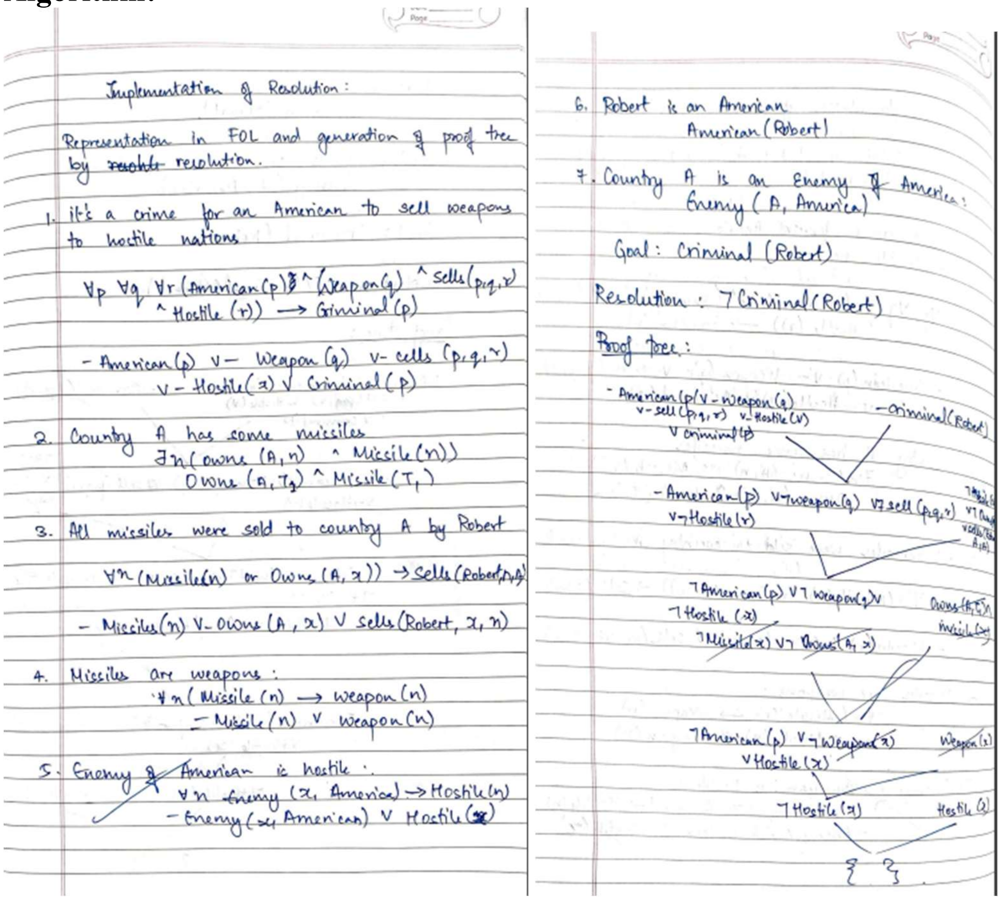
Program9:

Create a knowledge base consisting of first order logic statements and prove the given query using Resolution.

Algorithm:



Implementation of Resolution:

Representation in FOL and generation of proof tree by ~~resolute~~ resolution.

1. it's a crime for an American to sell weapons to hostile nations

∀p ∀q ∀r (American(p) ^ Weapon(q) ^ Sells(p,q,r) ^ Hostile(r)) → Criminal(p)

- American(p) V - Weapon(q) V - Sells(p,q,r) V - Hostile(r) V Criminal(p)

2. Country A has some missiles

∃x(Owns(A,x) ^ Missile(x))

Owns(A, T1) ^ Missile(T1)

3. All missiles were sold to country A by Robert

∀x (Missile(x) or Owns(A, x)) → Sells(Robert,x,A)

- Missiles(x) V - Owns(A, x) V Sells(Robert, x, x)

4. Missiles are weapons:

∀x(Missile(x) → Weapon(x))

- Missile(x) V Weapon(x)

5. Enemy of American is hostile:

∀x Enemy (x, America) → Hostile(x)

- Enemy(x, American) V Hostile(x)

6. Robert is an American

American(Robert)

7. Country A is an Enemy of America:

Enemy(A, America)

Goal: Criminal(Robert)

Resolution: ¬Criminal(Robert)

Proof tree:

