程式說明

1. 建立圖

(1) Map node class

```python
# 地圖class
class MAP:
    def __init__(self, name, toPoint, bePoint, beWalk):
        self.name = name
        self.toPoint = toPoint  # 指向誰
        self.bePoint = bePoint  # 被誰指
        self.beWalk = beWalk    # 記錄有沒有被走過(boolean)
```

(2) 建立地圖

```python
def make_map(self):
    self.map_list = []  # 儲存map資訊

    # 開始存map資訊
    # A
    temp_map = MAP('A', ['D'], [], False)
    self.map_list.append(temp_map)

    # B
    temp_map = MAP('B', ['D'], [], False)
    self.map_list.append(temp_map)

    # C
    temp_map = MAP('C', ['E'], [], False)
    self.map_list.append(temp_map)

    # D
    temp_map = MAP('D', ['G', 'H'], ['A', 'B'], False)
    self.map_list.append(temp_map)

    # E
    temp_map = MAP('E', ['I'], ['C', 'H'], False)
    self.map_list.append(temp_map)

    # F
    temp_map = MAP('F', ['I', 'J'], [], False)
    self.map_list.append(temp_map)
```

```python
    # G
    temp_map = MAP('G', ['K'], ['D'], False)
    self.map_list.append(temp_map)

    # H
    temp_map = MAP('H', ['E', 'K'], ['D'], False)
    self.map_list.append(temp_map)

    # I
    temp_map = MAP('I', ['L'], ['E', 'F'], False)
    self.map_list.append(temp_map)

    # J
    temp_map = MAP('J', ['M'], ['F'], False)
    self.map_list.append(temp_map)

    # K
    temp_map = MAP('K', [], ['G', 'H'], False)
    self.map_list.append(temp_map)

    # L
    temp_map = MAP('L', [], ['I'], False)
    self.map_list.append(temp_map)

    # M
    temp_map = MAP('M', [], ['J'], False)
    self.map_list.append(temp_map)
```

2. 輸入題目

```python
def input_question(self):
    self.start_point = input("start point : ")
    input_z_point = input("z set(以逗號分隔) : ")

    self.z_point = input_z_point.split(',')
```

3. Find collider

```
def find_collider(self):
    self.collider_list = []       # 被兩個箭頭所指向的點
    head_list = []           # 箭頭所指向的點
    tail_list = []           # 箭尾的點

    for i in self.map_list:
        if len(i.bePoint) > 0:
            head_list.append(i.name)
        if len(i.toPoint) > 0:
            tail_list.append(i.name)

        if len(i.bePoint) > 1:
            self.collider_list.append(i.name)

        for j in i.bePoint:
            tail_list.append(j)

        for j in i.toPoint:
            head_list.append(j)
```

4. Find unblock

```
def find_unblock(self):
    self.unblock_list = []

    for i in self.collider_list:
        descendant = []
        descendant.append(i)

        number = 0
        while number < len(descendant):
            descendant = self.searchMap_zSet(descendant[number], descendant)
            number += 1

        for j in descendant:
            if self.judge_zSet(j):
                self.unblock_list.append(i)

    print('unblock', self.unblock_list)
```

使用 function

(1) Search map z set

如果 point 有當其他點的 tail，那就加入那一個點

```
def searchMap_zSet(self, point, d_list):
    for map in self.map_list:
        # 去查每個的tail，若有point，就將head加入list
        for i in map.bePoint:
            if i == point:
                d_list.append(map.name)

    return d_list
```

(2) Judge z Set

判斷該點是否在 z set 中

```
def judge_zSet(self, point):
    for i in self.z_point:  # 判斷是否在z set中
        if i == point:
            return True

    return False
```

5. Initial map

將所有地圖上的點都設為沒走過

```
def initial_map(self):
    for i in self.map_list:
        i.beWalk = False
```

6. Build path

依照輸入的起點開始走地圖，用 BFS 紀錄所走過的點

```
def build_path(self):
    self.path_stack = []

    start_location = self.start_point    # 起點

    temp_stack = STACK(start_location, None)
    self.path_stack.append(temp_stack)

    number_of_stack = 0     # 現在在stack的第幾層

    while number_of_stack < len(self.path_stack):

        # 現在位置
        location_now = self.path_stack[number_of_stack].now
        location_comeFrom_number = self.path_stack[number_of_stack].comeFrom
        # 查詢從哪裡來 以防無限循環
        if location_comeFrom_number != None:
            location_comeFrom = self.path_stack[location_comeFrom_number].now
        else:
            location_comeFrom = -1

        # print(location_now)
        # print(location_comeFrom)
```

```
        # 查詢地圖
        now_map = None
        for i in self.map_list:
            if i.name == location_now:
                now_map = i
                break

        # 標記已經被走過
        now_map.beWalk = True

        # 儲存toPoint和bePoint資料
        now_toPoint = now_map.toPoint
        now_bePoint = now_map.bePoint

        # print(now_toPoint)
        # print(now_bePoint)

        # 增加stack
        # 儲存 toPoint
        for i in now_toPoint:
            if self.judge_beWalk(i):
                continue

            if i != location_comeFrom:   # 從誰來的不能存
                temp_stack = STACK(i, number_of_stack)
                self.path_stack.append(temp_stack)
```

```
        # 儲存 bePoint
        for i in now_bePoint:
            if self.judge_beWalk(i):
                continue

            if i != location_comeFrom:  # 從誰來的不能存

                temp_stack = STACK(i, number_of_stack)
                self.path_stack.append(temp_stack)

        # print('len fo path stack', len(self.path_stack))

        # 前往下一個stack
        number_of_stack += 1
        # print('number of stack', number_of_stack)
```

