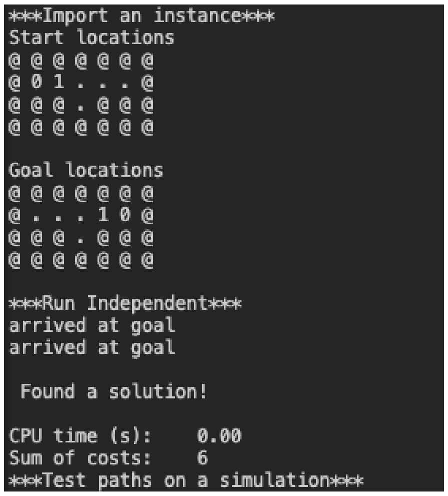
Leon Lee 301188000

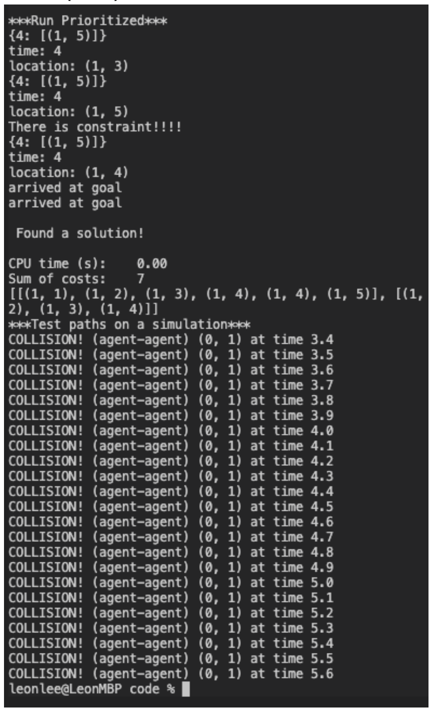
1.1



```
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ 0 1 . . . @
@ @ @ . @ @ @
@ @ @ @ @ @ @

Goal locations
@ @ @ @ @ @ @
@ . . . 1 0 @
@ @ @ . @ @ @
@ @ @ @ @ @ @

***Run Independent***
arrived at goal
arrived at goal

 Found a solution!

CPU time (s):    0.00
Sum of costs:    6
***Test paths on a simulation***
```

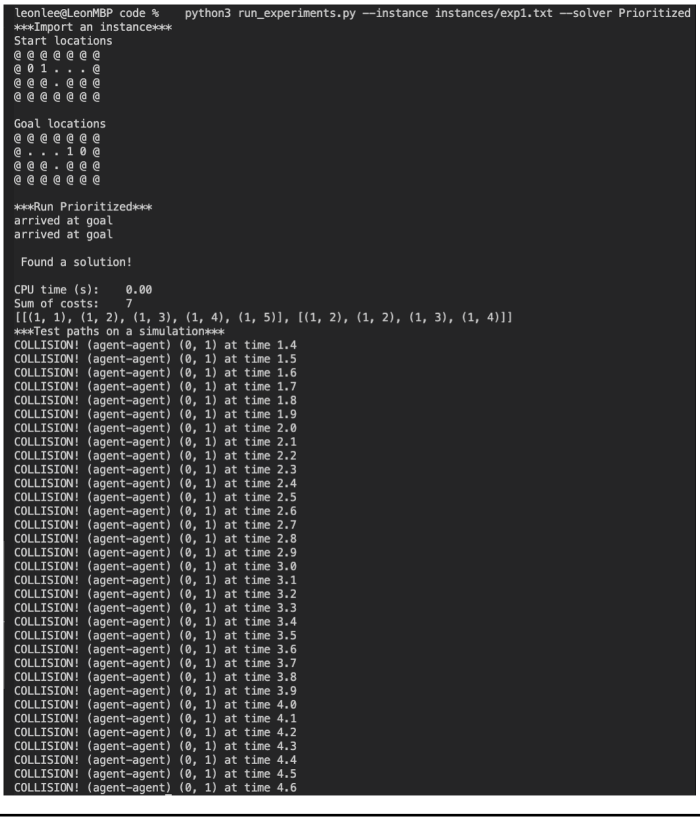
1.2

The constraint `agent 0 from being at its goal cell (1, 5) at time step 4  ` makes agent 0 stop at (1,5) at timestep 4, which we can see by the extended amount of collision with agent 1, since agent 1's goal is at (1,5)



