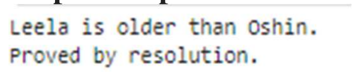
```
Knowledge_Base = {
    frozenset({('Mother', 'Leela', 'Oshin')}),
    frozenset({('Alive', 'Leela')}),
    frozenset({('not','Mother', 'x','y')}),
    frozenset({('Parent','x','y')}),
    frozenset({('not','Parent', 'w', 'z')}),
    frozenset({('not','Alive','w','z')}),
    frozenset({('Older','w','z')}),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

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:

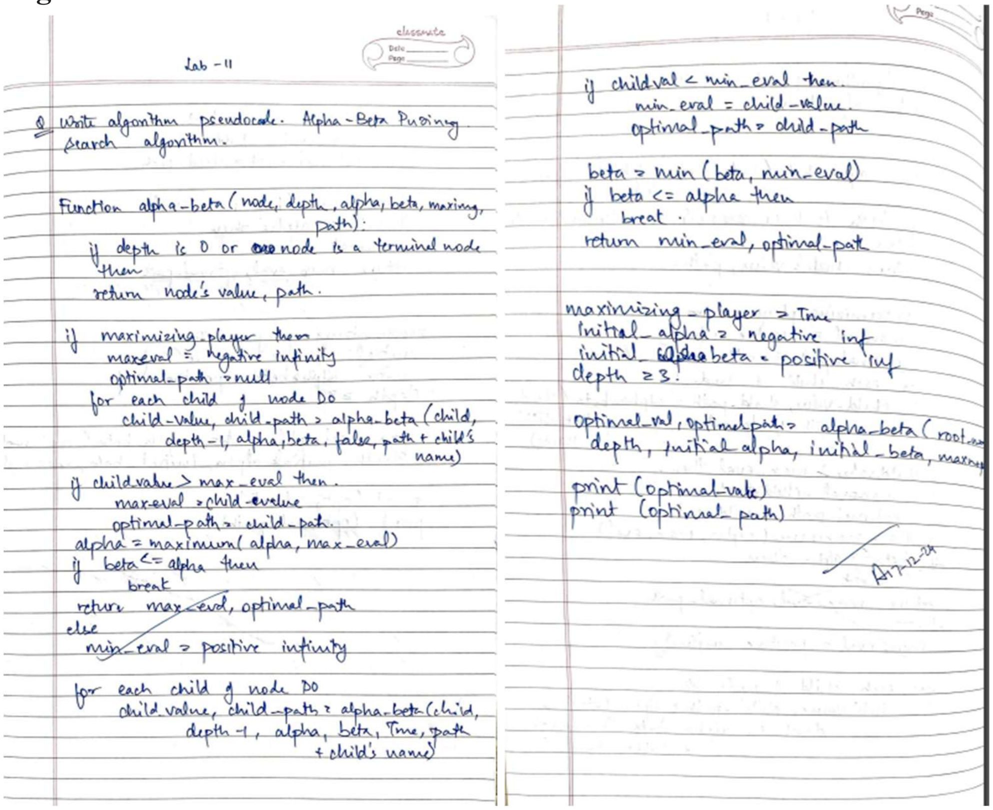


