 to ∞ do
result ← DEPTH-LIMITED-SEARCH (problem, depth)
if result ≠ cutoff then return result

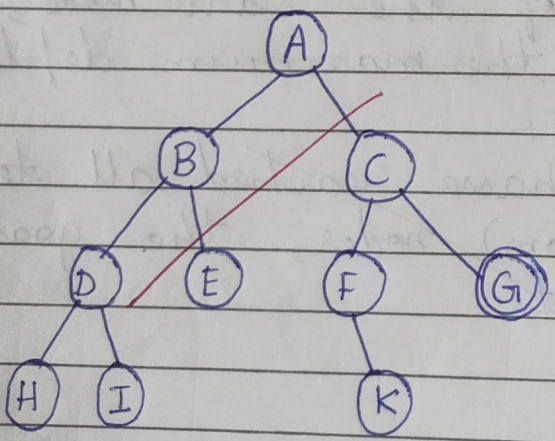
1. For each child of the current node
2. If it is the target node, return
3. If the current maximum depth is reached, return
4. Set the current node to this node and go back to 1.
5. After having gone through all children, go to the next child of the parent (the next sibling)
6. After having gone through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

Initial State Goal State

2	8	3
1	4	3
7		6
5	8	2

1	4	3
7	6	2
5	8	





I Iteration A

II Iteration A, B, C

III Iteration A, B, D, E, C, F, G

Date _____ Page 26

22-10-24

Implement Hill Climbing search algorithm to solve N-Queens problem (N=4)

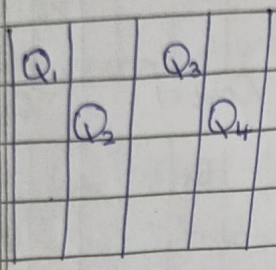
Hill-climbing search algorithm

function HILL-CLIMBING (problem)
returns a state that is a local maximum

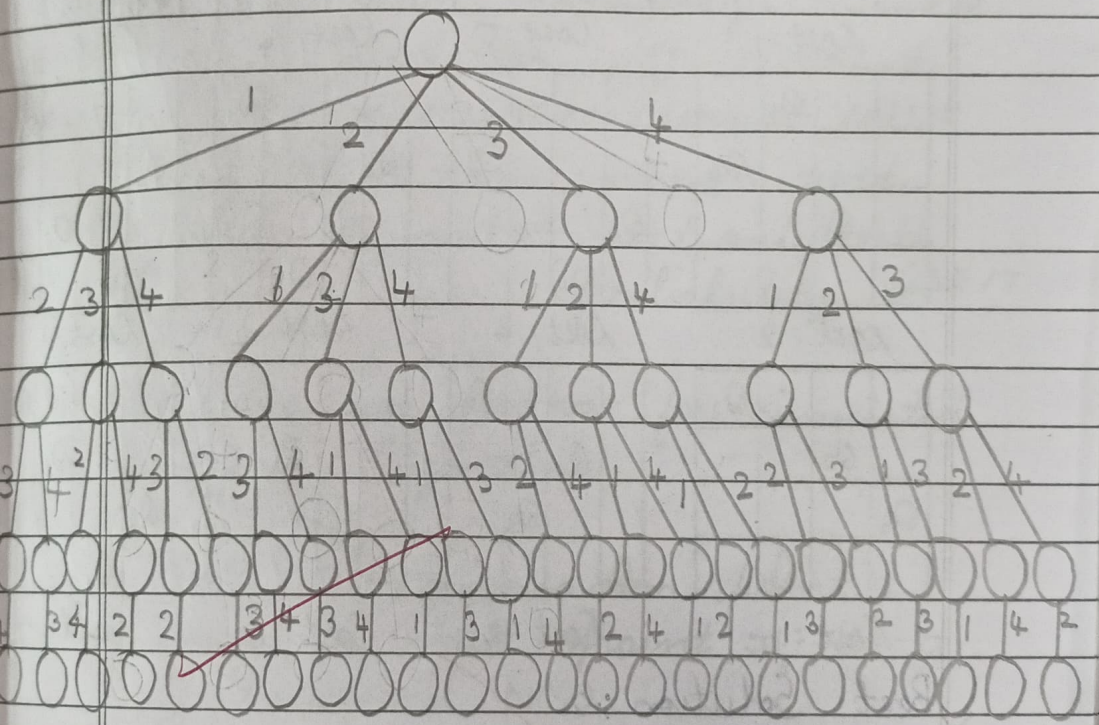
current ← MAKE-NODE(problem.INITIAL-STATE)
loop do
neighbor ← a highest-valued successor of current
if neighbor.VALUE ≤ current.VALUE then return current.STATE
current ← neighbor