Code:

```
from itertools import combinations

def unify_sentences_var(var, x, theta):
    if var in theta:
        return unify_sentences(theta[var], x, theta)
    elif x in theta:
        return unify_sentences(var, theta[x], theta)
    else:
        theta[var] = x
        return theta

def resolve(sentence1, sentence2):
    resolvents = []
    for predicate1 in sentence1:
        for predicate2 in sentence2:
            theta = unify_sentences(predicate1, negate(predicate2))
```

```
Knowledge_Base = {
    frozenset({('Mother', 'Leela', 'Oshin')}),
    frozenset({('Alive', 'Leela')}),
    frozenset({('not','Mother', 'x','y')}),
    frozenset({('Parent','x','y')}),
    frozenset({('not','Parent', 'w', 'z')}),
    frozenset({('not','Alive','w','z')}),
    frozenset({('Older','w','z')}),
}

query = ('Older', 'Leela', 'Older')
result = proof_by_resolution(Knowledge_Base, query)
if result:
    print("Leela is older than Oshin.\nProved by resolution.")
else:
    print("Cannot prove. Leela is not older than Oshin.")
```

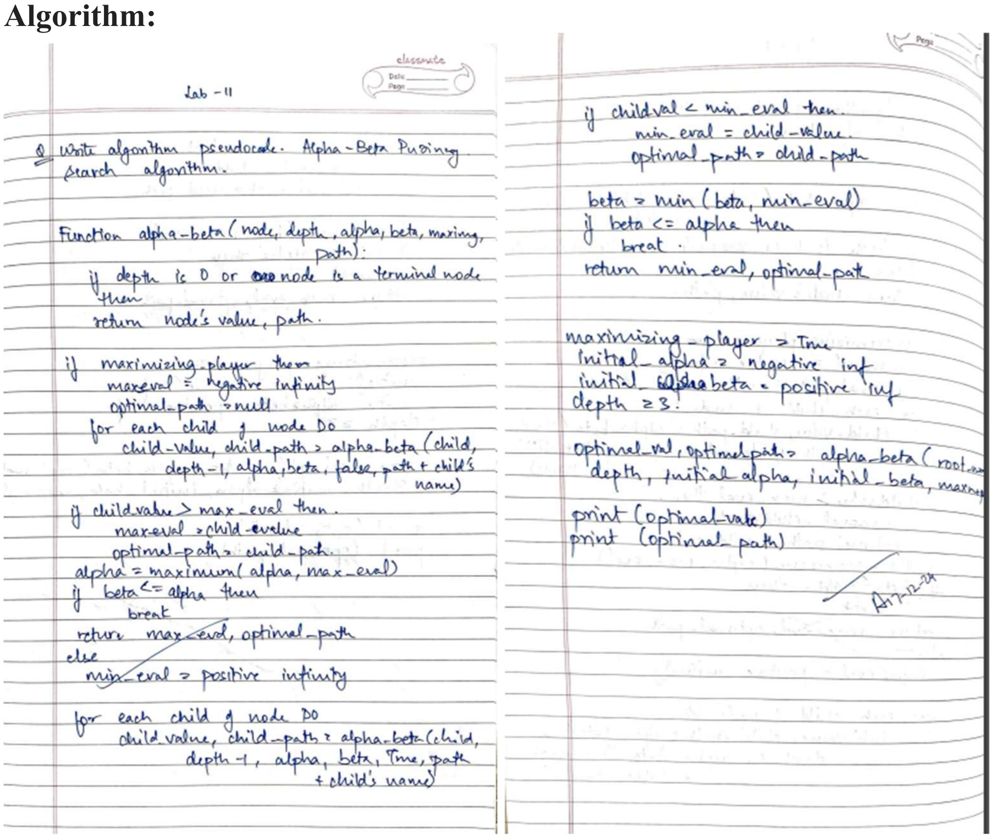
Output Snapshot:

```
Leela is older than Oshin.
Proved by resolution.
```

<u>Program10:</u>

Implement Alpha-Beta Pruning.

Algorithm:


Lab - 11

Q Write algorithm pseudocode. Alpha-Beta Pruning search algorithm.

```
Function alpha-beta(node, depth, alpha, beta, maxing,
                                         path):
   if depth is 0 or node is a terminal node
   then
   return node's value, path.

if maximizing-player then
    maxeval = negative infinity
    optimal-path = null
  for each child of node Do
      child-value, child-path = alpha-beta(child,
            depth-1, alpha, beta, false, path + child's
                                              name)
 if childvalue > max_eval then.
     max-eval = child-evalue
     optimal-path = child-path.
 alpha = maximum(alpha, max_eval)
 if beta <= alpha then
      break
 return max_eval, optimal-path
else
  min_eval = positive infinity

 for each child of node DO
      child-value, child-path = alpha-beta(child,
            depth-1, alpha, beta, True, path
                                 + child's name)

 if childval < min_eval then.
      min_eval = child-value.
      optimal_path = child-path

 beta = min(beta, min-eval)
 if beta <= alpha then
    break.
 return min_eval, optimal-path


maximizing-player = True
 initial_alpha = negative inf
 initial_beta = positive inf
depth = 3.

optimal_val, optimalpath = alpha_beta(root...
 depth, initial alpha, initial_beta, maxi...

print (optimal-vale)
print (optimal_path)
```

A 17-12-24