

report

這次作業的要求是用五種演算法解rush hour的問題，以下是我針對五種演算法，分別比較其所需移動的步數(深度)及所經過之節點數(空間)。

L_stage	BFS	moves	nodes	time	DFS	moves	nodes	time	IDS	moves	nodes	time	A*	moves	nodes	time	IDA*	moves	nodes	time
1		8	1058	00:04.5		205	219	00:04.7		8	1058	00:05.6		8	506	00:03.5		8	1054	00:03.9
2		8	2526	00:12.0		820	857	01:11.7		8	2526	00:18.4		14	1029	00:09.9		8	2240	00:10.2
3		14	775	00:04.2		161	336	00:06.4		14	775	00:04.2		18	548	00:02.8		14	783	00:02.9
4		9	341	00:01.5		92	188	00:02.0		9	341	00:01.9		11	308	00:01.7		9	337	00:01.5
10		17	1978	00:11.6		218	245	00:05.9		17	1978	00:14.1		17	1518	00:12.1		17	1870	00:11.4
11		25	830	00:04.6		157	204	00:03.5		25	830	00:04.9		33	795	00:05.7		25	812	00:04.8
20		10	1558	00:07.8		195	2894	03:28.2		10	1558	00:09.7		17	2949	00:27.9		10	1585	00:07.4
21		21	258	00:00.7		72	125	00:01.0		21	258	00:01.0		23	255	00:01.3		21	258	00:00.9
22		26	3460	00:28.6		832	914	01:32.2		26	3460	00:34.6		29	2702	00:28.0		26	3443	00:30.1
23		29	2380	00:16.8		550	819	00:56.7		29	2380	00:22.3		38	1695	00:16.2		29	2185	00:18.6
24		25	4342	00:26.9		827	888	01:46.8		25	4342	00:37.2		30	4202	00:34.2		25	4342	00:30.4
25		27	8475	01:05.5		1653	1750	05:13.2		27	8475	01:34.4		46	6827	01:21.0		27	8341	01:09.9
26		28	4700	00:32.4		808	906	01:15.2		28	4700	00:39.8		33	2595	00:25.0		28	4690	00:28.1
27		28	2661	00:15.7		487	655	00:31.4		28	2661	00:20.1		42	1995	00:20.8		28	2569	00:14.0
28		30	1924	00:14.0		494	687	00:34.1		30	1924	00:18.8		38	1307	00:13.8		30	1888	00:15.2
29		31	4328	00:26.2		724	909	01:04.1		31	4328	00:39.3		50	3185	00:27.4		31	4325	00:31.1
30		32	1164	00:07.4		231	311	00:07.6		32	1164	00:09.1		34	978	00:08.0		32	1158	00:07.6
31		37	3976	00:27.6		717	2047	02:53.4		37	3976	00:38.1		43	3639	00:33.8		37	3947	00:31.9
40		51	3025	00:31.4		473	628	00:27.9		51	3025	00:37.3		54	2707	00:28.2		51	3045	00:33.7

左邊第一行是關卡(題目)編號Lxx.txt，然後後方是各演算法步數、節點數、時間。

根據觀察，DFS雖然所需之步數會較BFS多，但節點數反而會較少，時間方面，DFS 若在錯誤的路徑下不斷搜索，也可能導致浪費較多時間。

而第三種方法，IDS在此種情況下，表現就和BFS一樣，可能是因為其演算法是固定層數的DFS，最後導致和BFS一樣的行為。

第四種A*演算法，由於是採用blocking heuristic的方法，搜索時會以blocking value值較小的情況優先搜索，因此在快接近答案時(blocking value很小)，能較快找到答案，所以所需步數類似於BFS，但經過之節點數明顯優於BFS。

最後一種IDA*的方法，感覺起來表現和BFS差不多，推測是因為每層的深度都固定，所以無法在接近答案時，繼續向下搜索，因此表現會類似於IDS和BFS。

在實作的過程時發現，DFS的答案很容易有步數和節點數很接近之情形，可能是因為在某種情形下，不斷加深的深度都沒有遇到死路，所以一直做多餘的步數，導致深度不斷加深，最後才找到答案。

A*演算法還滿讓我意外的，本來以為要做很久，沒想到加了一些東西就能有不錯的結果了，也因為每次的下一步都是從可能的步數裡挑blocking value最小的，因而

跳過很多會增加blocking value的動作(雖然也可能會跳過答案)，不過這樣的演算法確實有發揮效果。

一開始要做作業時，本來覺得很麻煩，有五種演算法，加上之前學過的BFS、DFS和資料結構都還得差不多了，不過實際做了之後發現好像沒有那麼複雜，也學到了一些python的資料結構用法。另外我有把題目和解法轉換成圖形的形式，以第一題的答案為例，如下所示。

