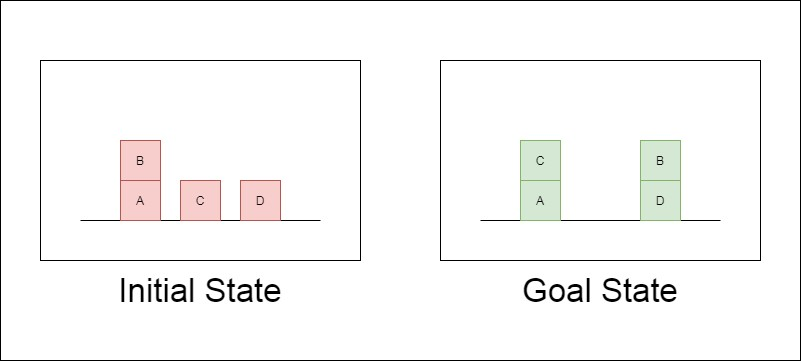
Block World Problem

**What is Blocks World Problem?**

This is how the problem goes — There is a table on which some blocks are placed. Some blocks may or may not be stacked on other blocks. We have a robot arm to pick up or put down the blocks. The robot arm can move only one block at a time, and no other block should be stacked on top of the block which is to be moved by the robot arm. Our aim is to change the configuration of the blocks from the Initial State to the Goal State, both of which have been specified in the diagram below.



**Predicates**

1. ON(A,B) : Block A is on B
2. ONTABLE(A) : A is on table
3. CLEAR(A) : Nothing is on top of A
4. HOLDING(A) : Arm is holding A.
5. ARMEMPTY : Arm is holding nothing.
6. MOVE(A,B,C): move A which is on top of B to top of C.

**PROGRAM**

*import* os *import* utils *import* time *import* sys

*from* tree\_node *import* TreeNode

*def* search(*queue*, *method*, *initial*, *goal*):

"""Searches the tree for a solution based on the search algorithm

."""

*if* method == 'itdeep':

"""This is for iterative deepening"""

*for* upperlimit *in range*(0, 200):

root = *TreeNode*(initial, *None*, *None*, 0, 0, 0)

depth = 0

limit = upperlimit

queue.*put*(root)

visited\_set = *set*() # *Set of visited states.*

start = time.*time*()

*while* (not queue.*empty*()) and (time.*time*() - start <= 60)

:

# *While the queue is not empty and a minutes hasn't p*

*assed.*

*if* method == 'limited':

*if* depth <= limit:

*break*

current = queue.*get*()

*if* current.*is\_goal*(goal):

*return* current

depth += 1

# *print(str(current.state))*

*if str*(current.state) in visited\_set:

*it to the children*

# *If this state has been visited before don't add*

# *and continue with the next child. continue*

current.*find\_children*(method, goal) # *Mark the state as visited.* visited\_set.*add*(*str*(current.state))

# *Add every child in the search queue. for* child *in* current.children:

queue.*put*(child)

*return None else*:

"""This is for depth, breadth and depth limitied search""" root = *TreeNode*(initial, *None*, *None*, 0, 0, 0)

depth = 0

limit = 1

queue.*put*(root)

visited\_set = *set*() # *Set of visited states.*

start = time.*time*()

*while* (not queue.*empty*()) and (time.*time*() - start <= 60):

### # While the queue is not empty and a minutes hasn't passe

*d.*

*if* method == 'limited':

*if* depth <= limit:

*break*

current = queue.*get*()

*if* current.*is\_goal*(goal):

*return* current

depth += 1

### # print(str(current.state))

*if str*(current.state) in visited\_set:

### # If this state has been visited before don't add it

*to the children*

# *and continue with the next child. continue*

*visited.*

current.*find\_children*(method, goal) visited\_set.*add*(*str*(current.state)) # *Mark the state as*

# *Add every child in the search queue. for* child *in* current.children:

queue.*put*(child)

*return None*

*def* main():

start = time.*time*() # *Start time.*

os.*system*('cls' *if* os.name == 'nt' *else* 'clear') # *Clears the te rminal.*

# *Handles the arguments. if len*(sys.argv) == 3:

### # If the args are 3 no output file name wasn't specified.

method = sys.argv[1] input\_file = sys.argv[2]

