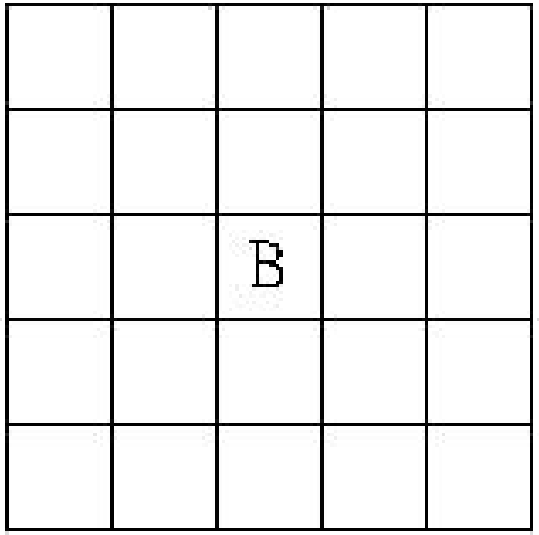
Flood Fill Algorithm

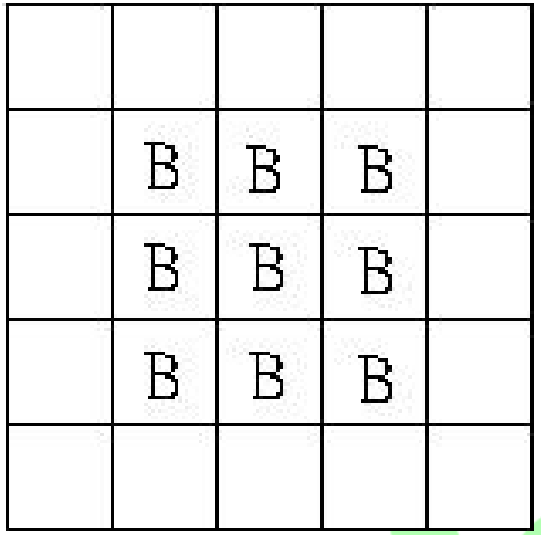
by Chin Huang Lin

Sprout

關於淹水……

- 想像你往一個空的冰塊盒裡面倒水……
- 溢出來的水，會以什麼樣的方式填滿整個格子呢？
  - 也許有些格子還有完整的冰塊，作為障礙物





如何實作？

- Naïve idea: 每次都檢查所有格子，並且讓它們「淹」到鄰居格
- 假設冰塊盒大小是 $n * m$，那每次都要 $O(nm)$ 檢查
- 總共最多需要 $O(n + m)$ 回合 (想想看，為什麼？)
- 複雜度 $O(nm\,(n + m))$
- 問題：花太多時間找出每回合要「溢出去」的格子了！

仔細思考一下……

- 其實每次都只有「最外圍」的格子需要往外淹
- 每個格子都只有一次當「最外圍」的機會！
- 長江後浪推前浪：每回合一定都是先把一些老的最外圍處理掉，然後產生新的最外圍
- 乾脆用成一個 queue 就好啦！

Demo

		1		

		2		
	4	1	5	
		3		

		6		
	7	2	8	
	4	1	5	
		3		

如何實作？

- 怎麼把一個「格子」放進 queue 裡？
  1. 把格子的 (x, y) 座標包成一個 struct/class
  2. 對 x, y 分別開一個 queue
- 有障礙物怎麼辦？
  - 沒怎麼辦，就當作該格早就滿了即可
- 同時往多個格子裡加水怎麼辦？
  - 一開始把所有加水格都丟到 queue 裡
- 邊界好麻煩怎麼辦？
  - 假想一圈外圍（例如 $n * m$ 的冰塊盒就調整成 $(n + 2) * (m + 2)$ ），並且把外圍都標記成障礙物
- 模擬「淹水」的過程很麻煩，幾個方向就要寫幾個很大串的 if，怎麼辦？
  - 先自己想想看囉，我們上課時會討論這個問題 :D
  - 提示：有沒有發現所有的 if 其實都長得很像？

嶄新的複雜度

- 執行的回合數跟 queue 內元素個數有關，但每回合只拓展一格
- 所有會被淹到的格子都恰好會進入 queue 一次
  - queue 內元素個數不超過格子個數，$O(nm)$
- 對於每個元素，拓展都只需要 $O(1)$ 次操作
- 整體複雜度為 $O(nm) * O(1) = O(nm)$

看看練習題吧!