(一開始的puzzle)

```
1 1 _ _ 7
4 _ 6 _ 7
4 0 0 6 _ 7
4 _ 6 _ _
5 _ _ 2 2
5 _ 3 3 3 _
```

$\langle 2, 4, 4 \rangle \implies \langle 2, 4, 1 \rangle$ (代表該編號的車2，(超過10就用16進位ABC)從左邊的位置移到右邊(4,4 \rightarrow 4,1))

然後印出下一個puzzle，之後以此類推

```
1 1 _ _ 7
4 _ 6 _ 7
4 0 0 6 _ 7
4 _ 6 _ _
5 2 2 _ _
5 _ 3 3 3 _
```

$\langle 7, 0, 5 \rangle \implies \langle 7, 3, 5 \rangle$

```
1 1 _ _ _
4 _ 6 _ _
4 0 0 6 _ _
4 _ 6 _ 7
5 2 2 _ 7
5 _ 3 3 3 7
```

$\langle 1, 0, 0 \rangle \implies \langle 1, 0, 4 \rangle$

```
_ _ _ 1 1
4 _ 6 _ _
4 0 0 6 _ _
4 _ 6 _ 7
5 2 2 _ 7
5 _ 3 3 3 7
```

$\langle 4, 1, 0 \rangle \implies \langle 4, 0, 0 \rangle$

```
4 _ _ 1 1
4 _ 6 _ _
4 0 0 6 _ _
_ _ 6 _ 7
5 2 2 _ 7
5 _ 3 3 3 7
```

$\langle 5, 4, 0 \rangle \implies \langle 5, 3, 0 \rangle$

```

4 _ _ _ 1 1
4 _ _ 6 _ _
4 0 0 6 _ _
5 _ _ 6 _ 7
5 2 2 _ _ 7
_ _ 3 3 3 7

```

< 3, 5, 2 > ==> < 3, 5, 0 >

```

4 _ _ _ 1 1
4 _ _ 6 _ _
4 0 0 6 _ _
5 _ _ 6 _ 7
5 2 2 _ _ 7
3 3 3 _ _ 7

```

< 6, 1, 3 > ==> < 6, 3, 3 >

```

4 _ _ _ 1 1
4 _ _ _ _ _
4 0 0 _ _ _
5 _ _ 6 _ 7
5 2 2 6 _ 7
3 3 3 6 _ 7

```

< 0, 2, 1 > ==> < 0, 2, 4 >

```

4 _ _ _ 1 1
4 _ _ _ _ _
4 _ _ 0 0
5 _ _ 6 _ 7
5 2 2 6 _ 7
3 3 3 6 _ 7

```

Algorithm: A*
Puzzle completed in 8 moves.
Number of nodes visited in search: 506
time: 0:00:03.476703

雖然說把 puzzle 的圖形印出來在驗證答案時比較清楚也比較能接受，但也花了不少時間，但總體來說還是滿值得的，畢竟感覺踏實了不少。

說到 remaining questions，應該是 heuristic 的方法應該不止這一種吧，還有老師提到的自己生產題目，目前我只有想法，就是先隨便產生一個小紅在(2, 4)上的 puzzle，把這個狀態當成是最後的答案，然後用上面那五種方法，看看能不能讓小紅走回(2,0)或(2,1)，如果可以，那可能就是一個好題目，不行就再試一組，簡而言之，就是先產生答案，再倒回去變成題目。

連假期間我突然想到，關於 blocking 的計算，應該不止於 blocking 住紅車的車需

要計算，可以把 blocking 住紅車的其他車也分別計算他們各自的 blocking value，還有 blocking 住 blocking 住紅車的其他車(如下圖，6 號車 block 住 0 號，3 號又 block 住 5 號，最多計算四層，避免卡死)，把這些值都加到初始狀態的 blocking value 裡，然後之後從 value 最小的開始搜尋，搜尋的效果對於空間和空間複雜度有進步不少，雖然深度有稍微多一點，但整體效率感覺比原本的 A*還要好。

```

1 1 _ _ 7
4 _ _ 6 _ 7
4 0 0 6 _ 7
4 _ _ 6 _ _
5 _ _ _ 2 2
5 _ 3 3 3 _

