If there is only one conflict zone, then there is no type-1 edge in graph G'. However, resource conflict graph H' is created by merging nodes of type-1 edge in G'. As a result, the graph H' of such special case G' doesn't exist.

The goal of building graph H' is to avoid deadlock of multiple conflict zones in the intersection. In special case, there is only one conflict zone in the intersection. This means that there is impossible occurring deadlock. Therefore, the solution is to just solve the cyclic and acyclic problem of graph G. The acyclic graph G' is the final answer.

Level 3

儘管現在 Tesla 能做到基本上符合 level 3 的自動操作，但駕駛仍需保持專注並時刻掌握方向盤，視情況拿回控制權，故無法到達 level 3 的標準。不過有鑑於此，我認為 level 3 在 5 年內應該可以實現，因為表示現有技術其實可以達到類似的標準，只是安全上仍有疑慮而已，只要能夠克服駕駛在做自己事情時對突發狀況的控制以及危險的避免，就能完美做到 level 3 的標準，尤其是在封閉型道路如高速公路、快速道路以及高架道路等路線及路權單純的道路，應能更早應用 level 3 的自駕車技術。

Level 4

就車輛本身而言，相關的辨識以及控制技術其實目前可以做得不錯，而且大多數先進國家都有詳細的圖資系統來幫助自駕車的訓練，就連在台灣，中華電信與勤崴科技合作的自駕車也已能夠在單獨的情況下在封閉訓練場良好運轉，所以我認為這部分不是主要的 bottleneck。主要的問題應當是在車輛間的通訊與控制，包括路口優先權、位置資訊分享、危險資訊分享等，當每輛車不再是個體各自行動而是群體互相配合行動的時候，那麼才能避免風險如相撞、追撞等，並達到全然的自動控制。此外法律也是問題，若法律仍然跟不上科技變遷，因擔憂碰撞的肇事責任或是機械故障等問題而持續對自駕車抱持負面態度，那 level 4 的自駕車仍難以上路。有鑑於此，我認為要到達 level 4 至少還有 5-10 年，而必須克服的就是車聯網及法律的部分。

Level 5

除了要先克服 level 4 的問題之外，level 5 的自駕車也必須克服惡劣氣候與情況，在軟體方面，如果是開在預定外的道路或是沒有足夠的地圖資訊，那麼車輛必續自己決定如何避免風險，那麼更強的辨識與決策功能是必須的，能夠在惡劣氣候與環境下辨識道路、行人、環境或是其他車輛並做出行駛的決策。此外，硬體也必須有所提升，例如控制、打滑，或甚至是結凍，以解決惡劣氣候可能會造成的負面影響。當車輛能夠自己決策並在惡劣情況下仍能夠運行的話，那麼無方向盤的完全自駕車才可能實現，而我認為這必須至少 15-20 年才能成真。

1.

Acyclic

2.

Cyclic

Remove 3 -> 0

3.

```
import numpy as np
import time

output = []

def readData(filename):
    with open(filename, "r") as f:
        Nvertex = int(f.readline())
        Nedges = int(f.readline())
        result = [[] for i in range(Nvertex)]
        for i in range(Nedges):
            s, t = [int(x) for x in f.readline().split()]
            result[s].append(t)
    return result

def dfs(node, graph, record):
    if record[node] == 1:
        return 1
    else:
        record[node] = 1
    for index in range(len(graph[node])):
        i = graph[node][index]
        tmp = np.array(record)
        result = dfs(i, graph, record)
        record = tmp.tolist()
        if result == 1:
            if (node, i) not in output:
                print("cycle detect!!")
                print("remove edge of {} to {}".format(node, i))
                graph[node].pop(index)
                output.append((node,i))
    return 0
```

```
graph = readData("input1.dat")
for i in range(len(graph)):
    record = [0 for i in range(len(graph))]
    dfs(i, graph, record)
print('input1 done')
print('--------------')
graph = readData("input2.dat")
for i in range(len(graph)):
    record = [0 for i in range(len(graph))]
    dfs(i, graph, record)
print('input2 done')
```