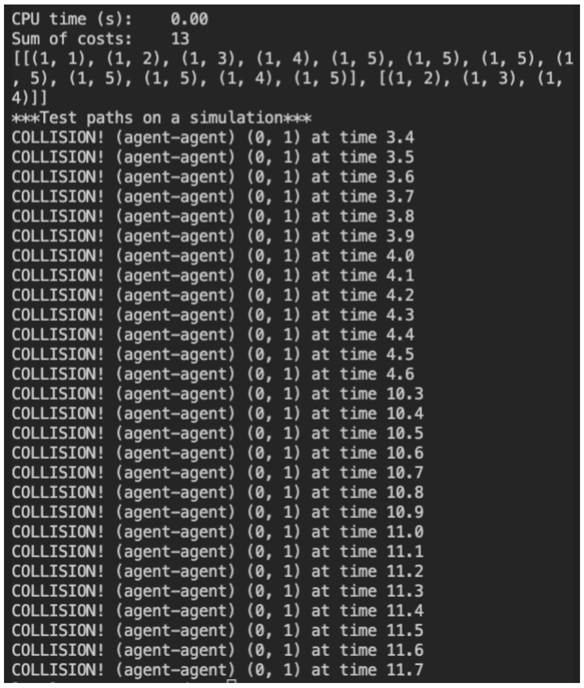
```
***Run Prioritized***
{4: [(1, 5)]}
time: 4
location: (1, 3)
{4: [(1, 5)]}
time: 4
location: (1, 5)
There is constraint!!!!
{4: [(1, 5)]}
time: 4
location: (1, 4)
arrived at goal
arrived at goal

 Found a solution!

CPU time (s):    0.00
Sum of costs:    7
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 4), (1, 5)], [(1,
2), (1, 3), (1, 4)]]
***Test paths on a simulation***
COLLISION! (agent-agent) (0, 1) at time 3.4
COLLISION! (agent-agent) (0, 1) at time 3.5
COLLISION! (agent-agent) (0, 1) at time 3.6
COLLISION! (agent-agent) (0, 1) at time 3.7
COLLISION! (agent-agent) (0, 1) at time 3.8
COLLISION! (agent-agent) (0, 1) at time 3.9
COLLISION! (agent-agent) (0, 1) at time 4.0
COLLISION! (agent-agent) (0, 1) at time 4.1
COLLISION! (agent-agent) (0, 1) at time 4.2
COLLISION! (agent-agent) (0, 1) at time 4.3
COLLISION! (agent-agent) (0, 1) at time 4.4
COLLISION! (agent-agent) (0, 1) at time 4.5
COLLISION! (agent-agent) (0, 1) at time 4.6
COLLISION! (agent-agent) (0, 1) at time 4.7
COLLISION! (agent-agent) (0, 1) at time 4.8
COLLISION! (agent-agent) (0, 1) at time 4.9
COLLISION! (agent-agent) (0, 1) at time 5.0
COLLISION! (agent-agent) (0, 1) at time 5.1
COLLISION! (agent-agent) (0, 1) at time 5.2
COLLISION! (agent-agent) (0, 1) at time 5.3
COLLISION! (agent-agent) (0, 1) at time 5.4
COLLISION! (agent-agent) (0, 1) at time 5.5
COLLISION! (agent-agent) (0, 1) at time 5.6
leonlee@LeonMBP code %
```