```

下圖是比較結果，最右邊是新的 heuristic 方法，可以明顯看到步數多了一點(比 DFS 少，但 nodes 和時間減少很多)

stage	BFS	moves	nodes	time	DFS	moves	nodes	time	IDS	moves	nodes	time	A*	moves	nodes	time	IDA*	moves	nodes	time	new_heuri	moves	nodes	time
1		8	1058	00:04.5		205	219	00:04.7		8	1058	00:05.6		8	506	00:03.5		8	1054	00:03.9		10	357	00:01.8
2		8	2526	00:12.0		820	857	01:11.7		8	2526	00:18.4		14	1029	00:09.9		8	2240	00:10.2		13	133	00:00.7
3		14	775	00:04.2		161	336	00:06.4		14	775	00:04.2		18	548	00:02.8		14	783	00:02.9		15	583	00:02.6
4		9	341	00:01.5		92	188	00:02.0		9	341	00:01.9		11	308	00:01.7		9	337	00:01.5		11	76	00:00.3
10		17	1978	00:11.6		218	245	00:05.9		17	1978	00:14.1		17	1518	00:12.1		17	1870	00:11.4		21	745	00:04.6
11		25	830	00:04.6		157	204	00:03.5		25	830	00:04.9		33	795	00:05.7		25	812	00:04.8		43	615	00:03.9
20		10	1558	00:07.8		195	2894	03:28.2		10	1558	00:09.7		17	2949	00:27.9		10	1585	00:07.4		28	2609	00:26.2
21		21	258	00:00.7		72	125	00:01.0		21	258	00:01.0		23	255	00:01.3		21	258	00:00.9		21	211	00:00.9
22		26	3460	00:28.6		832	914	01:32.2		26	3460	00:34.6		29	2702	00:28.0		26	3443	00:30.1		53	859	00:08.1
23		29	2380	00:16.8		550	819	00:56.7		29	2380	00:22.3		38	1695	00:16.2		29	2185	00:18.6		39	738	00:07.6
24		25	4342	00:26.9		827	888	01:46.8		25	4342	00:37.2		30	4202	00:34.2		25	4342	00:30.4		39	3918	00:30.6
25		27	8475	01:05.5		1653	1750	05:13.2		27	8475	01:34.4		46	6827	01:21.0		27	8341	01:09.9		74	4839	01:11.1
26		28	4700	00:32.4		808	906	01:15.2		28	4700	00:39.8		33	2595	00:25.0		28	4690	00:28.1		49	2141	00:23.1
27		28	2661	00:15.7		487	655	00:31.4		28	2661	00:20.1		42	1995	00:20.8		28	2569	00:14.0		53	1852	00:19.5
28		30	1924	00:14.0		494	687	00:34.1		30	1924	00:18.8		38	1307	00:13.8		30	1888	00:15.2		43	463	00:03.7
29		31	4328	00:26.2		724	909	01:04.1		31	4328	00:39.3		50	3185	00:27.4		31	4325	00:31.1		63	1988	00:17.7
30		32	1164	00:07.4		231	311	00:07.6		32	1164	00:09.1		34	978	00:08.0		32	1158	00:07.6		48	872	00:08.2
31		37	3976	00:27.6		717	2047	02:53.4		37	3976	00:38.1		43	3639	00:33.8		37	3947	00:31.9		84	2984	00:39.1
40		51	3025	00:31.4		473	628	00:27.9		51	3025	00:37.3		54	2707	00:28.2		51	3045	00:33.7		80	1501	00:16.6

附錄

Code

compile 方式：python test.py 1 prog1_puzzle/L01.txt

(1-6 代表六種演算法)

也可用./test 之後選擇演算法

程式有放在 github

<https://github.com/tim310579/Artificial-Intelligence/tree/main/HW1>

test.py(主程式)

```
from Puzzles import *
from Algos import *
from sys import argv
import datetime
import psutil
import os
```

```
begin = datetime.datetime.now()
```

```
useless, algo, filename = argv
#print(algo, filename)
f = open(filename, 'r')
k = f.readlines()
#for lines in k:
    #print(lines)
```

```
tmp = []
for lines in k:
    words = lines.split(' ')
    #print(words[3])
    ori = Orientations.vertical
    length = VehicleTypes.car
    words[4] = words[4][0]
    if words[4] == '1': ori = Orientations.horizontal
    if words[3] == '3': length = VehicleTypes.truck
    tmp.append(Vehicle((int(words[2]), int(words[1])), ori, length, words[0]))
    #print(words[2], words[1], words[4], words[3])
```

```
trafficJamtmp = Puzzles(6, 6, 2, tmp)
f.close()
```

```
#print(trafficJamtmp)
```

```

def printSolution(puzzle, solution):
    for m in solution:
        print(puzzle)
        print(m)
        puzzle.move(m.pos, m.moves)
    print(puzzle)