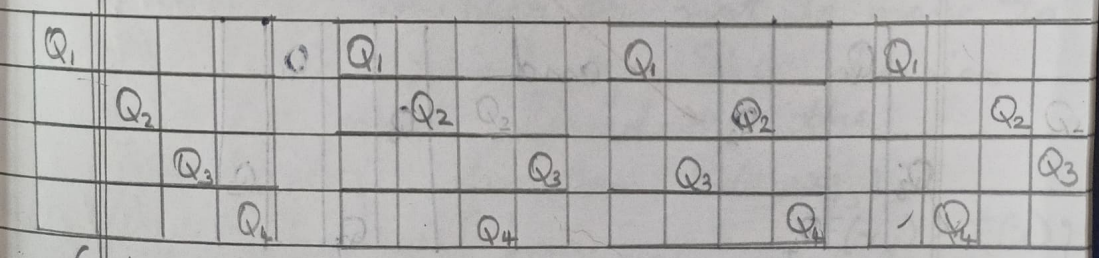
- State: 4 queens on the board. One queen per column.
  - Variables: x_0, x_1, x_2, x_3 where x_i is the ~~row~~ row position of the queen in column i. Assume that there is one queen per column
  - Domain for each variable: $x_i \in \{0,1,2,3\}, \forall i$
- Initial state: a random state.
- Goal state: 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.

