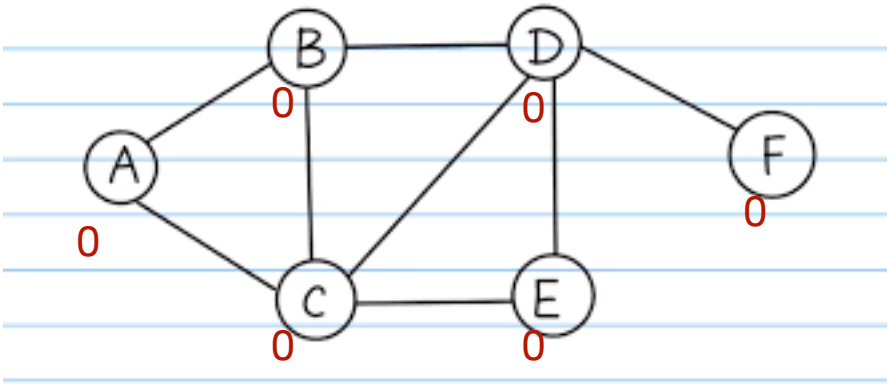
BFS，廣度優先搜尋

概念

也叫做橫向優先搜尋，屬於一種圖形搜尋演算法。概念是先從根節點出發，再存取距離根節點最近距離的子節點，就這樣一層搜完再搜下一層，最終目的是把所有子節點都存取一遍來搜尋結果，不考慮可能位址，徹底搜尋整張地圖。

因為所有節點都必須被儲存，因此BFS的空間複雜度為 O (| V | + | E |)，其中 | V | 是節點的數目，而 | E | 是圖中邊的數目。

流程

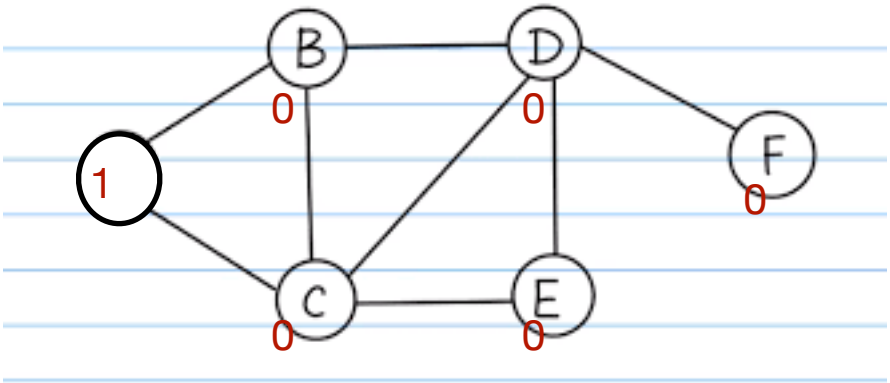


visited:

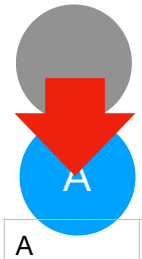
A	B	C	D	E	F
0	0	0	0	0	0

1.先設所有節點為沒搜尋過=>0，取一個根節點A，把A取出來，放到Queue。

把所有節點設為0，當沒有0時代表全部節點都已搜尋過，避免重複搜尋。



Queue:



Print:

A

Visited:

A	B	C	D	E	F
1	0	0	0	0	0

2.把 A 從Queue print出來並從Queue移除，最後把A改為已搜尋=>1。
3.再來是從A的子節開始，有B跟C，重複步驟1跟2。
4.再分別從B跟C的子節重複步驟1跟2。
＃BSF必須是一層搜完再往下一層搜下去，所以Queue的部分要滿足先進先出

學習歷程
12/11
Step1：先建一個能判斷是否走訪過的陣列visited以避免重複存取，只要乘上整張圖的長度就可以剛好把所有節點 設為同一個參數

Step2：創一個queue來暫存目標

Step3：把根節點丟進queue，只要有把節點丟進去就把visited裡的該節點改成True

Step4：再繼續下去之前，這邊要先判斷queue裡有沒有東西，如果沒有就代表圖是空的，就可以結束程式，如果不是空的就代表可以繼續搜尋

Step5：確定queue裡面有東西就要把它弄出來，因為必須符合FIFO，所以這邊用pop(0)，並把他print出來

Step6：接下來就是照順序把所有子節點重複上面的步驟，所以用for迴圈確定圖裡每一個節點都會被存取

Step7：for迴圈裡每一個節點都會被存取一次，所以先加一個判斷式，用visited裡的參數就可以知道哪些已經存取過哪些還沒，如果沒存取過就把它加到到queue裡，並改為已存取。

```
In [1]: from collections import defaultdict


        class Graph:

            def __init__(self):
                self.graph = defaultdict(list)

            def addEdge(self,u,v):
                self.graph[u].append(v)

            def BFS(self, s):
                visited = [0]*(len(self.graph)) #建個list：還沒走訪過的=False
                que = [] #儲存走訪過的元素
                que.append(s) #第一個丟進去
                visited[s] = 1 #改成已走訪=True
                while que:
                    a = que.pop(0) #建立路徑
                    print(a, end=',')
                    for i in self.graph[s]:
                        if visited[i] == 0:
                            que.append(i)
                            visited[i] = 1
```

```
In [4]: g = Graph()
        g.addEdge(0,1)
        g.addEdge(0,2)
        g.addEdge(1,2)
        g.addEdge(2,0)
        g.addEdge(2,3)
        g.addEdge(3,3)
```

```
In [3]: print(g.BFS(2))

        2,0,3,None
```