```

```

# Create AI agent and run on specified puzzles
agent = Algos()

```

```

solution = ""
if algo == '1':
    solution = agent.bfs(trafficJamtmp)
    #print('BFS')
elif algo == '2':
    solution = agent.dfs(trafficJamtmp)
    #print('DFS')
elif algo == '3':
    solution = agent.ids(trafficJamtmp)
    #print('IDS')
elif algo == '4':
    solution = agent.a_star(trafficJamtmp)
    #print('A*')
elif algo == '5':
    solution = agent.ida_star(trafficJamtmp)
elif algo == '6':
    solution = agent.another_h(trafficJamtmp)
    #print('IDA*')
#solution = agent2.dfs(trafficJamtmp)
printSolution(trafficJamtmp, solution)
print('Algorithm: ', end='')
if algo == '1': print('BFS')
elif algo == '2': print('DFS')
elif algo == '3': print('IDS')
elif algo == '4': print('A*')
elif algo == '5': print('IDA*')
elif algo == '6': print('Another heuristic')

print("Puzzle completed in " + str(len(solution)) + " moves.")
print("Number of nodes visited in search:  " + str(agent.nodesVisited))
#print("Space: " + str(agent.space))

```

```

end = datetime.datetime.now()
print('Time: ', end-begin)

#info = psutil.virtual_memory()

f = open('result/statistic.txt', 'a')
f.write(str(len(solution)) + ' ' + str(agent.nodesVisited) + ' ' + str(end-begin) + '\n')
f.close()

```

Algos.py(演算法部分)

```

from Puzzles import *
from collections import deque
import copy
from queue import PriorityQueue
class Algos:

    def __init__(self):
        self.nodesVisited = 0
        self.space = 0

    def bfs(self, puzzle):

        bfsQueue = deque([])
        self.nodesVisited = 0
        self.space = 0
        # The current node/state
        current = BfsNode(puzzle, [])

        seenPuzzleStates = {}
        seenPuzzleStates[str(current.puzzle.getGrid())] = True;

        while not current.puzzle.won():
            self.nodesVisited += 1

            for m in current.getPossibleMoves():

                # Duplicate puzzle state and perform a move
                newState = copy.deepcopy(current)
                newState.puzzle.move(m.pos, m.moves)
                self.space += 1
                # If new state is unseen, add to queue and seen states list

```

```

        if ((not str(newState.puzzle) in seenPuzzleStates) or
seenPuzzleStates[str(newState.puzzle)] > len(newState.movesSoFar)):
            bfsQueue.append(BfsNode(newState.puzzle,
current.movesSoFar + [m]))
            seenPuzzleStates[str(newState.puzzle)] = True;
            current = bfsQueue.popleft()

```

```

    return current.movesSoFar
def dfs(self, puzzle):

```

```

    # Queue to hold untraversed nodes
    dfsQueue = deque([])
    self.nodesVisited = 0

```

```

    # The current node/state
    current = DfsNode(puzzle, [])

```

```

    seenPuzzleStates = {}
    seenPuzzleStates[str(current.puzzle.getGrid())] = True;

```

```

    while not current.puzzle.won():
        self.nodesVisited += 1

```

```

        for m in current.getPossibleMoves():

```

```

            # Duplicate puzzle state and perform a move
            newState = copy.deepcopy(current)
            newState.puzzle.move(m.pos, m.moves)

```

```

            # If new state is unseen, add to queue and seen states list
            if ((not str(newState.puzzle) in seenPuzzleStates) or
seenPuzzleStates[str(newState.puzzle)] > len(newState.movesSoFar)):
                dfsQueue.append(DfsNode(newState.puzzle,
current.movesSoFar + [m]))
                seenPuzzleStates[str(newState.puzzle)] = True;
                current = dfsQueue.pop()

```

```

    return current.movesSoFar
def ids(self, puzzle):

```

```

    cnt = 0
    idsQueue = PriorityQueue()
    self.nodesVisited = 0
    self.space = 0