```
Leela is older than Oshin.
Proved by resolution.
```

Program10:

Implement Alpha-Beta Pruning.

Algorithm:



Lab - 11

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

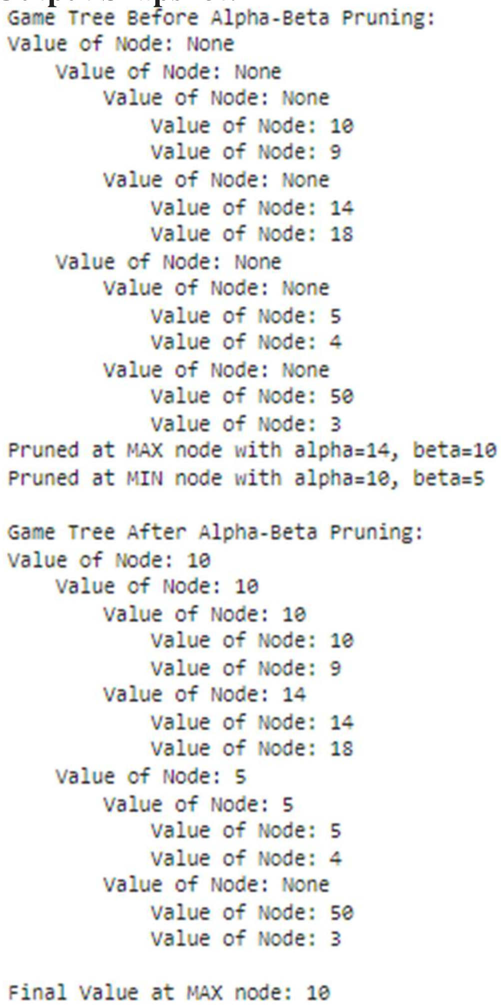
```
Function alpha-beta(node, depth, alpha, beta, maxing,
                                        path):
    if depth is 0 or node is a terminal node
    then
    return node's value, path.

  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 child.value > 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_~~alpha~~beta = positive inf
depth = 3!

optimal_val, optimalpath = alpha_beta(root,
  depth, initial alpha, initial_beta, maxing

print (optimal_vale)
print (optimal_path)
```

A 17-12-24

Code:

```
class Node:
    def __init__(self, value=None, children=None):
        self.value = value
        self.children = children if children else []

def alpha_beta_pruning(node, depth, alpha, beta, maximizing_player):
    if not node.children or depth == 0:
        return node.value

    if maximizing_player:
        max_eval = float('-inf')
        for child in node.children:
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                print(f"Pruned at MAX node with alpha={alpha}, beta={beta}")
                break
        node.value = max_eval
        return max_eval
    else:
        min_eval = float('inf')
        for child in node.children:
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                print(f"Pruned at MIN node with alpha={alpha}, beta={beta}")
                break
        node.value = min_eval
        return min_eval


def print_tree(node, level=0):
    print("  " * level *2 + f"Value of Node: {node.value}")
    for child in node.children:
        print_tree(child, level + 1)


if __name__ == "__main__":
    tree = Node(None, [
        Node(None, [
            Node(None,[Node(10),Node(9)]),
            Node(None, [Node(14),Node(18)])
        ]),
        Node(None, [
            Node(None, [Node(5),Node(4)]),
            Node(None, [Node(50),Node(3)])
        ])
    ])

    print("Game Tree Before Alpha-Beta Pruning:")
    print_tree(tree)
    final_value = alpha_beta_pruning(tree, depth=3, alpha=float('-inf'), beta=float('inf'),
maximizing_player=True)
    print("\nGame Tree After Alpha-Beta Pruning:")
    print_tree(tree)
    print("\nFinal Value at MAX node:", final_value)
```

Output Snapshot:



```
Game Tree Before Alpha-Beta Pruning:
Value of Node: None
    Value of Node: None
        Value of Node: None
            Value of Node: 10
            Value of Node: 9
        Value of Node: None
            Value of Node: 14
            Value of Node: 18
    Value of Node: None
        Value of Node: None
            Value of Node: 5
            Value of Node: 4
        Value of Node: None
            Value of Node: 50
            Value of Node: 3
Pruned at MAX node with alpha=14, beta=10
Pruned at MIN node with alpha=10, beta=5

Game Tree After Alpha-Beta Pruning:
Value of Node: 10
    Value of Node: 10
        Value of Node: 10
            Value of Node: 10
            Value of Node: 9
        Value of Node: 14
            Value of Node: 14
            Value of Node: 18
    Value of Node: 5
        Value of Node: 5
            Value of Node: 5
            Value of Node: 4
        Value of Node: None
            Value of Node: 50
            Value of Node: 3

Final Value at MAX node: 10
```