# Artificial Intelligence: Foundations and Applications

## Assignment 1
## Multi Agent Path Finding

**Members :**

- Anuraag Bhattacharya (16EE35002)
- Harsh Maheshwari (16EE35008)
- Ranajoy Sadhukhan (16EE35016)
- Siddhant Haldar (16EE35018)

## Proposed Algorithm

- Given a robot and its set of tasks, we divide the problem into 2 parts -
    - Obtain the optimal path for every task for every robot.
    - Considering the set of optimal paths, find out collisions(multiple robots occupying the same position at a particular time step) between robots and resolve these collisions.
- **Finding the optimal path for each robot**
    - Consider a robot with a set of tasks assigned to it. Each task is divided into 3 parts - (A) init to pickup, (B) pickup to delivery, and (C) delivery to the final position.
    - For every part of the task, find out the optimal path to go from start to goal using the Breadth-First Search (BFS) algorithm.
    - If the final position(F) of a task is different from the init position(I) of the next task, find out the optimal path to go from F to I using the BFS algorithm.
    - Do the same for all robots for every task.
    - **Example:** Consider robot R1 with tasks T1, T2, T3. We obtain the optimal path using BFS for tasks T1, T2, and T3 and also for the path between (a) end point of T1 and starting point of T2 (b) end point of T2 and starting point of T3.
- **Resolving conflicts**
    - Run a loop over time steps such that every iteration of the loop performs operations with regards to a particular time step. Start from t=0 and keep iterating till all tasks are completed.
    - In a particular iteration, use the calculated optimal paths for each robot simultaneously to obtain the positions of all robots at that time step.
    - Add the positions of all robots to a set. Whenever 2 or more robots occupy the same position, we say there is a collision.
        - To resolve the collision, we need to choose a winner robot that is allowed to stay at the position and the rest are required to replan their paths from

the parent cell (and are not allowed to inhabit the cell under consideration at this particular time step).
- In case of a collision, the robot that is closest to finishing the subpart(A, B or C) of its task is allowed to stay. The rest are required to replan.
- **RE-PLAN:** For each robot whose path needs to be replanned, we make it move a step back to its previous position. Now, we obtain the optimal path from this point till the end, not permitting the robot to move to the position where the conflict occurs. Then this robot moves a step along its newly planned path.
- This is done for all the robots involved in a conflict.

- These steps are continued till all tasks are completed.

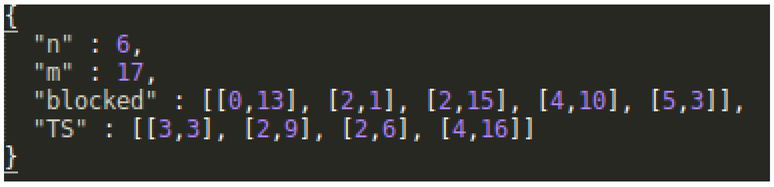
## Execution

- **Read graph:** The graph is defined in a JSON file with 4 parameters -
  - *n* : Height of the matrix.
  - *m* : Width of the matrix.
  - *blocked* : Blocked cells (containing obstacles) in the matrix.
  - *TS* : Temporary storage locations in the matrix.
- **Read tasks:** The tasks are defined in *task.txt* in the format - [*robot_name,init_h,init_w,final_h,final_w,pickup_h,pickup_w,deliver_h,deliver_w*]
- **main.py** - Contains the main execution of the code. This is the file that must be executed to perform the task.
- **robot.py** - A class for robots is defined in this file.
- **utils.py** - Contains functions for reading the graph, reading the tasks, find the shortest path, and determining if all the tasks have been completed or not.

## Results

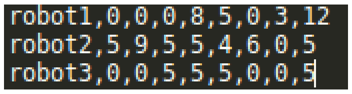
The results for a given set of inputs are printed on the terminal.