```



```

# The current node/state
current = IdsNode(puzzle, [], 0)
seenPuzzleStates = {}
seenPuzzleStates[str(current.puzzle.getGrid())] = True
#print(type(current))
while not (current.puzzle.won()):
    if 1 == 1:
        self.nodesVisited += 1

        for m in current.getPossibleMoves():
            cnt += 1
            #print('iiii')
            # Duplicate puzzle state and perform a
move
            newState = copy.deepcopy(current)
            newState.puzzle.move(m.pos,
m.moves)

            self.space += 1
            # If new state is unseen, add to queue
            and seen states list
            if ((not str(newState.puzzle) in
seenPuzzleStates) or seenPuzzleStates[str(newState.puzzle)] >
len(newState.movesSoFar)):

#idsQueue.append(IdsNode(current.thedeep+1, newState.puzzle, current.movesSoFar +
[m]))

idsQueue.put((current.thedeep+1, cnt, IdsNode(newState.puzzle, current.movesSoFar +
[m], current.thedeep+1)))

seenPuzzleStates[str(newState.puzzle)] = True;

#deep += 1
#current = idsQueue.popleft()
tmp = idsQueue.get()
current = tmp[2]
#print(tmp[0])

return current.movesSoFar

def a_star(self, puzzle):

# Queue to hold untraversed nodes
a_starQueue = PriorityQueue()
self.nodesVisited = 0
self.space = 0

```

```

# The current node/state
current = A_starNode(puzzle, [], 0)

seenPuzzleStates = {}
seenPuzzleStates[str(current.puzzle.getGrid())] = True;
cnt = 0
while not(current.puzzle.won()):
    #print(current.blocking)
    self.nodesVisited += 1

    for m in current.getPossibleMoves():
        cnt += 1
        # Duplicate puzzle state and perform a move
        newState = copy.deepcopy(current)
        newState.puzzle.move(m.pos, m.moves)
        self.space += 1
        # If new state is unseen, add to queue and seen states list
        if ((not str(newState.puzzle) in seenPuzzleStates) or
seenPuzzleStates[str(newState.puzzle)] > len(newState.movesSoFar)):
            #a_starQueue.append(A_starNode(newState.puzzle,
current.movesSoFar + [m], 0))
            a_starQueue.put((current.blocking, cnt,
A_starNode(newState.puzzle, current.movesSoFar + [m], current.getblocking()))
seenPuzzleStates[str(newState.puzzle)] = True;
            #current = a_starQueue.popleft()
            tmp = a_starQueue.get()
            current = tmp[2]
    return current.movesSoFar
def ida_star(self, puzzle):

    ida_starQueue = PriorityQueue()
    self.nodesVisited = 0
    self.space = 0
    # The current node/state
    current = IDA_starNode(puzzle, [], 0, 0)

    seenPuzzleStates = {}
    seenPuzzleStates[str(current.puzzle.getGrid())] = True;
    cnt = 0
    while not(current.puzzle.won()):
        #print(current.blocking)
        self.nodesVisited += 1

        for m in current.getPossibleMoves():

```

```

        cnt += 1
        # Duplicate puzzle state and perform a move
        newState = copy.deepcopy(current)
        newState.puzzle.move(m.pos, m.moves)
        self.space += 1
        # If new state is unseen, add to queue and seen states list
        if ((not str(newState.puzzle) in seenPuzzleStates) or
seenPuzzleStates[str(newState.puzzle)] > len(newState.movesSoFar)):
            #a_starQueue.append(A_starNode(newState.puzzle,
current.movesSoFar + [m], 0))
            ida_starQueue.put((current.thedeep, current.blocking, cnt,
IDA_starNode(newState.puzzle, current.movesSoFar + [m], current.getblocking(),
current.thedeep+1)))
            seenPuzzleStates[str(newState.puzzle)] = True;
        #current = a_starQueue.popleft()
        tmp = ida_starQueue.get()
        current = tmp[3]
    return current.movesSoFar

```

```

def another_h(self, puzzle):

```

```

    # Queue to hold untraversed nodes
    another_hQueue = PriorityQueue()
    self.nodesVisited = 0
    self.space = 0
    # The current node/state
    current = another_hNode(puzzle, [], 0)

```

```

    seenPuzzleStates = {}
    seenPuzzleStates[str(current.puzzle.getGrid())] = True;
    cnt = 0
    while not(current.puzzle.won()):
        #print(current.blocking)
        self.nodesVisited += 1

```

```

        for m in current.getPossibleMoves():
            cnt += 1
            # Duplicate puzzle state and perform a move
            newState = copy.deepcopy(current)
            newState.puzzle.move(m.pos, m.moves)
            self.space += 1
            # If new state is unseen, add to queue and seen states list
            if ((not str(newState.puzzle) in seenPuzzleStates) or
seenPuzzleStates[str(newState.puzzle)] > len(newState.movesSoFar)):
                #a_starQueue.append(A_starNode(newState.puzzle,