7. Find path

記錄所經過的每一點

```
def find_path(self):
    self.point_list = []

    for i in range(len(self.path_stack)):
        self.point_list.append(self.path_stack[i].now)
```

8. d_separation

若 d_connect_list 不等於 past_list 的話，就繼續跑 loop 直到兩者相等

```
def d_separation(self):

    self.d_connect_list = []
    self.d_connect_list.append(self.start_point)
    past_list = []

    while self.d_connect_list != past_list:
        for i in self.d_connect_list:
            if i not in past_list:
                past_list.append(i)
                if not self.judge_zSet(i):

                    if i in self.unblock_list:
                        self.d_connect_list = self.searchMap_zSet_head(i, self.d_connect_list)
                        self.d_connect_list = self.searchMap_zSet_tail(i, self.d_connect_list)

                    else:
                        self.d_connect_list = self.searchMap_zSet_tail(i, self.d_connect_list)

                self.d_connect_list = sorted(set(self.d_connect_list))
                past_list = sorted(set(past_list))


    print(self.d_connect_list)
    self.different_set()
```

使用 function

(1) judge z set

判斷是否在 z set 中

```
def judge_zSet(self, point):
    for i in self.z_point:  # 判斷是否在z set中
        if i == point:
            return True

    return False
```

(2) search map z set head

去找 map 中每個點誰指向 point，而該點就是 tail，若有則判斷是否為 z set 中的 value，若不在 z set 中，就加入到 list 中

```
def searchMap_zSet_head(self, point, list):
    for map in self.map_list:
        # 去查每個的to point，若有point，就將tail加入list
        for i in map.toPoint:
            if i == point:
                if not self.judge_zSet(map.name):
                    list.append(map.name)

    return list
```

(3) search map z set tail

去找 map 中每個點誰被 point 指向，而該點就是 head，若有則判斷是否為 z set 中的 value，若不在 z set 中，就加入到 list 中

```
def searchMap_zSet_tail(self, point, list):
    for map in self.map_list:
        # 去查每個的be point，若有point，就將head加入list
        for i in map.bePoint:
            if i == point:
                if not self.judge_zSet(map.name):
                    list.append(map.name)

    return list
```

(4) different set

將 d_connect_list 與所有的 node list（original_set）取差集，就會得到最終答案 d_separation

```
def different_set(self):
    original_set = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M']
    d_separation = []
    for i in original_set:
        if i not in self.d_connect_list:
            d_separation.append(i)

    print(d_separation)
```

執行結果

(a)

```
start point : A
z set(以逗號分隔) : K,J
unblock ['D', 'D', 'K']
['A', 'B', 'D', 'E', 'G', 'H', 'I', 'L']
['C', 'F', 'J', 'K', 'M']
```

The predicted set Y is [C, F, J, K, M]，J 有在 d-separation 中
因此此題答案為 **TRUE**

(b)

```
start point : G
z set(以逗號分隔) : D
unblock ['D']
['G', 'K']
['A', 'B', 'C', 'D', 'E', 'F', 'H', 'I', 'J', 'L', 'M']
```

The predicted set Y is [A, B, C, D, E, F, H, I, J, L, M]，L 有在 d-separation 中
因此此題答案為 **TRUE**

(c)

```
start point : B
z set(以逗號分隔) : C,L
unblock ['D', 'E', 'I']
['A', 'B', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'M']
['C', 'L']
```

The predicted set Y is [C, L]，F 沒有在 d-separation 中
因此此題答案為 **FALSE**

(d)

```
start point : A
z set(以逗號分隔) : K,E
unblock ['D', 'D', 'D', 'E', 'K']
['A', 'B', 'D', 'G', 'H']
['C', 'E', 'F', 'I', 'J', 'K', 'L', 'M']
```

Ans:[C, E, F, I, J, K, L, M]

(e)

```
start point : B
z set(以逗號分隔) : L
unblock ['D', 'E', 'I']
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'M']
['L']
```

Ans:[L]