CMPT 417 Individual Project Report

Task 1.1

```
(base) wangxubin@d207-023-185-192 code % python run_experiments.py --instance instances/exp1.txt --solver Independent
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ 0 1 . . . @
@ @ @ . @ @ @
@ @ @ @ @ @ @

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

***Run Independent***

 Found a solution!

CPU time (s):    0.00
Sum of costs:    6
***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
(base) wangxubin@d207-023-185-192 code %
```

Task 1.2

```
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
```

Task 1.4

Since agent 0 should not be at location (1,5) at time step 10, it means there maybe future constrains after an agent arriving its goal location, so, I consider the latest constrain into account. If the current time step is less than the time of the latest constrain of this agent, I choose to let this agent continue searching path to avoid the future constrain.

Task 1.5

For this part, I choose to let agent 1 leave path to let agent 0 pass by.

That means agent 1 should be at location (2, 3) at time step 2, so, my constrains are:

1. Prohibit agent 1 from being at (1, 4) at time step 2
2. Prohibit agent 1 from being at (1, 3) at time step 2
3. Prohibit agent 1 from being at (1, 2) at time step 2

Task 2.3

I choose to give a limited time limitation to the higher priorities; this makes sure that before that the time limitation, the future agents will not move the goal location of the higher priorities. I choose to set map size plus the “last” path size as the limitation of “this “agent. This method can partly fix the issue, the rest problem is that once all agents are in their goal location, the program will terminate until the time limitation of the very last agent. This takes extra time steps into account.

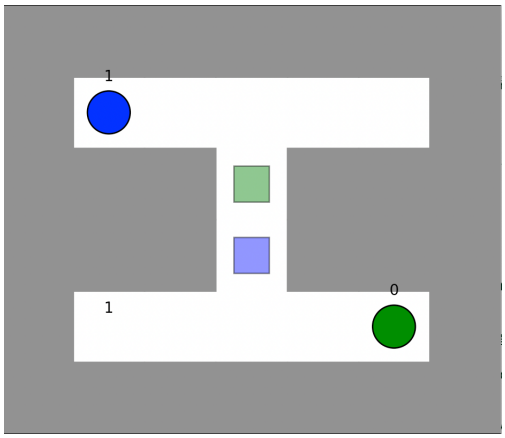
Task 2.4

The program can find a solution, since I have set a time limitation to each agent and the program will let the future agents wait until the end of the higher priorities and pass over them. The correct solution of this exp2.3.txt should be “No Solution”, since agent 0 is the priority one and once it gets to its goal location, there will be no path for agent 1 to get to its goal location and agent 1 will wait forever. In this situation, our program should return “No Solution”.

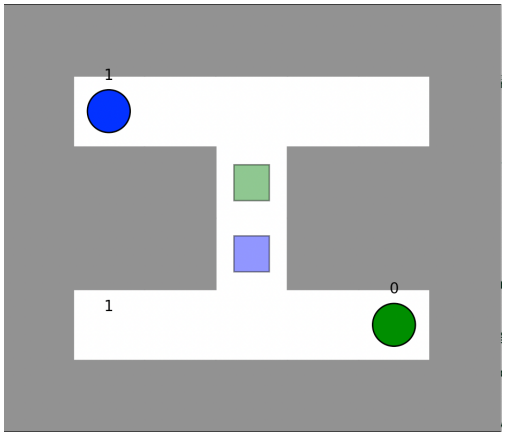
As instructor teach us in the pdf document, I give each agent an upper bound which is map size plus the sum of the path size of all past agents. If the path size of the current agent is larger than its upper bound, I consider it as an infinite waiting path, and return the “No Solution” expectation.

Task 2.5

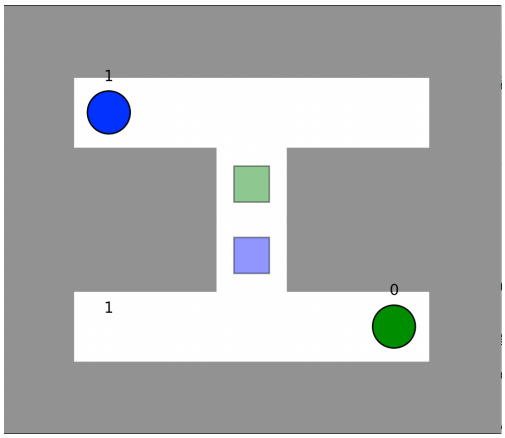
1.



2.



3.