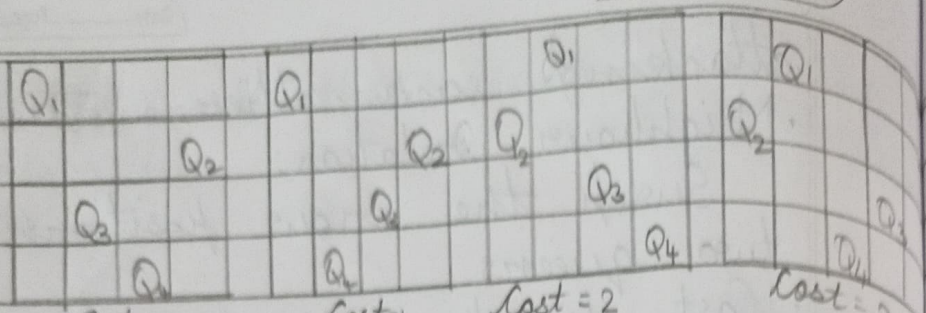


cost = 2 + 3 + 3 + 2 = 10

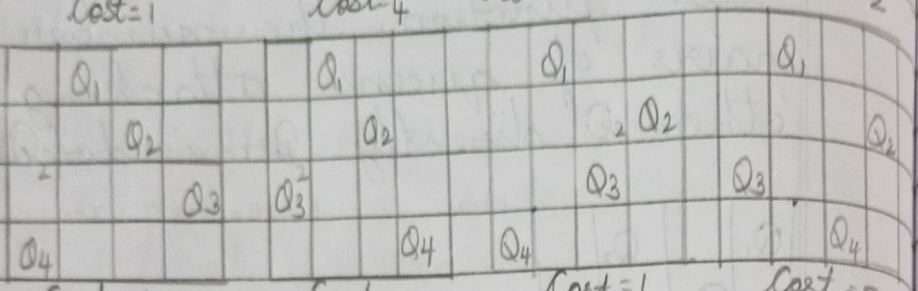




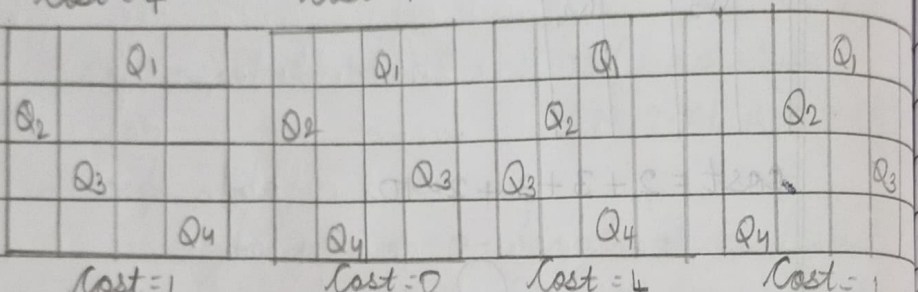
Cost = 6 Cost = 2 Cost = 2 Cost = 1



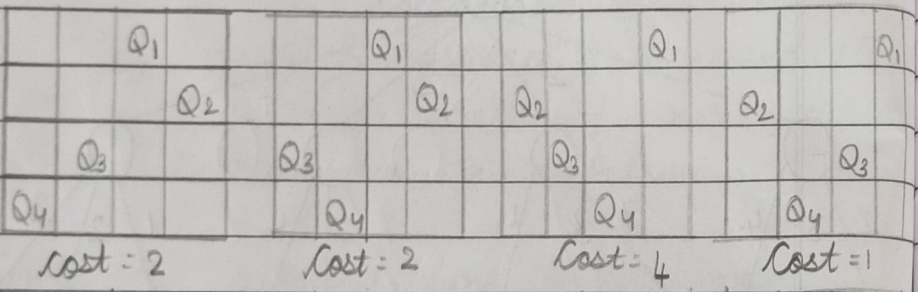
Cost = 1 Cost = 4 Cost = 2 Cost = 2



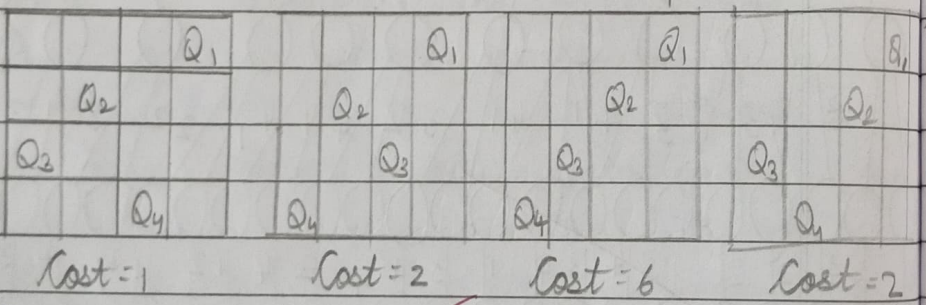
Cost = 4 Cost = 1 Cost = 1 Cost = 0



Cost = 1 Cost = 0 Cost = 4 Cost = 1

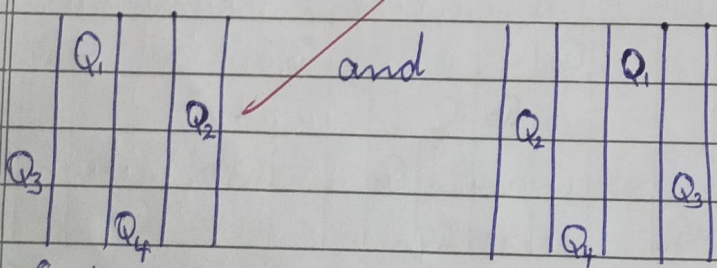


Cost = 2 Cost = 2 Cost = 4 Cost = 1



Cost = 1 Cost = 2 Cost = 6 Cost = 2

Best Solution :-2



Cost = 0 Cost = 0

29-10-24

WAP to implement Simulated Annealing Algorithm

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 $e^{\Delta E / T}$

Steps ~~to use~~ ~~increase~~ for simulated annealing Algorithm

1. Start at a random point x.
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 $P(x, x_j, T)$

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

4. Reduce T.

~~Steps~~

Python Code

```
import mlrose_hiive as mlrose
import numpy as np

def queensmax(position):
    queennotattacking=0
    for i in range(len(position)-1):
        noattack=0
        for j in range(i+1, len(position
            if (position[j] != position[i])
            and (position[j] != position[i
                + (j-i)) and
            position[j] != position[i]-
                (j-i)):
                noattack+=1
            if noattack == len(position)
                -1-i:
                queennotattacking+=1
    return queennotattacking

try:
    ui = input("Enter the initial
        position as 8 comma-separated
        integers (e.g., '0,1,2,3,4,5,6,7'):
initialpos = np.array([int(x) for
        x in initialpos):
```