1.3



```
leonlee@LeonMBP code %    python3 run_experiments.py --instance instances/exp1.txt --solver Prioritized
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ 0 1 . . . @
@ @ @ . @ @ @
@ @ @ @ @ @ @

Goal locations
@ @ @ @ @ @ @
@ . . . 1 0 @
@ @ @ . @ @ @
@ @ @ @ @ @ @

***Run Prioritized***
arrived at goal
arrived at goal

 Found a solution!

CPU time (s):    0.00
Sum of costs:    7
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 2), (1, 3), (1, 4)]]
***Test paths on a simulation***
COLLISION! (agent-agent) (0, 1) at time 1.4
COLLISION! (agent-agent) (0, 1) at time 1.5
COLLISION! (agent-agent) (0, 1) at time 1.6
COLLISION! (agent-agent) (0, 1) at time 1.7
COLLISION! (agent-agent) (0, 1) at time 1.8
COLLISION! (agent-agent) (0, 1) at time 1.9
COLLISION! (agent-agent) (0, 1) at time 2.0
COLLISION! (agent-agent) (0, 1) at time 2.1
COLLISION! (agent-agent) (0, 1) at time 2.2
COLLISION! (agent-agent) (0, 1) at time 2.3
COLLISION! (agent-agent) (0, 1) at time 2.4
COLLISION! (agent-agent) (0, 1) at time 2.5
COLLISION! (agent-agent) (0, 1) at time 2.6
COLLISION! (agent-agent) (0, 1) at time 2.7
COLLISION! (agent-agent) (0, 1) at time 2.8
COLLISION! (agent-agent) (0, 1) at time 2.9
COLLISION! (agent-agent) (0, 1) at time 3.0
COLLISION! (agent-agent) (0, 1) at time 3.1
COLLISION! (agent-agent) (0, 1) at time 3.2
COLLISION! (agent-agent) (0, 1) at time 3.3
COLLISION! (agent-agent) (0, 1) at time 3.4
COLLISION! (agent-agent) (0, 1) at time 3.5
COLLISION! (agent-agent) (0, 1) at time 3.6
COLLISION! (agent-agent) (0, 1) at time 3.7
COLLISION! (agent-agent) (0, 1) at time 3.8
COLLISION! (agent-agent) (0, 1) at time 3.9
COLLISION! (agent-agent) (0, 1) at time 4.0
COLLISION! (agent-agent) (0, 1) at time 4.1
COLLISION! (agent-agent) (0, 1) at time 4.2
COLLISION! (agent-agent) (0, 1) at time 4.3
COLLISION! (agent-agent) (0, 1) at time 4.4
COLLISION! (agent-agent) (0, 1) at time 4.5
COLLISION! (agent-agent) (0, 1) at time 4.6
```