```

```

current.movesSoFar + [m], 0))
        another_hQueue.put((current.blocking, cnt,
another_hNode(newState.puzzle, current.movesSoFar + [m], current.getblocking()))
        seenPuzzleStates[str(newState.puzzle)] = True;
        #current = a_starQueue.popleft()
        tmp = another_hQueue.get()
        current = tmp[2]
    return current.movesSoFar

```

```

class BfsNode:

```

```

    def __init__(self, puzzle, movesSoFar):
        self.puzzle = puzzle
        self.movesSoFar = movesSoFar

    def getPossibleMoves(self):
        results = []
        current = self.puzzle
        for v in current.vehicles:
            for i in current.moveRange(v):
                #print('v:',v,'v')
                # Don't move if move length is 0
                if not i == 0:
                    results += [Move(v.pos, i, v.orientation, v.number)]
        #print(results)
        return results

```

```

class DfsNode:

```

```

    def __init__(self, puzzle, movesSoFar):
        self.puzzle = puzzle
        self.movesSoFar = movesSoFar

    def getPossibleMoves(self):
        results = []
        current = self.puzzle
        for v in current.vehicles:
            for i in current.moveRange(v):
                #print('v:',v,'v')
                # Don't move if move length is 0
                if not i == 0:
                    results += [Move(v.pos, i, v.orientation, v.number)]
        #print(results)
        return results

```

```

class IdsNode:

```

```

def __init__(self, puzzle, movesSoFar, thedeep):
    self.puzzle = puzzle
    self.movesSoFar = movesSoFar
    self.thedeep = thedeep

def getPossibleMoves(self):
    results = []
    current = self.puzzle
    for v in current.vehicles:
        for i in current.moveRange(v):
            #print('v:',v,'v')
            # Don't move if move length is 0
            if not i == 0:
                results += [Move(v.pos, i, v.orientation, v.number)]
    #print(results)
    return results

```

```

class A_starNode:
    def __init__(self, puzzle, movesSoFar, blocking):
        self.puzzle = puzzle
        self.movesSoFar = movesSoFar
        self.blocking = blocking

    def getPossibleMoves(self):
        results = []
        current = self.puzzle
        for v in current.vehicles:
            for i in current.moveRange(v):
                #print('v:',v,'v')
                # Don't move if move length is 0
                if not i == 0:
                    results += [Move(v.pos, i, v.orientation, v.number)]
        #print(results)
        return results

    def getblocking(self):
        current = self.puzzle
        blockingcars = 0
        red_car_pos = 0
        for v in current.vehicles:          #find red car
            if v.number == '0':
                if v.pos[0] == 4:
                    return 0
            else:
                red_car_pos = v.pos[0]

```

```

                                break
                blockingcars = 1
                for v in current.vehicles:
                    if v.orientation == Orientations.vertical and v.pos[0] >
red_car_pos:    #may block
                                if v.vType == VehicleTypes.car and (v.pos[1] == 1
or v.pos[1] == 2):    #is block
                                blockingcars +=1
                                elif v.vType == VehicleTypes.truck and (v.pos[1]
>= 0 and v.pos[1] <=2):    #is block
                                blockingcars +=1
                return blockingcars

```

```

class IDA_starNode:
    """Represents a single state of the BFS
    Attributes:
        puzzle (JamPuzzle):  the puzzle state this node represents
        movesSoFar (Move[]):  array of moves taken to get to the current
                               state.  Holds the solution at the end, since BFS itself
                               doesn't track moves so far for each state.
        getPossibleMoves(self): retrieves list of all valid moves from this
                               node's state
    """

    def __init__(self, puzzle, movesSoFar, blocking, thedeep):
        """Constructor takes a puzzle state and list of moves taken
        so far to get there.
        """
        self.puzzle = puzzle
        self.movesSoFar = movesSoFar
        self.blocking = blocking
        self.thedeep = thedeep

    def getPossibleMoves(self):
        """Find the moveRange() of each vehicle in puzzle state and
        adds every move (except 0 moves) in the range for each vehicle
        to a result list of Move objects
        Return:
            Move[]: the array of all valid moves for this node's state
        """
        results = []
        current = self.puzzle
        for v in current.vehicles:
            for i in current.moveRange(v):