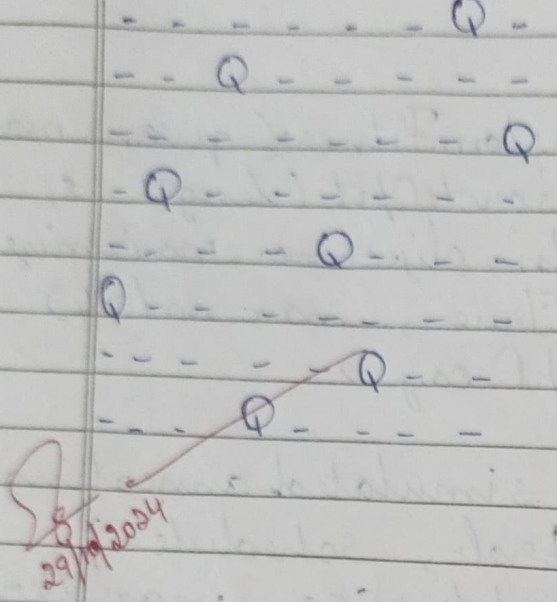
```
// take input

obj = mlrose.CustomFitness(queens_max)
prob = mlrose.DiscreteOpt(length = 8,
        fitness_fn = objective, maximize
        = True, max_val = 4)
T = mlrose.ExpDecay()

result = mlrose.simulated_annealing
    (problem = prob, schedule = T,
     max_attempts = 500, max_iters =
     5000, init_state = initial_pos)

best_state = result[0]
best_fitness = result[1]

print(best_state)
print(best_fitness + 1)

print("\n Best State Diagram:")
// print best_state
```

O/P:-

Enter the initial position :
0, 1, 2, 3, 4, 5, 6, 7
The best position found is:
[5 3 1 7 4 6 0 2]
The number of queens that are not attacking each other is : 8.0

Date Page

Best State Diagram:



29/9/2024

SURYA Gold
Date Page

12-11-24

Lab No. 6

Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.

Propositional Inference: Enumeration Method

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

A	B	C	AVC	BV¬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

function TT-ENTAILS?(KB, α) returns 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, {})
function TT-CHECK-ALL(KB, α, symbols, model)
  returns 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 returns 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}))

O/P:-

Combinations where both KB and α (AVB) are true

A	B	C
0	1	1
1	0	0
1	1	0
1	1	1

13/11/2024

19-11-24

Lab No. 7

Implement Unification in First Order Logic (FOL)

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 $\{(\Psi_2/\Psi_1)\}$
c) Else if Ψ_2 is a variable,
   a. If Ψ_2 occurs in Ψ_1 then return FAILURE
   b. Else return $\{(\Psi_1/\Psi_2)\}$
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_2.

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 return 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.

Ex 1:-
P(x, F(y)) - ①
P(a, F(g(x))) - ②

① and ② an identical if
x is replaced with a in ①
P(a, F(y))
if y is replaced with g(x)
P(a, F(g(x)))
Unification is achieved

Ex 2:-
Q(a, g(x, a), f(y)) - ①
Q(a, g(f(b), a), x) - ②

Replace x with f(b) in ②

Q(a, g(f(b), a), f(y)) - ①
~~Replace f(y) x with x f(y) in x~~

~~Q(a, g(f(b), a), x)~~
Q(a, g(f(b), a), f(y)) -②
They are not unified

Ex 3:-
$\psi_1 = P(b, x, f(g(z)))$ -①
$\psi_2 = P(z, f(y), f(y))$ -②
Replace z in ② as b
$\psi_2 = P(b, f(y), f(y))$
Replace x in ① as f(y)
$\psi_1 = P(b, f(y), f(g(z)))$
Replace y in ② as g(z)
$\psi_2 = P(b, f(y), f(g(z)))$

Unification is achieved.