1.4



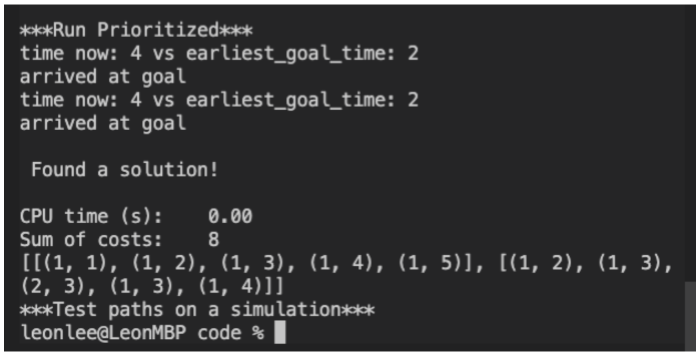
```
CPU time (s):    0.00
Sum of costs:    13
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 5), (1, 5), (1
, 5), (1, 5), (1, 5), (1, 4), (1, 5)], [(1, 2), (1, 3), (1,
4)]]
***Test paths on a simulation***
COLLISION! (agent-agent) (0, 1) at time 3.4
COLLISION! (agent-agent) (0, 1) at time 3.5
COLLISION! (agent-agent) (0, 1) at time 3.6
COLLISION! (agent-agent) (0, 1) at time 3.7
COLLISION! (agent-agent) (0, 1) at time 3.8
COLLISION! (agent-agent) (0, 1) at time 3.9
COLLISION! (agent-agent) (0, 1) at time 4.0
COLLISION! (agent-agent) (0, 1) at time 4.1
COLLISION! (agent-agent) (0, 1) at time 4.2
COLLISION! (agent-agent) (0, 1) at time 4.3
COLLISION! (agent-agent) (0, 1) at time 4.4
COLLISION! (agent-agent) (0, 1) at time 4.5
COLLISION! (agent-agent) (0, 1) at time 4.6
COLLISION! (agent-agent) (0, 1) at time 10.3
COLLISION! (agent-agent) (0, 1) at time 10.4
COLLISION! (agent-agent) (0, 1) at time 10.5
COLLISION! (agent-agent) (0, 1) at time 10.6
COLLISION! (agent-agent) (0, 1) at time 10.7
COLLISION! (agent-agent) (0, 1) at time 10.8
COLLISION! (agent-agent) (0, 1) at time 10.9
COLLISION! (agent-agent) (0, 1) at time 11.0
COLLISION! (agent-agent) (0, 1) at time 11.1
COLLISION! (agent-agent) (0, 1) at time 11.2
COLLISION! (agent-agent) (0, 1) at time 11.3
COLLISION! (agent-agent) (0, 1) at time 11.4
COLLISION! (agent-agent) (0, 1) at time 11.5
COLLISION! (agent-agent) (0, 1) at time 11.6
COLLISION! (agent-agent) (0, 1) at time 11.7
```

Assuming 'step 10' means timestep=10, agent 0 actually moves away from goal state just to adhere to the constraint, and then quickly moves back to goal.

As a result for the edge constraint, agent 1 doesn't start moving until agent 0 gets to (1,2), which is a coincidence, and they start moving towards goal, which creates even more collision.
Since "from timestep 4 to 5" is 'timestep': 5,
"from timestep 0 to 1" is just 'timestep': 1

For goal constraint, we make the biggest timestep value to be earliest_goal_timestep. Then when we check for whether an agent has arrived at goal node yet, we add in one more criteria: agent must arrive at goal no earlier than earliest_goal_time.

1.5

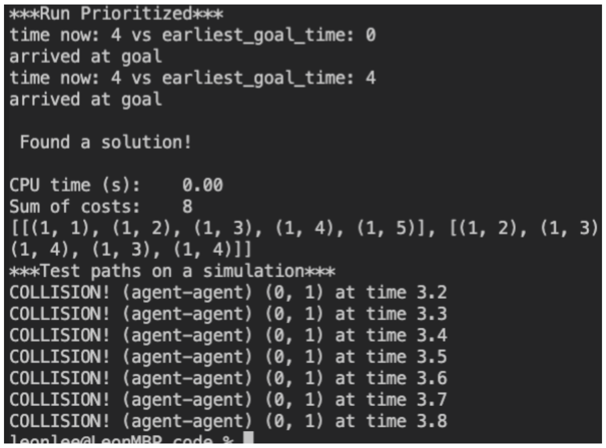


```
***Run Prioritized***
time now: 4 vs earliest_goal_time: 2
arrived at goal
time now: 4 vs earliest_goal_time: 2
arrived at goal

 Found a solution!

CPU time (s):    0.00
Sum of costs:    8
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3),
(2, 3), (1, 3), (1, 4)]]
***Test paths on a simulation***
leonlee@LeonMBP code %
```

Since there is no way to avoid collision with just one constraint, I had to put >1 constraints.
But our single_agent_planner.py is only designed to handle 1 constraint at most, so we have to account for this.
In is_constrained, we check for each constraint at each [next_time]. If there is a vertex constraint (like [(1,2)] AND the location constraint is = the next location, then there is constraint.
If there is an edge constraint (like [(1,2), (1,3)]), then there is violation if the timestep and the cell movement matches the constraint table.