- Zerojudge d537 染色遊戲 (北縣 98 年學科能力競賽)
  - http://zerojudge.tw/ShowProblem?problemid=d537
- 如果直接使用 Naïve 的方法實作：$O(n^3)$
  - 其實當 $N = 100$ 時可以通過！甚至連 $O(n^4)$ 都行
- 如果使用剛剛的方式，擴散變成 $O(n^2)$ 了，可是要怎麼做到
  1. 同時擴散 (剛剛是一格一格擴散)
  2. 計算某顏色的數量呢？

關於同時擴散……

- 觀察一下我們的 queue，不難發現在 queue 裡面「梯數」一定是非嚴格遞增的
  - ex. 起點為 (2,2), (4,5), (10,3)
  - 為了方便起見，假設題目是四方位擴散
  - queue 裡前幾個元素：

    (2,2), (4,5), (10,3),
    (2,3), (2,1), (1,2), (3,2), (4,6), (4,4), (3,5), (5,5), (10,4), (10,2), (9,3), (11,3),
    (2,4), (1,3), (3,3), ……
  - 紅色的是第一梯，藍色的是第二梯，綠色的是第三梯
- 也就是說，我們只需要多記錄每個格子的「梯數」，然後每回合都把 queue 裡面最前面幾個同梯的都一口氣做完就可以了！

關於記錄顏色……

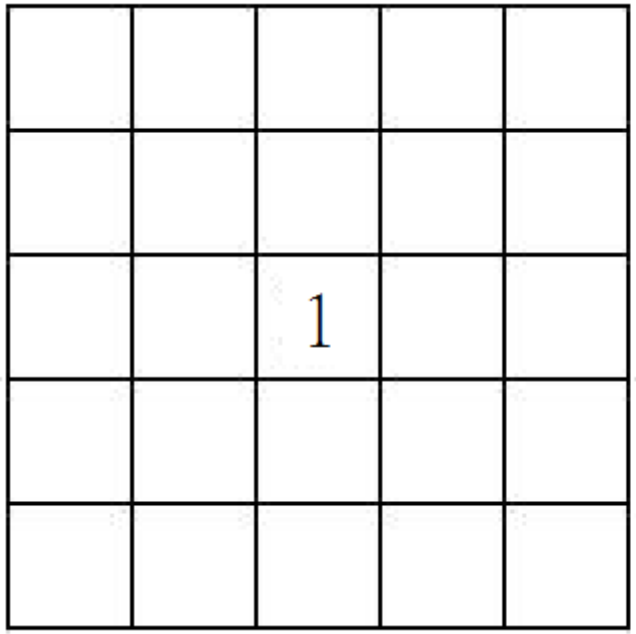
- 如果我們知道如何同時擴散，其實就知道怎麼維護每回合末的顏色個數了
  - 一邊擴散，一邊記錄每種顏色的數量
- 每回合當梯的都擴散完以後直接查看我們的記錄即可
- ✶ 別忘了擴散時要注意到現在在擴散哪種顏色。理論上每個格子都會被擴散三次！