Ex 4:-
$\psi_1 = P(f(a), g(y))$
$\psi_2 = P(x, x)$
Failure as x cannot be replaced by f(a), g(y)

O/P:-
i) Enter the first term : ['P', 'b', '?x', ['f', ['g', '?z']]]
Enter the second term : ['P', '?z', ['f', '?y'], ['f', '?y']]
Unification successful!
Substitution : {'?z' : 'b', '?x' : ['f', '?y'], '?y' : ['g', '?z']}

2. Enter the first term:
['P', ['f', ['a']], ['g', ['?y']]]
Enter the second term: ['P', '?x', '?x']
Unification failed: Predicate symbols don't match: g != f

19/11/24

26-11-24

8. Create a knowledge base consisting of FOL statements and prove the given query using forward reasoning

Algorithm

function 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 $\leftarrow$ { }
for each rule in KB do
$(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(rule)
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
$\phi \leftarrow$ UNIFY(q', α)
if ϕ is not fail then return ϕ
add new to KB
return false

It is a crime for an American to sell weapons to hostile nations
American (p) ∧ Weapon(q) ∧ Sells(p, q, r) ∧ Hostile(r) ⇒ Criminal(p)
Country A has some missiles.
∃ x Owns(A, x) ∧ Missile

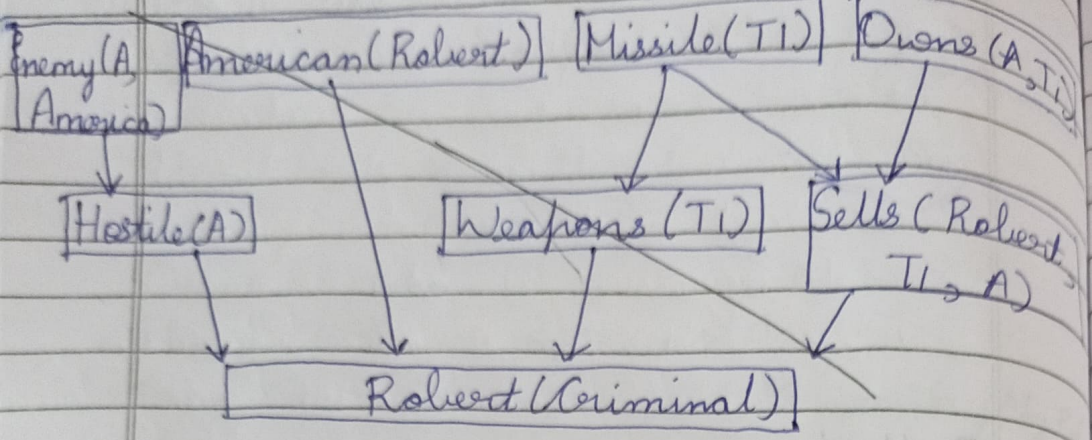
Owns (A, T1)
Missile (T1)
All of the missiles were sold to country A by Robert
∀ x Missile (x) ∧ Owns (A, x)
⇒ Sells (Robert, x, A)
Missiles are weapons
Missile (x) ⇒ Weapon (x)