For build_constraint_table, it's all cool if there are no agent+timesteps constraints that are duplicate (ie only 1 constraint for agent 0 at time x)
But if there are duplicates (ie agent 0 at time x cannot be at cell A & agent 0 at time x cannot be at cell B), then our dictionary can't handle that. In such case, we build a list, which would now allow us to simply append new constraints onto the list.

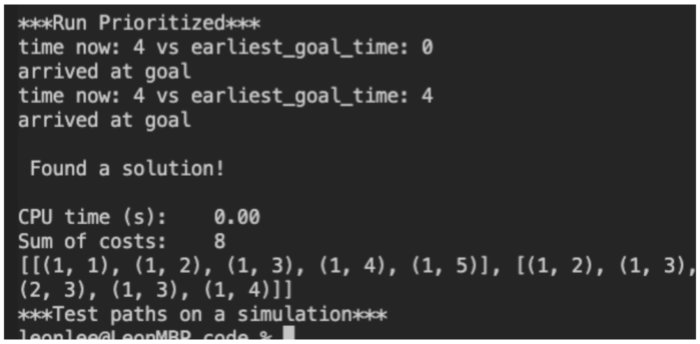
2.1



```
***Run Prioritized***
time now: 4 vs earliest_goal_time: 0
arrived at goal
time now: 4 vs earliest_goal_time: 4
arrived at goal

 Found a solution!

CPU time (s):    0.00
Sum of costs:    8
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3)
(1, 4), (1, 3), (1, 4)]]
***Test paths on a simulation***
COLLISION! (agent-agent) (0, 1) at time 3.2
COLLISION! (agent-agent) (0, 1) at time 3.3
COLLISION! (agent-agent) (0, 1) at time 3.4
COLLISION! (agent-agent) (0, 1) at time 3.5
COLLISION! (agent-agent) (0, 1) at time 3.6
COLLISION! (agent-agent) (0, 1) at time 3.7
COLLISION! (agent-agent) (0, 1) at time 3.8
leonlee@LeonMBP code %
```

For each agent, go through the path planned by

2.2

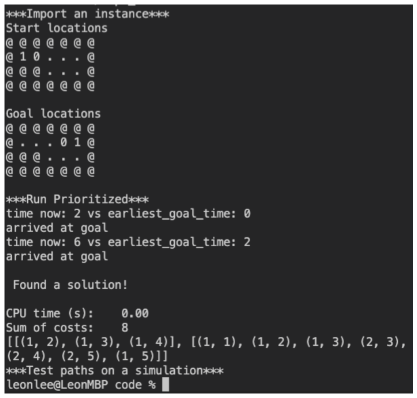


```
***Run Prioritized***
time now: 4 vs earliest_goal_time: 0
arrived at goal
time now: 4 vs earliest_goal_time: 4
arrived at goal

 Found a solution!

CPU time (s):    0.00
Sum of costs:    8
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3),
(2, 3), (1, 3), (1, 4)]]
***Test paths on a simulation***
leonlee@LeonMBP code %
```

Exact same concept as 2.1.

a_star. If a different agent share path, create constraint based on that, and add that to the table of constraints.

