```python
def searchSolution():
    goal = Node(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)
    neuronal_network = NeuronalNetwork("nn.h5")  # importa red neuronal

    # binary heap de nodos ordenados por valor f
    # binary heap de nodos con preferencia según la red neuronal
    preferred_heap = BinaryHeap()

    normal_heap = BinaryHeap()  # binary heap de nodos sin preferencia según la red neuronal

    # se crea el nodo del estado inicial
    # por ejemplo:
    initial_node = Node(8, 14, 2, 3, 4, 11, 15, 0, 1, 9, 10, 12, 13, 5, 6, 7)

    # se calcula la heuristica desde el punto inicial
    heuristic = calculateHeuristic(initial_node)
    initial_node.g = 0
    initial_node.f = initial_node.g + heuristic # valor de f siempre es igual a g + h

    states_visited = []  # lista de estados ya visitados

    normal_heap.insert(initial_node)

    current = 0
    # mientras exista de donde sacar un nodo
    while not (normal_heap.isEmpty() and preferred_heap.isEmpty()):
        # iteramos 1 a 1 de entre los heaps para sacar un nodo
        if current == 0:
            heap = normal_heap
        else:
            heap = preferred_heap
        current = (current + 1) % 2  # alternamos current entre 0 y 1
        node = heap.extract()  # retiramos el nodo con menor valor f
        if not node:
            continue  # volver al while
        if node == goal:
            return node  # retornamos el objetivo y salimos del while

        # obtenemos los estados posibles con un movimiento
        node_successors = node.getSuccessors()
        for successor in node_successors:

            is_new = successor in states_visited  # primera vez que visita el estado?
            is_shorter_path = False
            if is_new:
                successor.g = node.g + 1
                states_visited.insert(successor)
            elif successor.g < node.g + 1: # encontramos una ruta más barata para un estado?
                is_shorter_path = True

            if is_new or is_shorter_path:
                successor.f = successor.g + calculateHeuristic(successor)
                # si no hay hijo del nodo o encontramos un mejor hijo
                if (not node.child or successor.f < node.child.f):
                    node.child = successor  # dejamos registro del movimiento
                prediction_posibility = neuronal_network.predictNode(node, successor)
                if prediction_posibility > 0.5:
                    # inserta y ordena el binaryHeap segun valor f = g + h
                    preferred_heap.insert(successor)
                elif prediction_posibility > 0.1:
                    # inserta y ordena el binaryHeap segun valor f = g + h
                    normal_heap.insert(successor)

    return initial_node
```