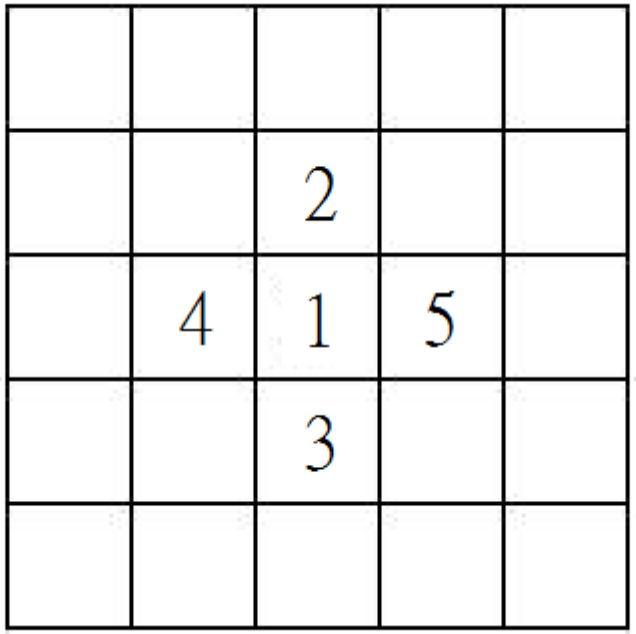
有了剛剛的重大結論……

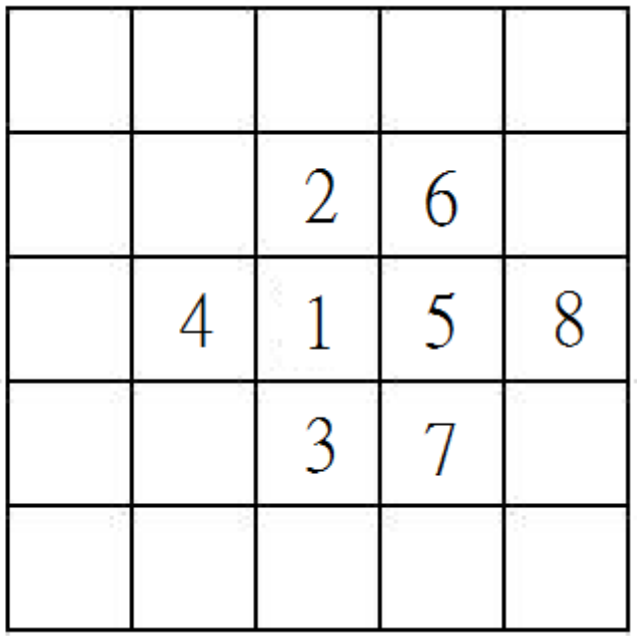
- Zerojudge b224 喵喵抓老鼠 (2008 NPSC 國中組初賽)
  - http://zerojudge.tw/ShowProblem?problemid=b224
- 這次要求的是最短路，跟淹水有什麼關係？
- 讓我們回到剛剛的結論……
  - 觀察一下我們的 queue，不難發現在 queue 裡面「梯數」一定是非嚴格遞增的
- 梯數代表什麼呢？不就是要走幾步才會到嗎？
- 又因為走每一步的成本都是 1，所以「梯數」恰好就是最短路！
  - 想一想，為什麼？可以跟走每一步的成本不見得是 1 的情況比較，例如每個格子都有一個高度值，從某格子走到鄰居格子需要的成本就是高度值的差
- 從起點開始「淹水」，淹到第一隻老鼠就可以停了