2.3

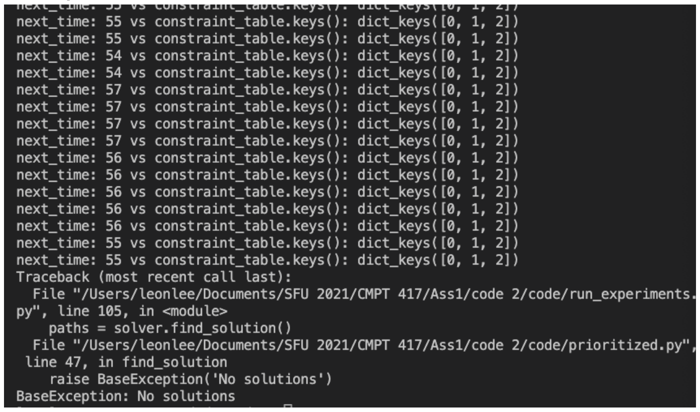


```
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ 1 0 . . . @
@ @ @ . . . @
@ @ @ @ @ @ @

Goal locations
@ @ @ @ @ @ @
@ . . . 0 1 @
@ @ @ . . . @
@ @ @ @ @ @ @

***Run Prioritized***
time now: 2 vs earliest_goal_time: 0
arrived at goal
time now: 6 vs earliest_goal_time: 2
arrived at goal

 Found a solution!

CPU time (s):    0.00
Sum of costs:    8
[[(1, 2), (1, 3), (1, 4)], [(1, 1), (1, 2), (1, 3), (2, 3),
(2, 4), (2, 5), (1, 5)]]
***Test paths on a simulation***
leonlee@LeonMBP code %
```

I ran:
python3 run_experiments.py --instance instances/exp2_2.txt --solver Prioritized

One current limitation is that after agent 0 reaches goal, agent 1 does not consider the constraint of where agent 0 sits, and might traverse right over agent 0. As the assignment suggests, we add support to consider the location constraint to all future timestep after agent 0 reaches goal.
In is_cnstrained, the current time +1 is the timestep of next step. Since constraint table is indexed by timesteps (let's say, n timesteps), agent 0 could produce, at most, n constraints. If agent 1 takes more than n timesteps to reach goal, then it needs to be aware to not trespass agent 0, who is sitting at goal since n. So we take the bigger value of next_time and the number of entries in the constraint dictionary table. If next_time(aka timestep taken by agent 1) is bigger, set it to the number of entries in constraint table.

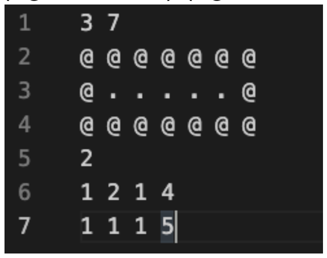
2.4
Using code from 2.3, the code would run on infinitely.
I changed the code according to the pdf, and gave it a map dimension limit.



```
next_time: 55 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 55 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 54 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 54 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 57 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 57 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 57 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 57 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 56 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 56 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 56 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 56 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 56 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 55 vs constraint_table.keys(): dict_keys([0, 1, 2])
next_time: 55 vs constraint_table.keys(): dict_keys([0, 1, 2])
Traceback (most recent call last):
  File "/Users/leonlee/Documents/SFU 2021/CMPT 417/Ass1/code 2/code/run_experiments.
py", line 105, in <module>
    paths = solver.find_solution()
  File "/Users/leonlee/Documents/SFU 2021/CMPT 417/Ass1/code 2/code/prioritized.py",
 line 47, in find_solution
    raise BaseException('No solutions')
BaseException: No solutions
```

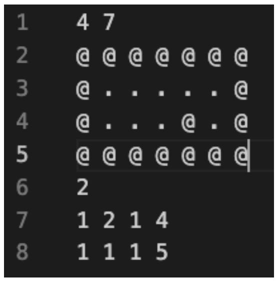
^ This is the output after the change.
I found that the maps were denoted given by . and @.
Either way, the map is defined by x amount of column and y amount of row.
So I just go len(my_map) to get rows, and len(my_map[0]) to get the columns.
To let the second agent explore all possible paths, we just * the agent by the dimension. (for agent 0, we can't just *0, so we just do (agent+1) to avoid this zero. For agent 1, (1+1) is plenty for traversing all cells, so if there is still no result found after (agent+1)*cell dimension, then there is truly no solution.)

