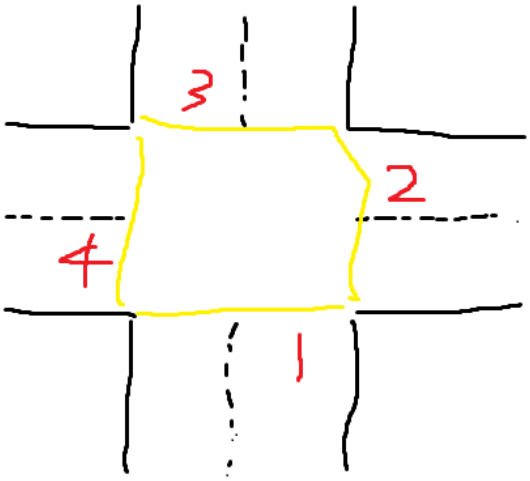
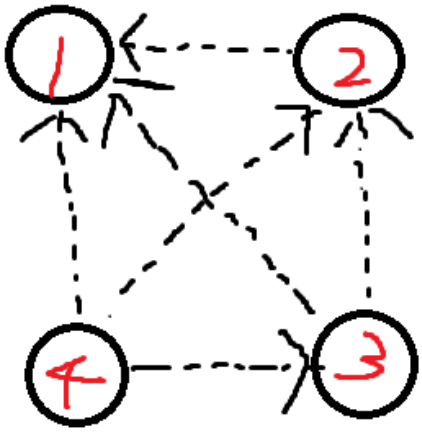
1. Intersection Management

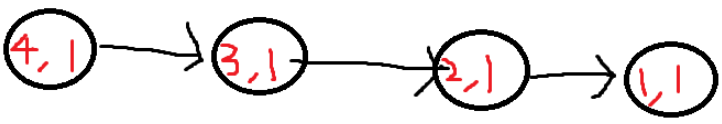
當只有一個 conflict zone 時，代表只要有一台車在 conflict zone 裡面，其他車都必須等他結束整個行車路徑才能移動。

舉個例子(下圖一)，這時 timing conflict zone G 會呈現沒有 Type-1 的 edge 而且所有的 vertex 都會有 Type-3 的 edge 和其他任何 vertex 互指，這時把一些 edge 移除之後可以形成沒有 cycle 的 G'(下圖二)，因為 conflict zone 只有一個的關係，resource conflict zone 就會呈現以下(下圖三)，這時無 cycle 也就是沒有 deadlock 的情況。

所以解決方式是增加 conflict zone 的數量或是規定十字路口哪個方位來的車子有優先權，由以下例子為(西，北，東，南)。







2. Realization of Level-X Autonomy

現在市場上有穩定供應的是有 ADAS 的 level-1 和 level-2 的自駕車，不過在這兩個階段，人仍然必須全程全神貫注在開車上，自動化的系統儼然只是輔助行車安全的部分。

到了 level-3 之後才真的有在部分路段，例如:高速公路這種設計較為直觀，且除了車子不太會有其他額外物件的地方，可以讓自動化的系統去接管車子的駕駛。不過相較於 level-4 和 level-5，level-3 的自動車仍然需要人為去接管突發狀況，也就是隨時需要準備接手，取代系統操控車子。而 level-4 之後，是強調已經可以不需要有駕駛的存在，也就是自動化駕駛系統足以應變各種突發狀況。

我個人認為 level-3 的車子可以真的在台灣路上真實呈現大概還需要 5~7 年的時間。在技術層面上就算達到了，但是台灣相關的法規還有駕駛的自駕行車教育還不完善的形況下，還有很大的努力空間。最重要的是，假如發生行車糾紛，甚至車禍時，到底該如何調停與裁罰，這也還是很大的問題。

而 level-4 的車子我認為大概還需要 10~15 年才能真正上路。因為相較於 level-3 不只技術層面提升很多， ADS 系統的升級，這個價格的差距也未必是所有消費者都能接受的。

最後 level-5 的車子我認為需要 20 年以上，因為 level-5 主打的就是，相比於 level-4，可以無視任何地形、地理位置、道路狹長和車流車況等等。也就是說自動駕駛系統能到達人腦的地步。但是要達到 100%安全性，這技術性的障礙瓶頸，會是最後的一道關卡。

而最後駕駛的接受度還是必須要看個人，因為對我來說 level-3 的自駕車就很足夠。我不太能想像在高速行車的途中，完全分神聊天講話，打遊戲或是倒頭就睡等等。

不過在現代科技發達的時代，各大廠都在追求技術的頂尖，也期待 Google's Waymo 或 Telsa 等等這些自駕車先驅哪一天真正實現 level-5 的自駕車。

3. Cycle Removal

(1) No

(2) Yes，有兩個 cycle，但是移除掉(3,0)這條邊就可以直接使 graph acyclic

(3) Code Segment

將 input 檔輸入，並將資料進行切割

```
edges = []
with open('input1.dat', 'r', encoding='UTF-8') as file:
    vertex_num = float(file.readline())
    edge_num = float(file.readline())
    for data in file.readlines():
        data = data.strip()
        data = data.split()
        edges.append(data)
```

While 迴圈測試有無 cycle(跑完迴圈無 output 代表沒有 cycle，有 output 則代表有 cycle 出現，且移除該邊能使此 cycle 不存在)
測試結果為移除(3,0)就能使 cycle 消失。

```
edge_start_vertex = []
remove_list = []
empty = []

while(edges != []):
    edge_start_vertex = []
    for i in range(len(edges)):
        edge_start_vertex.append(edges[i][0])
    s = edges[0][1]
    v = edges[0][0]
    while(s in edge_start_vertex):
        # print(len(edge_start_vertex),len(edges))
        idx = edge_start_vertex.index(s)
        s = edges[idx][1]
        if s == v :
            print("cycle!")
            print(edges[0])
            break
        edges.remove(edges[idx])
        edge_start_vertex.remove(edge_start_vertex[idx])
    edges.remove(edges[0])
```

以下為用現成 library 偵測 cycle 是否存在。
得證移除(3,0)的確能使兩個 cycle 都消失。

```
G = nx.DiGraph(edges)
num_of_cycle = 0
for cycle in nx.simple_cycles(G):
    print(cycle)
    num_of_cycle += 1
print("總共有", num_of_cycle, "個cycle")
```

```
[0, 1, 3]
[0, 2, 3]
總共有 2 個cycle
```