問題1，不知道為什麼最後的1沒辦法跑出來

12/16

發現a=que.pop(0)這邊如果用a的話就，原本圖的根節點就不會被改變，s還是等於s，這樣的話再找下一層的時候就只會找到原本根的子節點，沒辦法往下一層找，所以我才沒辦法print出1。
所以要用原本的變數s。

```
def BFS(self, s):
    visited = [0] * (len(self.graph))
    que = []
    que.append(s)
    visited[s] = 1
    while que:
        s = que.pop(0)
        print(s, end=',')
        for i in self.graph[s]:
            if visited[i] == 0:
                que.append(i)
                visited[i] = 1
```

In [5]:
```
g = Graph()
g.addEdge(0,1)
g.addEdge(0,2)
g.addEdge(1,2)
g.addEdge(2,0)
g.addEdge(2,3)
g.addEdge(3,3)
```

In [6]:
```
g.BFS(2)
```

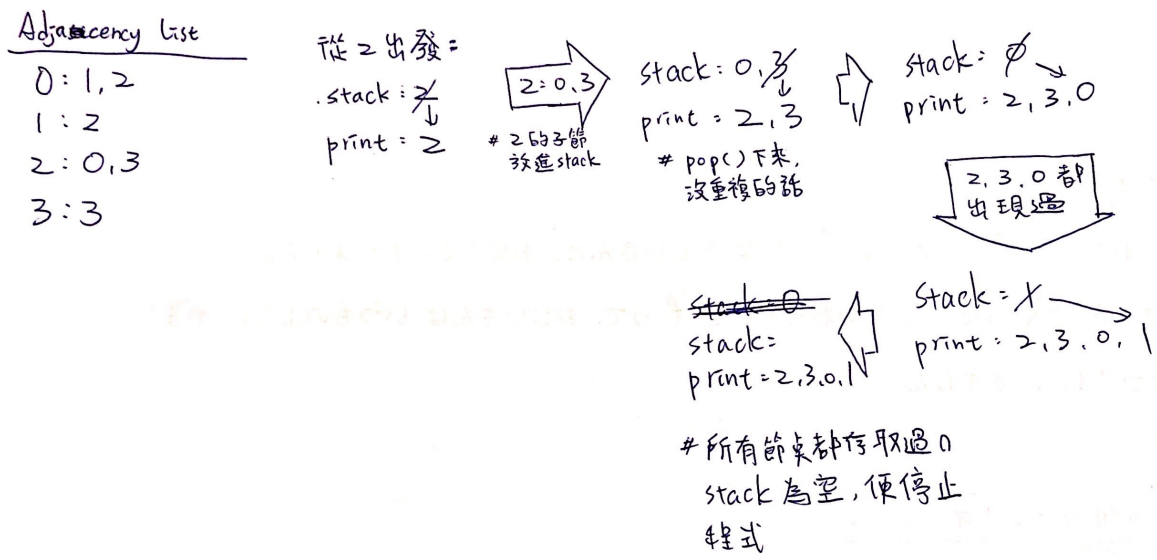
```
2,0,3,1,
```

DFS，深度優先搜尋

概念

也是屬於一種圖形搜尋演算法。概念是從根節點出發，只要他有子節點就存取，直到該節點沒有任何子節點後再回朔到上一層往其他還未存取的節點前進，最終目的是把所有子節點都存取一遍來搜尋結果，不考慮可能位址，徹底搜尋整張地圖。
因為所有節點都必須被儲存，因此DFS的空間複雜度為 O (| V | + | E |)，其中 | V | 是節點的數目，而 | E | 是圖中邊的數目。

流程



學習歷程

12/16

應該跟BFS差不多，所以先把架構架起來，想說最後讓每個節點都跑一次dfs就可以。但這樣stack永遠不為空，就結束不了。而且這樣就跟BFS一樣會先存取同一層所有節點。

```
def DFS(self, s):
    stack = []
    stack.append(s)
    while stack:
        s = stack.pop(-1) # -1：取最後一個
        print(s,end=',')
        for i in self.graph[s]:
            if i not in stack:
                self.DFS(i)
```

In [35]:
```
g = Graph()
g.addEdge(0,1)
g.addEdge(0,2)
g.addEdge(1,2)
g.addEdge(2,0)
g.addEdge(2,3)
g.addEdge(3,3)
```

In [37]:
```
g.DFS(2)
```

```
2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,
,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,
1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0
,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,
0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2
,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,
2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,
```

我用另一個list來存放已存取過的節點，當這個list的數量跟graph節點數量一樣時就停止迴圈，解決重複的問題。

```
def DFS(self, s):
    stack = []
    a=[]
    a.append(s) #直接把根放進去
    while len(a) != len(self.graph):
        for i in self.graph[s]:
            if i not in stack and i not in a:
                stack.append(i)
        s=stack.pop(-1)
        a.append(s)
    return a
```

In [66]:
```
g = Graph()
g.addEdge(0,1)
g.addEdge(0,2)
g.addEdge(1,2)
g.addEdge(2,0)
g.addEdge(2,3)
g.addEdge(3,3)
```

In [67]:
```
g.DFS(2)
```

Out[67]: [2, 3, 0, 1]

DFS & BFS的比較

	BFS	**DFS**
方法	Queue棧儲存節點	Stack佇列儲存節點
存取順序	先進先出	後進先出
應用	一條路徑的邊的數量時 (最短路徑)	一條路徑的權重合時 (最大路徑)
最差時間複雜度	O(v+e)	
最佳時間複雜度	O(1)	
平均時間複雜度	O(v+e)	
額外最差空間複雜度	O(v)	

#以v來表示節點的數量。以e來表示邊的數量。

參考資料

https://zh.wikipedia.org/wiki/广度优先搜索
https://zh.wikipedia.org/wiki/深度优先搜索
https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
https://www.itread01.com/content/1549388200.html
http://simonsays-tw.com/web/DFS-BFS/DepthFirstSearch.html