2.5
This map is ran with 2.3 code, and finds no collision-free solution for a given ordering of agents (agent 0: 1,2) (agent1: 1,1)



```
3 7
@ @ @ @ @ @ @
@ . . . . . @
@ @ @ @ @ @ @
2
1 2 1 4
1 1 1 5
```

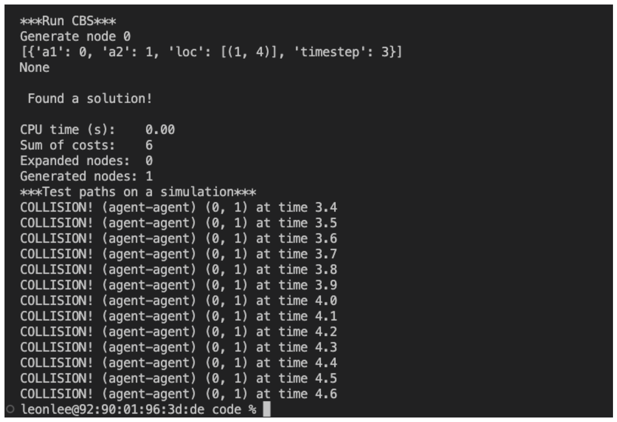
2.5 BONUS



```
4 7
@ @ @ @ @ @ @
@ . . . . . @
@ . . . @ . @
@ @ @ @ @ @ @
2
1 2 1 4
1 1 1 5
```

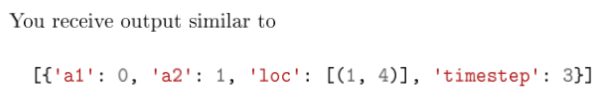
Agent 0 starts at 1,2.
Agent 1 starts at 1,1.
As Agent 0 reaches goal at (1,4), Agent 1 is blocked, since Agent 0 is given priority.
Agent 0 could've stalled first to let Agent 1 pass.

3.1



```
***Run CBS***
Generate node 0
[{'a1': 0, 'a2': 1, 'loc': [(1, 4)], 'timestep': 3}]
None

 Found a solution!

CPU time (s):    0.00
Sum of costs:    6
Expanded nodes:  0
Generated nodes: 1
***Test paths on a simulation***
COLLISION! (agent-agent) (0, 1) at time 3.4
COLLISION! (agent-agent) (0, 1) at time 3.5
COLLISION! (agent-agent) (0, 1) at time 3.6
COLLISION! (agent-agent) (0, 1) at time 3.7
COLLISION! (agent-agent) (0, 1) at time 3.8
COLLISION! (agent-agent) (0, 1) at time 3.9
COLLISION! (agent-agent) (0, 1) at time 4.0
COLLISION! (agent-agent) (0, 1) at time 4.1
COLLISION! (agent-agent) (0, 1) at time 4.2
COLLISION! (agent-agent) (0, 1) at time 4.3
COLLISION! (agent-agent) (0, 1) at time 4.4
COLLISION! (agent-agent) (0, 1) at time 4.5
COLLISION! (agent-agent) (0, 1) at time 4.6
○ leonlee@92:90:01:96:3d:de code %
```

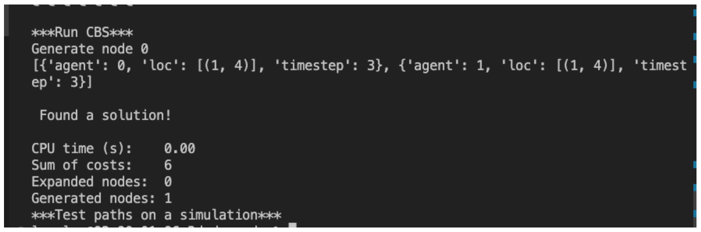
In detect_collision, pass in path 1 and 2. Find the longer path, and traverse that number of steps. That's t (timestep). If, at any of these steps, their paths collide(the big if statement), then we return that location and it's timestep as collision. We do this n^2 times in detect_collisions() via the two loops.
The output order is a little different, so I've formatted the output above to match the output order given below:



You receive output similar to

```
[{'a1': 0, 'a2': 1, 'loc': [(1, 4)], 'timestep': 3}]
```

3.2

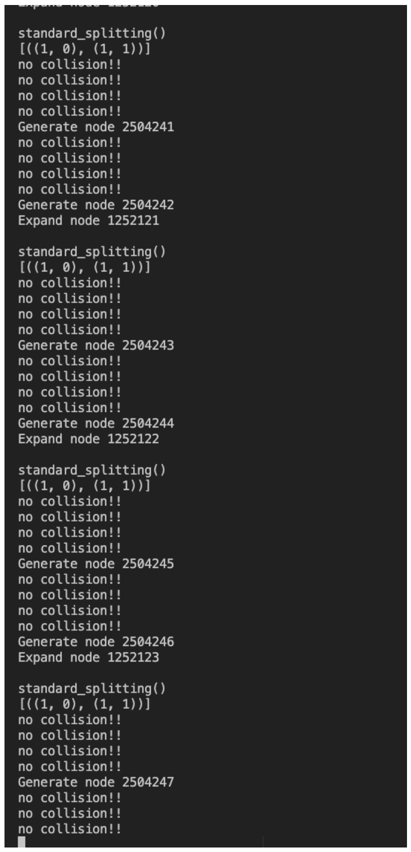


```
***Run CBS***
Generate node 0
[{'agent': 0, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 4)], 'timest
ep': 3}]

 Found a solution!

CPU time (s):    0.00
Sum of costs:    6
Expanded nodes:  0
Generated nodes: 1
***Test paths on a simulation***
```

In standard_splitting(), if there are more than 1 collisions, constraint1 for agent 1 is unchanged. Constraint2 for agent 2 needs to have it's location swapped, as per assignment pdf. So we set (x,y) to be 'loc': y,x.
If there are just 1 collisions, then no change.

My output is:
[{'agent': 0, 'loc': [(1, 4)], 'timestep': 3},
{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}]

3.3



```
standard_splitting()
[((1, 0), (1, 1))]
no collision!!
no collision!!
no collision!!
no collision!!
Generate node 2504241
no collision!!
no collision!!
no collision!!
no collision!!
Generate node 2504242
Expand node 1252121

standard_splitting()
[((1, 0), (1, 1))]
no collision!!
no collision!!
no collision!!
no collision!!
Generate node 2504243
no collision!!
no collision!!
no collision!!
no collision!!
Generate node 2504244
Expand node 1252122

standard_splitting()
[((1, 0), (1, 1))]
no collision!!
no collision!!
no collision!!
no collision!!
Generate node 2504245
no collision!!
no collision!!
no collision!!
no collision!!
Generate node 2504246
Expand node 1252123

standard_splitting()
[((1, 0), (1, 1))]
no collision!!
no collision!!
no collision!!
no collision!!
Generate node 2504247
no collision!!
no collision!!
no collision!!
```

It's been over 30 minutes and it's still going… I've reached the end before I swear, but I didn't screen shot it for proof.

3.4
See 3.3

In cbs.py -> find_solution(), we do the following while open_list is not empty: -Make list of constraints by calling standard_splitting(). -For each of these constraints, we move/copy each item in each constraint (ie ‘paths’ and ‘constraints’) into child/nextnode, then send these into a_star as parameter. -Let our a_star() do its job, and spit out a path. -Speaking of a_star(), I was running into issues where it says key error with locations like (1,-1). Since -1 doesn’t exist in my_map, I had to make sure the child node doesn’t go out of bound(moving outside of the map) so this would be no negative value for location, and node cannot go beyond the coordinates of the map: my_map’s row and col. -if the output path is good to go, we set the ‘path,’collisions’, and ‘cost’ and then push_node.	