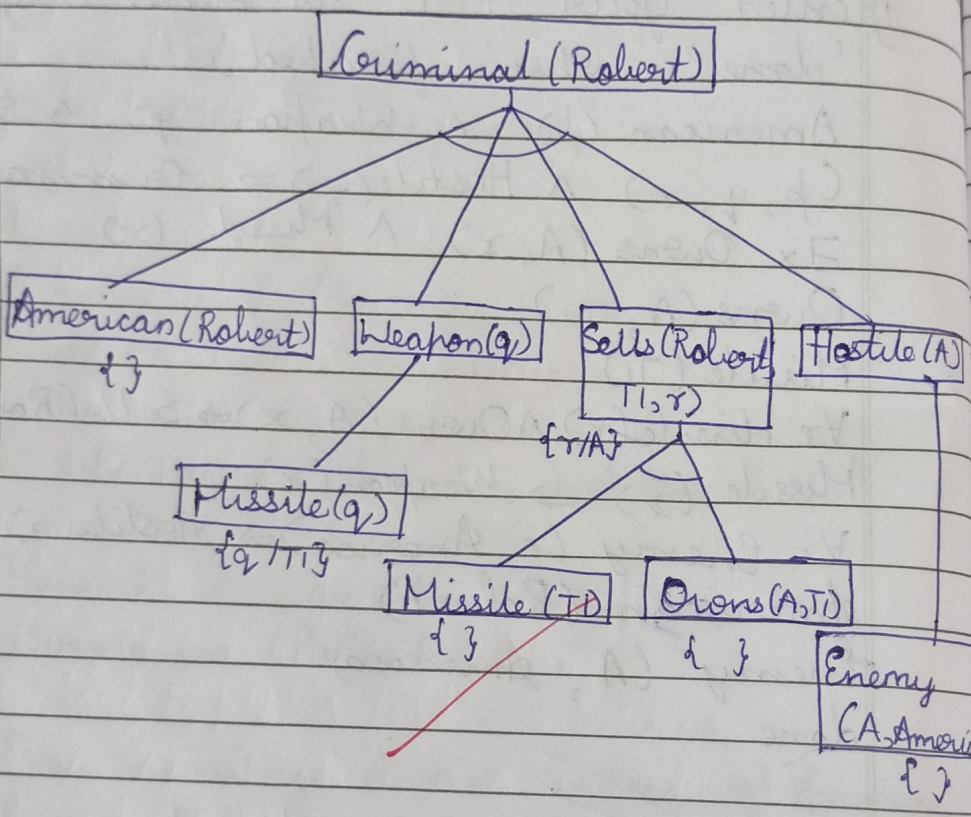
Enemy of America is known as hostile

∀ x Enemy (x, America) ⇒ Hostile (x)

Robert is an American
American (Robert)

The country A, an enemy of America
Enemy (A, America)


Enemy(A, America)
American(Robert)
Missile(T1)
Owns(A,T1)
Hostile(A)
Weapons(T1)
Sells(Robert, T1, A)
Robert(Criminal)


Criminal(Robert)
American(Robert)
{}
Weapon(q)
Sells(Robert, T1, r)
{r/A}
Hostile(A)
Missile(q)
{q/T1}
Missile(T1)
{}
Owns(A,T1)
{}
Enemy(A, America)
{}

Output:-

1) Enter your FOL statements (type 'done' when finished):
American (p) ∧ Weapon (q) ∧ Sells (p, q, r) ∧ Hostile(r) ⇒ Criminal (p)
∃x Owns (A, x) ∧ Missile (x)
Owns (A, T1)
Missile (T1)
∀x Missile(x) ∧ Owns (A, x) ⇒ Sells (Robert, x, A)
Missile (x) ⇒ Weapon (x)
∀x Enemy (x, America) ⇒ Hostile(x)
American (Robert)
Enemy (A, America)
done

Enter the query to prove:
Criminal (Robert)
Proven : Criminal (Robert)

3/12/2021

26-11-24

9. Convert a given first order logic statement into Conjuctive Normal Form (CNF) to Resolution

Basic steps for proving a conclusion S given premises

Premise$_1$, - - -, Premise$_n$

(all expressed in FOL):

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 unifications
   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.

Given the KB or Premises:
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.

Representation in FOL
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. $\forall x: alive(x) \rightarrow \neg killed(x)$
h. $likes(John, Peanuts)$