- **Sample 1**
  - **Input Graph**



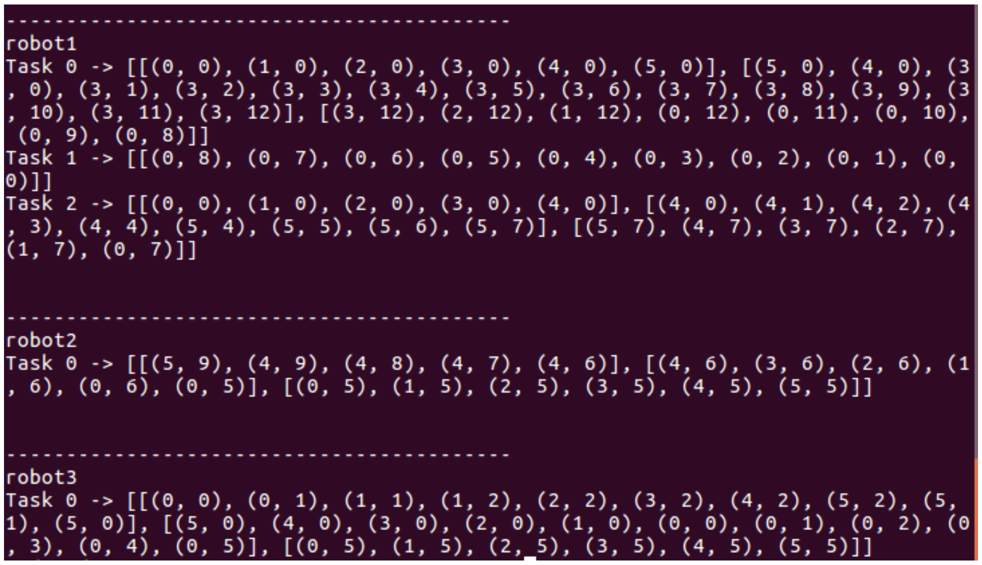
```
{
  "n" : 6,
  "m" : 17,
  "blocked" : [[0,13], [2,1], [2,15], [4,10], [5,3]],
  "TS" : [[3,3], [2,9], [2,6], [4,16]]
}
```

  - **Input Task (Each robot has just one task)**



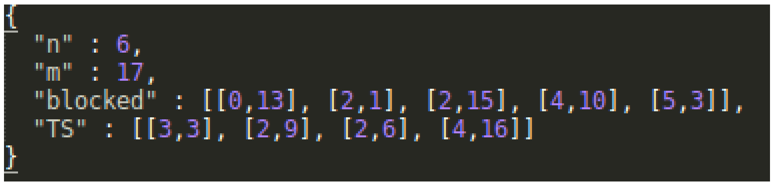
```
robot1,0,0,0,8,5,0,3,12
robot2,5,9,5,5,4,6,0,5
robot3,0,0,5,5,5,0,0,5
```

  - **Optimal path printed on the terminal**



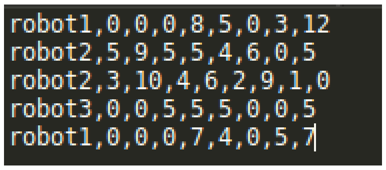
```
------------------------------------------
robot1
Task 0 -> [[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0)], [(5, 0), (4, 0), (3
, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3
, 10), (3, 11), (3, 12)], [(3, 12), (2, 12), (1, 12), (0, 12), (0, 11), (0, 10),
 (0, 9), (0, 8)]]
Task 1 -> [[(0, 8), (0, 7), (0, 6), (0, 5), (0, 4), (0, 3), (0, 2), (0, 1), (0,
0)]]
Task 2 -> [[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0)], [(4, 0), (4, 1), (4, 2), (4
, 3), (4, 4), (5, 4), (5, 5), (5, 6), (5, 7)], [(5, 7), (4, 7), (3, 7), (2, 7),
(1, 7), (0, 7)]]


------------------------------------------
robot2
Task 0 -> [[(5, 9), (4, 9), (4, 8), (4, 7), (4, 6)], [(4, 6), (3, 6), (2, 6), (1
, 6), (0, 6), (0, 5)], [(0, 5), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5)]]


------------------------------------------
robot3
Task 0 -> [[(0, 0), (0, 1), (1, 1), (1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5,
1), (5, 0)], [(5, 0), (4, 0), (3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (0
, 3), (0, 4), (0, 5)], [(0, 5), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5)]]
```