*elif len*(sys.argv) == 4:

### # If the args are 4 the output file name was specified.

method = sys.argv[1] input\_file = sys.argv[2] output\_file = sys.argv[3]

### else:

*print*(

*f*'Usage: {sys.argv[0]} <search algorithm> <problem file n ame> <solution file name>')

*print*('- search algorithms: depth (Depth First), breadth (Bre adth First), best (Best First), astar (A\*)')

sys.*exit*()

### # Initializes the type of queue based on the search method.

search\_queue = utils.METHODS[method]

### # Parse the data and get the objects (blocks), initial state and the goal state.

data = utils.*load\_problem*(input\_file) objects = utils.*get\_objects\_from\_file*(data) initial\_state = utils.*get\_initial\_state*(data) goal\_state = utils.*get\_goal\_state*(data)

*print*('OBJECTS:', objects)

n')

*print*('\n#################### INITIAL STATE ####################\

*print*(initial\_state)

i\_blocks = utils.*initialize\_blocks*(objects, initial\_state)

*print*('\n#################### GOAL STATE ####################\n')

*print*(goal\_state)

g\_blocks = utils.*initialize\_blocks*(objects, goal\_state)

solution\_node = *search*(search\_queue, method, i\_blocks, g\_blocks)

*if* solution\_node != *None*:

### # If a solution is found.

*print*('\n#################### SOLUTION ####################\n

')

solution\_node.*print\_state*()

*print*(*f*'Number of moves: {solution\_node.g}')

# *Calculates the time it took to find the solution. print*('Took: ', time.*time*() - start)

solution\_path = solution\_node.*get\_moves\_to\_solution*() solution\_path.*reverse*()

*print*(solution\_path)

### slashes.

*if len*(sys.argv) == 3:

### # If the output file name was not specified. try:

# *Handling the paths with forward-slashes and back-*

file\_name = input\_file.*split*('\\')[-1] output\_file = './solutions/' + method + '-

' + file\_name

utils.*write\_solution*(output\_file, solution\_path) sys.*exit*()

*except* FileNotFoundError:

file\_name = input\_file.*split*('/')[-1] output\_file = './solutions/' + method + '-

' + file\_name

### else:

utils.*write\_solution*(output\_file, solution\_path) sys.*exit*()

# *If the output file name is specified.* utils.*write\_solution*(output\_file, solution\_path) sys.*exit*()

*else*:

*print*('Took: ', time.*time*() - start)

*print*('############ ONE MINUTE PASSED AND NO SOLUTION WAS FOU ND ############')

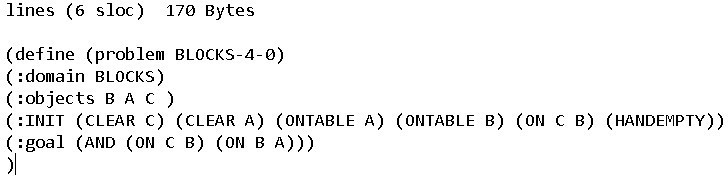
sys.*exit*()

*if* name == ' main ':

*main*()

# INPUT

## Question/Input:





C

|  |
| --- |
| C |
| B |
|  |

A



B

A

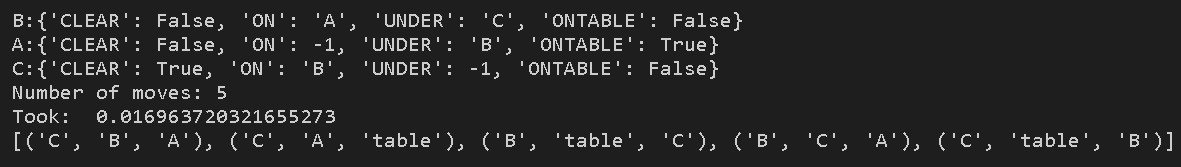
Initial State final state

**OUTPUT**

1. move ('C', 'B', 'A')

1. move ('C', 'A', 'table')
2. move ('B', 'table', 'C') 4. move ('B', 'C', 'A')

5. move ('C', 'table', 'B')



**RESULT:**

Thus, Tic Tac Toe was successfully implemented and executed in Python.

**NAME :- HARIT KUMAR KHARB**

**Reg No :- RA1811030010037**

**Sec :- L1**