# 1. Intersection Management (16 pts)

假設一個conflict zone同時只有一輛車能夠進入

若只有一個conflict zone, 因為沒有transition所以resource conflict graph中並沒有點(vertex), 因此resource conflict graph中也不會有cycle, 因此只要讓維持conflict zone中只有一輛車, 等到該車完全通過後再放另一輛車進入conflict zone, 依序執行即可。

挑選下一輛車的方法, 我可能會選擇minimize average waiting time, 因為這樣的假設下, 車子通過conflict zone的效率很差, 因此使用minimal average waiting time, 因此通常不會設置在交通繁忙的十字路口, 較有可能設置在車輛較少的郊區,。

假設一個conflict zone同時能有多輛車能夠進入

為了避免車輛的碰撞, 我會選擇同時只讓同一條車道的車進入conflict zone, 利用車子自身的ACC/CACC(假設車子是有安裝的)確保前後車的距離, 此種方式因為也只有一個conflict zone, 沒有transition, 因此也不會有deadlock。

挑選車道的方法, 我可能會依照Back – Pressure的方式來選擇, Back – Pressure 中每個Phase的來源(source)車道都只會有一條, 且根據道路的大小來調整v的大小。

# 2. Realization of Level-X Autonomy

| 自動駕駛分級 | 名稱 | 定義 | 駕駛操作 | 周邊監控 | 接管 | 應用場景 |
|---|---|---|---|---|---|---|
| L0 | 人工駕駛 | 由人類駕駛員全權駕駛車輛 | 人類駕駛員 | 人類駕駛員 | 人類駕駛員 | 無 |
| L1 | 輔助駕駛 | 車輛對方向盤和加減速中的一項操作提供駕駛，人類駕駛員負責其餘的駕駛動作 | 人類駕駛員和車輛 | 人類駕駛員 | 人類駕駛員 | 限定場景 |
| L2 | 部分自動駕駛 | 車輛對方向盤和加減速中的多項操作提供駕駛，人類駕駛員負責其餘的駕駛動作 | 車輛 | 人類駕駛員 | 人類駕駛員 | |
| L3 | 條件自動駕駛 | 由車輛完成絕大部分駕駛操作，人類駕駛員需保持注意力集中以備不時之需 | 車輛 | 車輛 | 人類駕駛員 | |
| L4 | 高度自動駕駛 | 由車輛完成所有駕駛操作，人類駕駛員無需保持注意力集，但限定道路和環境條件 | 車輛 | 車輛 | 車輛 | |
| L5 | 完全自動駕駛 | 由車輛完成所有駕駛操作，人類駕駛員無需保持注意力集中 | 車輛 | 車輛 | 車輛 | 所有場景 |

https://zh.wikipedia.org/wiki/%E8%87%AA%E5%8B%95%E9%A7%95%E9%A7%9B%E6%B1%BD%E8%BB%8A

1. Level - 3 : 我認為目前有些車廠(Ex : Tesla, 奧迪, Benz, etc)的高級車款已經可以達到L3的自動駕駛了, 且相關法規也在陸續完善中, 因此我認為Level - 3的自動駕駛會在2年內普及。

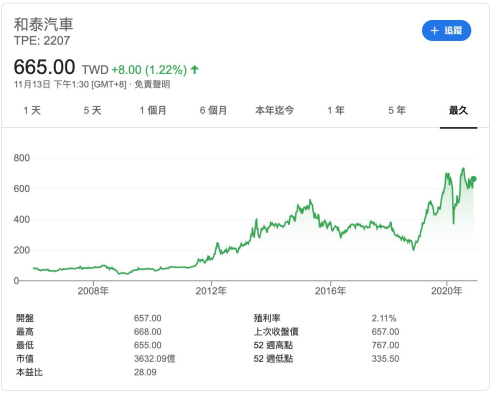
2. Level - 4 : 我認為以目前的車廠/學界的技術其實已經幾乎可以製造出L4等級的自駕車。其實現在的道路對自駕車的駕駛系統是相對困難的挑戰, 若未來自駕車的市佔率提升, 未來可能會有為了自駕車安全而安裝road side sensor, 以及車間通訊的設備, 這些都使的Level - 4 等級的自駕車有實現的可能。最後因為特斯拉的緣故, 2019年後車廠的股票都開始瘋漲,

特斯拉


奧迪


B MW


和泰汽車




在有資金, 有技術的情況下, 我認為L4的自駕車開始出現在市場應該會是在約4~5年後

3. Level - 5 : 我認為這個等級的自駕車, 最大的問題已經不是在技術, 而是在自駕車背後所衍生出的道德/法律問題, 因此我認為除非車廠可以負責此時的自駕車所衍生的相關問題, 不然L5的自家車應該是不會出現的。因此我認為, 哪一天當自駕車所衍生出的相關道德/法律問題被大家解決的那時候, 也就是Level - 5自駕車要問世的時候。

# 3. Cycle Removal (30 pts)

1. No, there is no cycle in the graph
2. Yes, there is a cycle in the graph, and the edge I removed is (3, 0)
3. Source code

```python
class Node():
    def __init__(self, val):
        self.val = val
        '''
        while = 0
        gray = 1
        black = 2
        '''
        self.color = 0


class Graphic():
    def __init__(self, filename):

        self.num_vertices = 0
        self.num_edges = 0
        self.nodes = []
        self.adjlist = []

        try:
            f = open(filename)
            self.num_vertices = int(f.readline())
            self.num_edges = int(f.readline())
            self.nodes = [Node(i) for i in range(self.num_vertices)]
            self.adjlist = [set() for _ in range(self.num_vertices)]


            for line in f.readlines():

                start, end = line.split()
                start, end = int(start), int(end)
                self.adjlist[start].add(end)

```

```python
        except:
            print(f"Error loading file from {filename}")

    def dfs(self):

        def inner_recursive(node):

            for child in self.adjlist[node]:
                if self.nodes[child].color == 0: # white
                    self.nodes[child].color # Visited, change color to gray
                    inner_recursive(child)

                elif self.nodes[child].color == 1: # gray
                    print(f"Back Edges : {(node, child)}")

            self.nodes[node].color = 2 # Finish, change color to black

        for start_node in self.nodes:

            '''
            while = 0
            gray = 1
            black = 2
            '''
            if start_node.color == 0:
                start_node.color = 1
                inner_recursive(start_node.val)


```