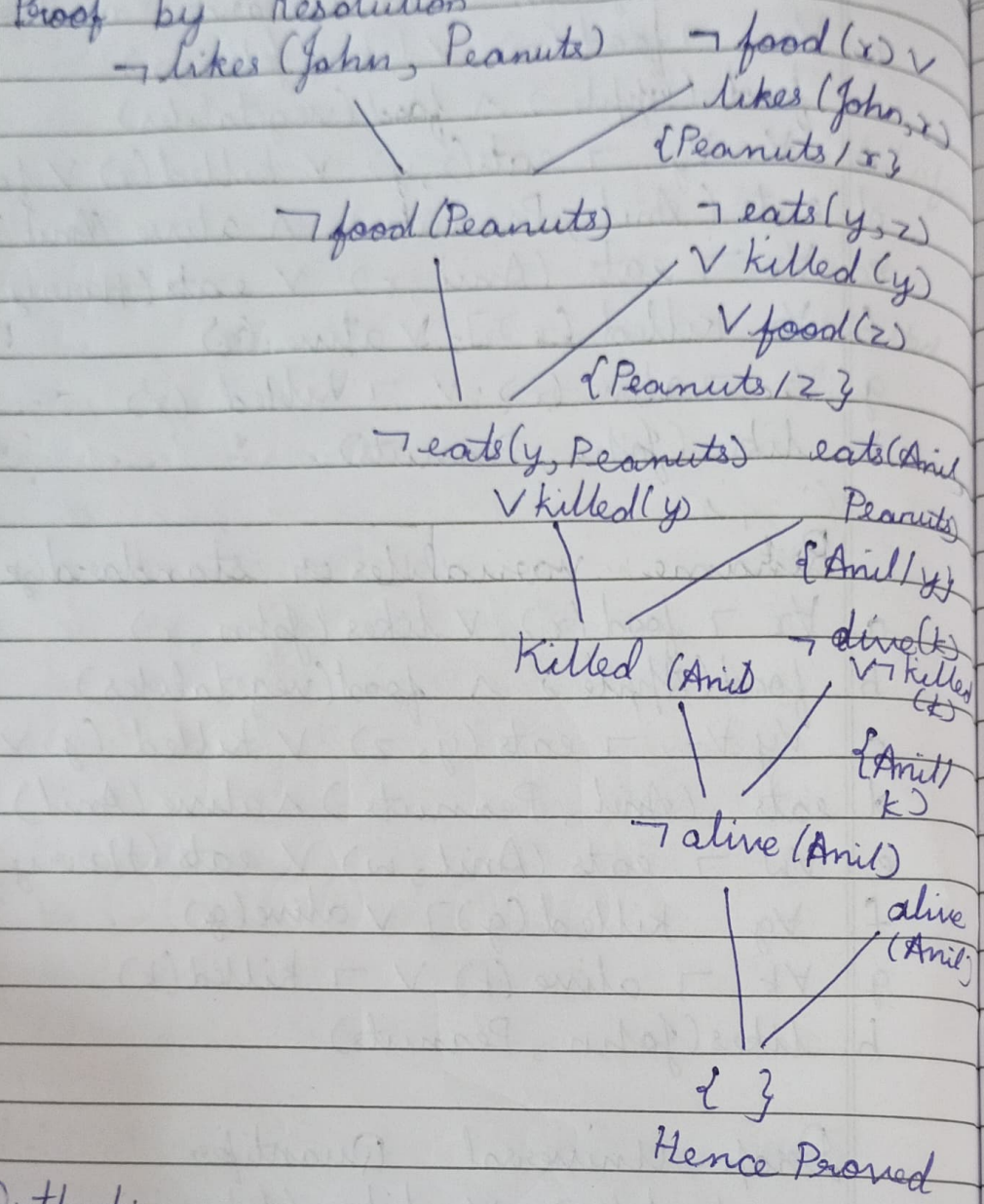
Eliminate implication $\alpha \Rightarrow \beta$ with $\neg \alpha \vee \beta$
a. $\forall x \neg food(x) \vee likes(John, x)$
b. $food(Apple) \wedge food(vegetables)$
c. $\forall x \forall y \neg [eats(x, y) \wedge \neg killed(x)] \vee food(y)$
d. $eats(Anil, Peanuts) \wedge alive(Anil)$
e. $\forall x \neg eats(Anil, x) \vee eats(Harry, x)$
f. $\forall x \neg [\neg killed(x)] \vee alive(x)$
g. $\forall x \neg alive(x) \vee \neg killed(x)$
h. $likes(John, Peanuts)$

Move negation ($\neg$) inwards and rewrite
a. $\forall x \neg food(x) \vee likes(John, x)$
b. $food(Apple) \wedge food(vegetables)$
c. $\forall x \forall y \neg eats(x, y) \vee killed(x) \vee food(y)$
d. $eats(Anil, Peanuts) \wedge alive(Anil)$
e. $\forall x \neg eats(Anil, x) \vee eats(Harry, x)$
f. $\forall x killed(x)] \vee alive(x)$
g. $\forall x \neg alive(x) \vee \neg killed(x)$
h. $likes(John, Peanuts)$.

Rename variables or standardize variables
a. $\forall x \neg food(x) \vee likes(John, x)$
b. $food(Apple) \wedge food(vegetables)$
c. $\forall y \forall z \neg eats(y, z) \vee killed(y) \vee food$
d. $eats(Anil, Peanuts) \wedge alive(Anil)$
e. $\forall w \neg eats(Anil, w) \vee eats(Harry, w)$
f. $\forall g killed(g)] \vee alive(g)$
g. $\forall k \neg alive(k) \vee \neg killed(k)$
h. $likes(John, Peanuts)$

Drop Universal Quantifier
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. $likes(John, Peanuts)$.