Explanation: This prioritized planning will fail to find solutions if the paths of two of the agents are totally symmetric with each other through some one-way road, in this situation, whatever order it takes, one agent will stack on the path of the other one.

Task 3.3

```
***Run CBS***
Generate node 0
Expand node 0
Generate node 1
Generate node 2
Expand node 1
Generate node 3
Generate node 4
Expand node 2
Generate node 5
Generate node 6
Expand node 3
Generate node 7
Generate node 8
Expand node 6
Generate node 9
Generate node 10
Expand node 10
Generate node 11
Generate node 12
Expand node 12
Generate node 13
Generate node 14
Expand node 14
Generate node 15
Generate node 16
Expand node 16

 Found a solution!

CPU time (s):    0.00
Sum of costs:    8
Expanded nodes:  9
Generated nodes: 17
***Test paths on a simulation***
```

I test my code with exp2_1.txt and it gives me this transcript with 9 expanded nodes.

Task 3.4

The test files come with different instance with exp_* files, it might has no wall at the very last side of the map, so I have to add an if statement to check whether the child node is moving out of the map.

I tried to run with all test files in one shot and it takes me quit a long time, about 10 minutes, and some tests may have a few costs more than the correct result. I finally found out that was because the $Q.\text{paths} \leftarrow P.\text{paths}$ operation at line 15 in Algorithm 1. I used to just assign P[‘paths’] to Q[‘paths’], this may cause chain changes in the future, so there will be extra cost added. Then I choose to make a deep copy of the P[‘paths’] to ensure P[‘paths’] and Q[‘paths’] may not change simultaneously. And I got all correct results.

Task 4.1 & 4.2

I added the positive status in prioritized.py and modified my build_constraint_table and is_constrained functions.

In build_constraint_table function, I traverse through all constrains and add constrain to the table when a constrain has the same agent number with the current agent or if this constrain is a positive constrain, then I create a new constrain with reversed edge (reversed vertex constrain is still itself) and set it as a negative constrain and then add to the table.
In is_constrained function, I process the positive constraints by considering it as constrained if it has different location at the time step with the constrain. This will prohibit the selected agent from moving to the other 4 directions that is different with the direction in the constraint.
And then in the cbs.py, I implemented the disjoint_splitting function by setting both positive and negative vertex or edge constrain to the randomly selected agent.

Task 4.3

By running exp4.txt file, I got 15 expanded nodes with standard splitting and randomly 7 to 10 expanded nodes with disjoint splitting.

```
(base) wangxubin@wangxubindeMacBook-Pro code % python run_experiments.py --instance instances/exp4.txt --solver CBS
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ . 1 . . . @
@ 0 . . . . @
@ . . . . . @
@ . . . . . @
@ @ @ @ @ @ @
@ @ @ @ @ @ @

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

***Run CBS***

 Found a solution!

CPU time (s):    0.03
Sum of costs:    11
Expanded nodes:  15
Generated nodes: 29
***Test paths on a simulation***
```

```
(base) wangxubin@wangxubindeMacBook-Pro code % python run_experiments.py --instance instances/exp4.txt --solver CBS --disjoint
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ . 1 . . . @
@ 0 . . . . @
@ . . . . . @
@ . . . . . @
@ @ @ @ @ @ @
@ @ @ @ @ @ @

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

***Run CBS***
Now is using Disjoint Splitting

 Found a solution!

CPU time (s):    0.00
Sum of costs:    11
Expanded nodes:  7
Generated nodes: 13
***Test paths on a simulation***
```

Task 5

I downloaded the map of Boston. And convert the data online into the given instance. It has 256 * 256 map size and 1000 agents. My code got key error after running for

over 10 minutes. But I tested all 50 test files with my disjoint splitting, they all worked fine. I failed to test my code with the benchmarking.

But I still got some inspiration during implementing and debugging. The three algorithms are all great for understand MAPF thought. But I found some drawbacks of the prioritized algorithm. Since I was told to set the upper bound by the map size and the sum of the path size of all prioritized agents, this makes agents with lower priority wait prioritized agents to finishing first if lower ones have shorter path. This means if we have two agents, the first one has optimal path of size 5 and the second one has optimal path of size 3, by running prioritized algorithm, the second one will wait at its goal location until the first one finish its path and the second one's path may has extra cost, i.e., larger than the optimal size 3. I tried several methods to figure it out but failed.

The CBS and CBS disjoint is better than prioritized and more reasonable, it can figure out the problems above. And the disjoint version can cut the running time in half.