回顧一下……

- 剛剛提到的淹水算法中，核心的資料結構是 queue
- 如果把 queue 換成 stack，會發生什麼事情呢？





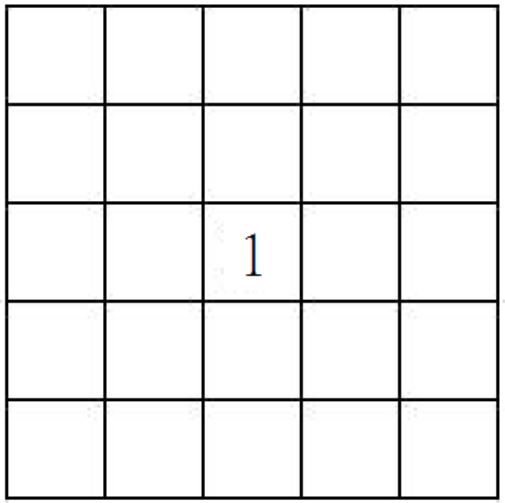


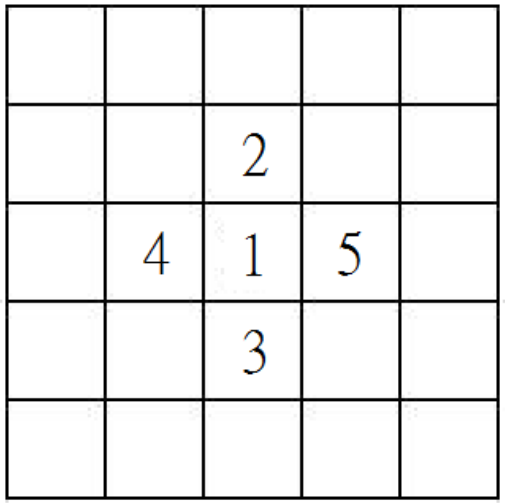
想像一下……

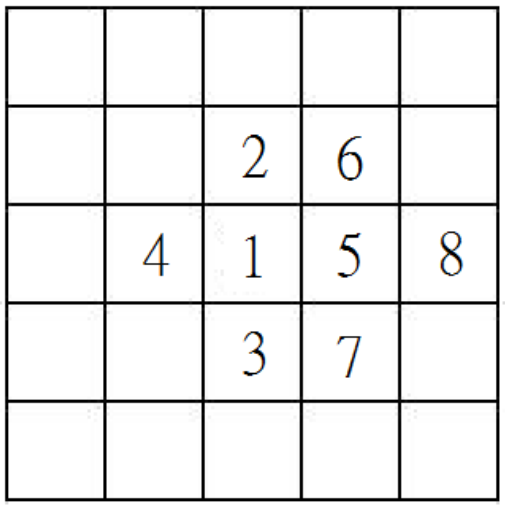
- 如果地圖很大，會看到水先往某個方向一直流，到盡頭才回來流其他方向
- 假如把方向看成學術領域的話：
  - 用 queue 來做，就好像全部的領域都同時涉獵，並且一層一層學深入
  - 用 stack 來做，就好像先一口氣專精某個領域，鑽研完了才鑽研其他領域
  - 用 queue 來做，被稱為廣度優先搜索 (***Breadth-first search***, **BFS**)
  - 用 stack 來做，被稱為深度優先搜索 (***Depth-first search***, **DFS**)
- 實際上，DFS 不會像剛剛提到的那樣實作

DFS 實作

- 觀察一下剛剛的作法……
  - 其實 2,3,4,5 沒有必要同時存在 stack 中！
  - 推入一個，就直接做下去就好了
  - 利用遞迴來實作！







Sample code

```
void dfs( int x, int y, int cur_dist ){
   dist[x][y] = cur_dist;
   if( map[x+1][y] != '#' && dist[x+1][y] > cur_dist + 1 )
      dfs( x+1, y, cur_dist + 1);
   if( map[x][y+1] != '#' ……
  ……
}
```

(x,y) = 當前所在的格子座標

cur_dist = 當前走的步數

dist[x][y] = 走到 (x,y) 的最短路長度

呼叫 (x0,y0,0) 即可求解所有格子的最短路，

其中 (x0,y0) 是貓貓所在

想一想

- 可不可以用 DFS 來做例題一？
- 可不可以用 DFS 來做例題二？
  - 有沒有什麼壞處？
  - 複雜度是多少呢？
- 如果題目是，每個格子都有一個高度值，高度值低於一定程度就會淹水，請問到最後會有幾塊水窪，那麼用 BFS/DFS 都可以做嘛？

BFS vs DFS

- 資料結構
  - queue vs stack
- 單位成本最短路中，最早找到的解就是解
  - 可以 vs 不可以
- 空間消耗
  - 最壞情況下一樣，但通常 DFS 比較省
  - 有些情況下 DFS 使用的空間會少到有量級上的差異
- 好寫程度
  - 需要維護 queue vs 利用遞迴維護 stack
- 例外情形
  - DFS 若用遞迴實作須考慮 stack overflow

Is BFS better？

- 乍看之下似乎 BFS 可以做到的事情比 DFS 還要多得多
- 事實上，在單純的 flood fill 中，DFS 並沒有特長之處；但在未來許多的領域中，DFS 將有許多意想不到的表現
- 敬請期待 :p