Proof by Resolution



Hence Proved

Output:-

Derived Fact : Food(Peanuts)

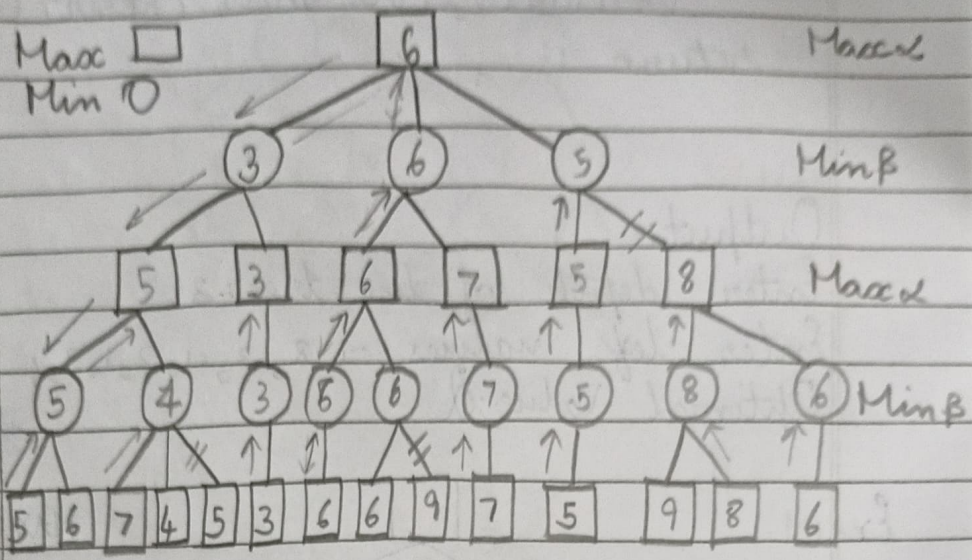
Derived Fact : Likes(John, Peanuts)

Proven : Likes(John, Peanuts)

3/10/2024

24

10. Alpha Beta Pruning Algorithm



$\alpha >= \beta$

Algorithm:

function MINIMAX-DECISION(state) returns an action

return arg max a ∈ ACTIONS(s)
MIN-VALUE (RESULT (state, a))

function MAX-VALUE (state) returns a utility value

if TERMINAL-TEST (state) then return UTILITY (state)

$v \leftarrow -\infty$

for each a in ACTIONS (state) do
$v \leftarrow$ MAX (v, MIN-VALUE (RESULT (s, a)))

return v

function MIN-VALUE (state) returns a utility value

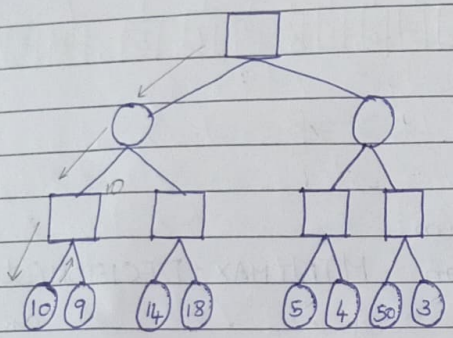
if TERMINAL-TEST (state) then return UTILITY (state)

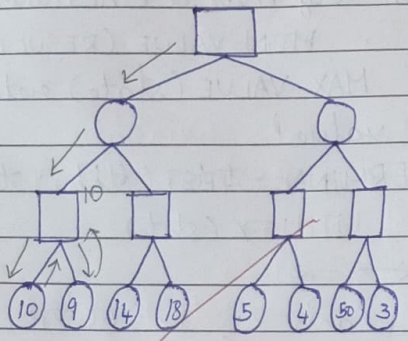
$v \leftarrow \infty$

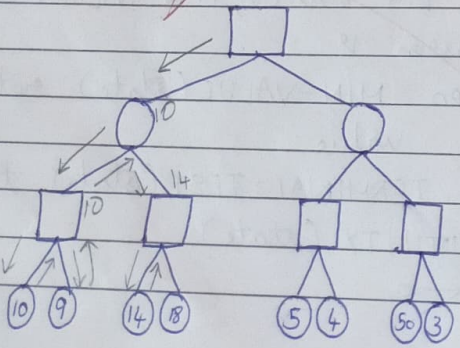
for each a in ACTIONS(state) do
v ← MIN(v, MAX-VALUE(RESULT(s, a)))
return v

Output:-
Enter depth of the tree: 3
Enter leaf values: -1, 8, -3, -1, 2, 1, 3, 4
Optimal Value: 2

Ex:







3/12/2024