CODE:

```
import math

class st():
      def __init__(self, cannL, misL, b, cannR, misR):
      self.cannL = cannL
      self.misL = misL
      self.b = b
      self.cannR = cannR
      self.misR = misR
      self.prev = None
      def isGoal(self):
      if self.cannL == 0 and self.misL == 0:
            return True
      else:
            return False
      def isVal(self):
      if self.misL >= 0 and self.misR >= 0 \
                  and self.cannL >= 0 and self.cannR >= 0 \
                  and (self.misL == 0 or self.misL >= self.cannL) \
                  and (self.misR == 0 or self.misR >= self.cannR):
            return True
      else:
            return False
      def __eq__(self, other):
      return self.cannL == other.cannL and self.misL == other.misL \
                  and self.b == other.b and self.cannR == other.cannR \
                  and self.misR == other.misR
      def __hash__(self):
      return hash((self.cannL, self.misL, self.b, self.cannR, self.misR))
def next1(currSt):
      children = [];
      if currSt.b == 'left':
      newSt = st(currSt.cannL, currSt.misL - 2, 'right',
                              currSt.cannR, currSt.misR + 2)

      if newSt.isVal():
            newSt.prev = currSt
            children.append(newSt)
      newSt = st(currSt.cannL - 2, currSt.misL, 'right',
                              currSt.cannR + 2, currSt.misR)

      if newSt.isVal():
            newSt.prev = currSt
            children.append(newSt)
      newSt = st(currSt.cannL - 1, currSt.misL - 1, 'right',
                              currSt.cannR + 1, currSt.misR + 1)
```

```
if newSt.isVal():
      newSt.prev = currSt
      children.append(newSt)
newSt = st(currSt.cannL, currSt.misL - 1, 'right',
                        currSt.cannR, currSt.misR + 1)

if newSt.isVal():
      newSt.prev = currSt
      children.append(newSt)
newSt = st(currSt.cannL - 1, currSt.misL, 'right',
                        currSt.cannR + 1, currSt.misR)

if newSt.isVal():
      newSt.prev = currSt
      children.append(newSt)
else:
newSt = st(currSt.cannL, currSt.misL + 2, 'left',
                        currSt.cannR, currSt.misR - 2)

if newSt.isVal():
      newSt.prev = currSt
      children.append(newSt)
newSt = st(currSt.cannL + 2, currSt.misL, 'left',
                        currSt.cannR - 2, currSt.misR)

if newSt.isVal():
      newSt.prev = currSt
      children.append(newSt)
newSt = st(currSt.cannL + 1, currSt.misL + 1, 'left',
                        currSt.cannR - 1, currSt.misR - 1)

if newSt.isVal():
      newSt.prev = currSt
      children.append(newSt)
newSt = st(currSt.cannL, currSt.misL + 1, 'left',
                        currSt.cannR, currSt.misR - 1)

if newSt.isVal():
      newSt.prev = currSt
      children.append(newSt)
newSt = st(currSt.cannL + 1, currSt.misL, 'left',
                        currSt.cannR - 1, currSt.misR)

if newSt.isVal():
      newSt.prev = currSt
      children.append(newSt)
return children
```

```
def bfs():
      inistate = st(3,3,'left',0,0)
      if inistate.isGoal():
      return inistate
      frontier = list()
      explored = set()
      frontier.append(inistate)
      while frontier:
      state = frontier.pop(0)
      if state.isGoal():
            return state
      explored.add(state)
      children = next1(state)
      for child in children:
            if (child not in explored) or (child not in frontier):
            frontier.append(child)
      return None
def printSol(sol):
      path = []
      path.append(sol)
      prev = sol.prev
      while prev:
            path.append(prev)
            prev = prev.prev
      for t in range(len(path)):
            state = path[len(path) - t - 1]
            print ("(" + str(state.cannL) + " , " + str(state.misL) \
                  + " , " + state.b + " , " + str(state.cannR) + " , " + \
                        str(state.misR) + ")")
sol = bfs()
print ("(cannL,misL,b,cannR,misR)")
printSol(sol)
```

OUTPUT:

```
(cannL,misL,b,cannR,misR)
(3 , 3 , left , 0 , 0)
(1 , 3 , right , 2 , 0)
(2 , 3 , left , 1 , 0)
(0 , 3 , right , 3 , 0)
(1 , 3 , left , 2 , 0)
(1 , 1 , right , 2 , 2)
(2 , 2 , left , 1 , 1)
(2 , 0 , right , 1 , 3)
(3 , 0 , left , 0 , 3)
(1 , 0 , right , 2 , 3)
```

```
(1 , 1 , left , 2 , 2)
(0 , 0 , right , 3 , 3)
```