- **Sample 2**
  - **Input Graph**



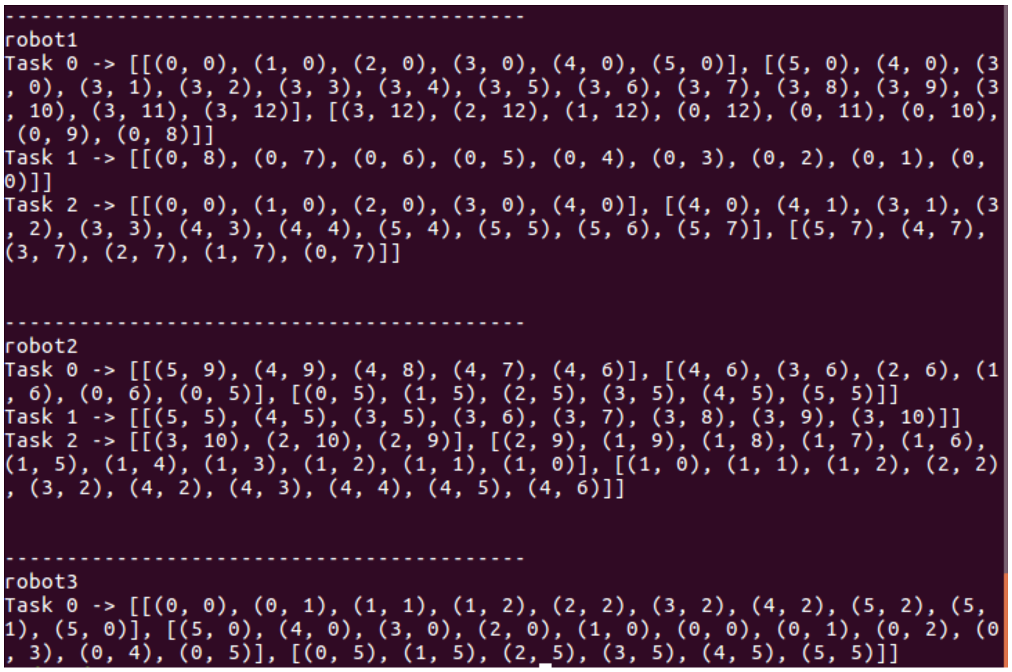
```
{
  "n" : 6,
  "m" : 17,
  "blocked" : [[0,13], [2,1], [2,15], [4,10], [5,3]],
  "TS" : [[3,3], [2,9], [2,6], [4,16]]
}
```

  - **Input Task (Robots with multiple tasks)**



```
robot1,0,0,0,8,5,0,3,12
robot2,5,9,5,5,4,6,0,5
robot2,3,10,4,6,2,9,1,0
robot3,0,0,5,5,5,0,0,5
robot1,0,0,0,7,4,0,5,7
```

  - **Optimal path printed on the terminal**



```
-----------------------------------------
robot1
Task 0 -> [[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0)], [(5, 0), (4, 0), (3
, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3
, 10), (3, 11), (3, 12)], [(3, 12), (2, 12), (1, 12), (0, 12), (0, 11), (0, 10),
 (0, 9), (0, 8)]]
Task 1 -> [[(0, 8), (0, 7), (0, 6), (0, 5), (0, 4), (0, 3), (0, 2), (0, 1), (0,
0)]]
Task 2 -> [[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0)], [(4, 0), (4, 1), (3, 1), (3
, 2), (3, 3), (4, 3), (4, 4), (5, 4), (5, 5), (5, 6), (5, 7)], [(5, 7), (4, 7),
(3, 7), (2, 7), (1, 7), (0, 7)]]


-----------------------------------------
robot2
Task 0 -> [[(5, 9), (4, 9), (4, 8), (4, 7), (4, 6)], [(4, 6), (3, 6), (2, 6), (1
, 6), (0, 6), (0, 5)], [(0, 5), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5)]]
Task 1 -> [[(5, 5), (4, 5), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10)]]
Task 2 -> [[(3, 10), (2, 10), (2, 9)], [(2, 9), (1, 9), (1, 8), (1, 7), (1, 6),
(1, 5), (1, 4), (1, 3), (1, 2), (1, 1), (1, 0)], [(1, 0), (1, 1), (1, 2), (2, 2)
, (3, 2), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6)]]


-----------------------------------------
robot3
Task 0 -> [[(0, 0), (0, 1), (1, 1), (1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5,
1), (5, 0)], [(5, 0), (4, 0), (3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (0
, 3), (0, 4), (0, 5)], [(0, 5), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5)]]
```