```

```

        #print('v:',v,'v')
        # Don't move if move length is 0
        if not i == 0:
            results += [Move(v.pos, i, v.orientation, v.number)]

    #print(results)
    return results
def getblocking(self):
    current = self.puzzle
    blockingcars = 0
    red_car_pos = 0
    for v in current.vehicles:        #find red car
        if v.number == '0':
            if v.pos[0] == 4:
                return 0
            else:
                red_car_pos = v.pos[0]
        break
    blockingcars = 1
    for v in current.vehicles:
        if v.orientation == Orientations.vertical and v.pos[0] >
red_car_pos:    #may block
                if v.vType == VehicleTypes.car and (v.pos[1] == 1
or v.pos[1] == 2):    #is block
                    blockingcars +=1
                elif v.vType == VehicleTypes.truck and (v.pos[1]
>= 0 and v.pos[1] <=2):    #is block
                    blockingcars +=1
    return blockingcars

class another_hNode:
    def __init__(self, puzzle, movesSoFar, blocking):
        self.puzzle = puzzle
        self.movesSoFar = movesSoFar
        self.blocking = blocking

    def getPossibleMoves(self):
        results = []
        current = self.puzzle
        for v in current.vehicles:
            for i in current.moveRange(v):
                #print('v:',v,'v')
                # Don't move if move length is 0
                if not i == 0:
                    results += [Move(v.pos, i, v.orientation, v.number)]

```

```

        #print(results)
        return results
    def getblocking(self):
        current = self.puzzle
        blockingcars = 0
        red_car_pos = 0
        for v in current.vehicles:          #find red car
            if v.number == '0':
                if v.pos[0] == 4:
                    return 0
                else:
                    red_car_pos = v.pos[0]
            break
        blockingcars = 1
        for v in current.vehicles:
            if v.orientation == Orientations.vertical and v.pos[0] >
red_car_pos:    #may block
                if v.vType == VehicleTypes.car and (v.pos[1] == 1
or v.pos[1] == 2):    #is block
                    blockingcars +=
get_another_blocking(current, v, 0)
                elif v.vType == VehicleTypes.truck and (v.pos[1]
>= 0 and v.pos[1] <=2):    #is block
                    blockingcars +=
get_another_blocking(current, v, 0)
                #print(blockingcars)
                return blockingcars

    def get_another_blocking(self, goal_car, itr):
        itr = itr + 1
        if itr >= 3: return 1
        current = self
        blockingcars = 0
        if goal_car.orientation == Orientations.vertical:
            for v in current.vehicles:
                if v.orientation == Orientations.horizontal:    #horizontal
block vertical
                    if v.vType == VehicleTypes.car and (v.pos[0] ==
goal_car.pos[0]-1 or v.pos[0] == goal_car.pos[0]):    #is block
                        blockingcars +=
get_another_blocking(current, v, itr)
                    elif v.vType == VehicleTypes.truck and (v.pos[0]
>= goal_car.pos[0]-2 and v.pos[0] <= goal_car.pos[0]):    #is block
                        blockingcars +=

```



```

get_another_blocking(current, v, itr)
    elif goal_car.orientation == Orientations.horizontal:
        for v in current.vehicles:
            if v.orientation == Orientations.vertical:    #vertical block
hor
                if v.vType == VehicleTypes.car and (v.pos[1] ==
goal_car.pos[1]-1 or v.pos[1] == goal_car.pos[1]):    #is block
                    blockingcars +=
get_another_blocking(current, v, itr)
                elif v.vType == VehicleTypes.truck and (v.pos[1]
>= goal_car.pos[1]-2 and v.pos[1] <= goal_car.pos[1]):    #is block
                    blockingcars +=
get_another_blocking(current, v, itr)
    return blockingcars
class Move:
    def __init__(self, pos, moves, orientation, number):
        self.pos = pos;
        self.moves = moves;
        self.orientation = orientation;
        self.number = number;

    def __str__(self):
        #print(self.orientation, self.number)
        #return "Move car at (" +str(self.pos[1])+' '+str(self.pos[0])+" ) by
"+str(self.moves)+" to (" +str(self.pos[1])+' '+str(self.pos[0])+" )"
        action = "< " + self.number + ", " + str(self.pos[1]) + ", " +
str(self.pos[0]) + " >"
        if self.orientation == Orientations.horizontal:
            return action + " ==> < " + self.number + ", " +
str(self.pos[1]) + ", " + str(self.pos[0]) + self.moves + " >"
        else:
            return action + " ==> < " + self.number + ", " +
str(self.pos[1]) + self.moves + ", " + str(self.pos[0]) + " >"

```

Puzzles.py(圖形部分)

```
from enum import Enum, IntEnum
```

```

class VehicleTypes(IntEnum):
    car = 2
    truck = 3
class Orientations(IntEnum):

```

```

horizontal = 0
vertical = 1
#class Number(IntEnum):

```

```

class Puzzles:

```

```

    def __init__(self, gridSizeX, gridSizeY, doorPos, vehicles):
        self.gridSizeY = gridSizeY
        self.gridSizeX = gridSizeX
        self.doorPos = doorPos
        self.vehicles = vehicles

```

```

    def getSizeTuple(self):
        """Returns grid sizes as an (x, y) tuple
        Return:
            (int, int): tuple representing (width, height) of puzzle grid
        """
        return (self.gridSizeX, self.gridSizeY)

```

```

    def getGrid(self):

```

```

        #symbol = ord('A')
        symbol = ord('1')
        grid = [["_"] for y in range(self.gridSizeY)] for x in range(self.gridSizeX)]
        for v in self.vehicles:
            # iterate through each vehicle, assigning it a symbol and replacing its
            # covered locations with that symbol in the grid
            tempSymbol = chr(symbol)

            if v.pos[1] == self.doorPos and v.orientation == Orientations.horizontal:
                #print(v.pos[1])
                tempSymbol = '0'
            else:
                symbol += 1
                if symbol == 58: symbol += 7
            locs = v.coveredUnits()
            #print(locs)
            for l in locs:
                #print(l,"|||")
                grid[l[0]][l[1]] = tempSymbol
        return grid

```

```

    def move(self, pos, moves):

```

```

"""Wrapper for moveVehicle()
Args:
    pos ((int, int)): position of vehicle to move (x, y)
    moves (int): number of moves to move vehicle
"""
v = self.getVehicleAt(pos)
if v == None:
    raise Exception("Can't move vehicle; not found", pos)
self.moveVehicle(v, moves)

```

```

def moveVehicle(self, veh, moves):
    orient = veh.orientation
    newPosList = list(veh.pos)
    newPosList[orient] += moves
    veh.pos = tuple(newPosList)

```

```

def moveRange(self, veh):
    minMove = 0
    # Iterate over spaces behind to check
    for i in range(-1, -veh.pos[veh.orientation]-1, -1):

        # Only way to change a value in a tuple by index :/
        newPosList = list(veh.pos)
        newPosList[veh.orientation] += i
        newPosTuple = tuple(newPosList)

```

```

        blocked = False
        for v in self.vehicles:
            if newPosTuple in v.coveredUnits():
                blocked = True
                break
        if blocked:
            break
        else:
            minMove = i

```

```

        maxMove = 0
        # iterate over spaces ahead to check, not pos of vehicle. Accounts for
length of vehicle
        for j in range(veh.vType,
self.getSizeTuple()[veh.orientation]-veh.pos[veh.orientation]):
            # j is # of spaces ahead of vehicle position to check!

```

```

        # not position to check, or # of moves
        newPosList = list(veh.pos)
        newPosList[veh.orientation]+=j
        newPosTuple = tuple(newPosList)

        blocked = False
        for v in self.vehicles:
            if newPosTuple in v.coveredUnits():
                blocked = True
                break
        if blocked:
            break
        else:
            maxMove = j - veh.vType + 1

    return range(minMove, maxMove+1)

def getVehicleAt(self, pos):
    for v in self.vehicles:
        if v.pos == pos:
            return v
    return None

def won(self):
    v = self.getVehicleAt((4, self.doorPos))
    #print('haha:', v,'over')
    if v != None and v.orientation == Orientations.horizontal:
        return True
    return False

def __str__(self):
    result = "  " * self.doorPos + " " + "  " * (self.gridSizeX - self.doorPos - 1) +
    "\n"
    grid = self.getGrid()
    result += "\n".join([" ".join([grid[x][y] for x in range(self.gridSizeX)]) for y in
    range(self.gridSizeY)]) + "\n"
    return result

def __eq__(self, b):
    return self.getGrid() == b.getGrid()

```

```

class Vehicle:

```

```

def __init__(self, pos, orientation, vType, number):
    self.pos = pos
    self.orientation = orientation
    self.vType = vType
    self.number = number

def coveredUnits(self):

    if self.orientation == Orientations.vertical:
        result = [(self.pos[0], self.pos[1] + i) for i in range(int(self.vType))]
    if self.orientation == Orientations.horizontal:
        result = [(self.pos[0] + i, self.pos[1]) for i in range(int(self.vType))]
    return result

def __str__(self):
    orientTxt = "Horizontal" if self.orientation == Orientations.horizontal else
"Vertical"
    vehTxt = "Car" if self.vType == VehicleTypes.car else "Truck"
    positions = str(self.coveredUnits())
    return orientTxt + " " + vehTxt + " at (" + str(self.pos[0]) + ", " + str(self.pos[1]) +
") covering " + positions

```