Heap/Basic Graph

2025/3/15

上課補充 by 8e7
Credit: fhvirus, enip, lawfung

目錄

- Q&A/複習
- Heap 實作
- Heap 的正確使用時機
- Grid BFS/DFS
- 圖上 DFS/BFS 小提醒
- DFS? BFS?
- 如何還原路徑
- A* 與 Live Coding(?

Sprout

一些小提醒

- 這堂課會有點雜
  - 會留時間給你們問問題
  - 課後歡迎用 Discord 問！
- 感謝 enip, fhvirus ，此份簡報大量沿用前年的簡報
- 想變強請多多翻以前的、其他區的簡報
  - 相關內容太多講不完

Q&A / 複習

複習 - Heap

- heap
- priority queue
- complete binary tree
- heap.push()
- heap.pop()
- heap.top()

複習 - ~~Flood Fill~~

- Flood
- Fill
- Queue
- 梯數 / 回合
- BFS
- Stack
- DFS

Sprout

複習 - Graph

- Vertex
- Edge
- Directed / Undirected
- Multiple Edges
- Loop
- (In / Out) Degree
- Path (Length)
- Cycle
- Connected Graph
- Adjacency List
- Adjacency Matrix
- Connected Components
- Bipartite Graph

Heap 實作

Heap 實作 - STL

- #include <queue>
- priority_queue<T> maxHeap;
- priority_queue<T, vector<T>, greater<T>> minHeap;
- 自定義 struct 的 operator <
- priority_queue<T, vector<T>, bool (*) (T, T)> pq(cmp);

__gnu_pbds::priority_queue

- #include <ext/pb_ds/priority_queue.hpp>
- __gnu_pbds::priority_queue<int> pque;

- priority_queue<int, greater<int> > lque

- h1.join(h2); // O(1) by pairing heap

Heap 實作 - 手刻

- 樹 -> 陣列
  - 根是 1
  - 左小孩 2i
  - 右小孩 2i + 1
  - 家長 i/2
- push/pop/top 如影片
- 建立大小為 N 的 Heap ?
  - push * N -> O(N log N)
  - 數歸？ -> O(N)

Heap 的正確使用時機

Heap 的正確使用時機

- Heap 可以快速做到什麼？

Heap 的正確使用時機

- Heap 可以快速做到什麼？
  - 插入一個數
  - 刪除最大值
  - 查詢最大值

- Heap 不能快速做到什麼？

Heap 的正確使用時機

- Heap 可以快速做到什麼？
  - 插入一個數
  - 刪除最大值
  - 查詢最大值

- Heap 不能快速做到什麼？
  - 刪除一個數（找到一個數）（無法簡單做到）
  - 同時維護最小、最大（binary heap 不行）
  - 找到第 k 大的數字

Heap 的正確使用時機

- Heap 可以快速做到什麼？
  - 插入一個數
  - 刪除最大值
  - 查詢最大值

- Heap 不能快速做到什麼？
  - 刪除一個數（找到一個數）
  - 同時維護最小、最大
  - 找到第 k 大的數字
  - ~~幫你 debug、叫你起床等等...~~

Heap 的正確使用時機

- 什麼時候該用 Heap ？

Heap 的正確使用時機

- 什麼時候該用 Heap ？
  - 非 Heap 不可
  - Heap 比其他方式乾淨/漂亮/

Heap 的正確使用時機

- 什麼時候該用 Heap ？
  - 非 Heap 不可
  - Heap 比其他方式乾淨/漂亮/快
- 什麼時候不該用 Heap ？

Heap 的正確使用時機

- 什麼時候該用 Heap ？
  - 非 Heap 不可
  - Heap 比其他方式乾淨/漂亮/快
- 什麼時候不該用 Heap ？
  - 能用純 array 解決的東西
  - 能用 sort 解決的東西
  - 能用 stack/queue 解決的東西（看情況）

Heap 的正確使用時機

- 給你一個初始就有一些元素的集合，請支援以下操作：
  1. 在集合中加入一個數字
  2. 查詢集合中最大值

怎麼做？

Heap 的正確使用時機

- 給你一個初始就有一些元素的集合，請支援以下操作：
  1. 在集合中加入一個數字
  2. 查詢集合中最大值

怎麼做？

使用 heap 加入元素，並回傳最大值！

Heap 的正確使用時機

- 給你一個初始就有一些元素的集合，請支援以下操作：
  1. 在集合中加入一個數字
  2. 查詢集合中最大值

怎麼做？

使用 heap 加入元素，並回傳最大值！

等等，是不是哪裡怪怪的？

維護當前最大值即可！

Heap 的正確使用時機

- 給你一個初始就有一些元素的集合，請支援以下操作：
  1. 在集合中刪除最大值
  2. 查詢集合中最大值

怎麼做？

Heap 的正確使用時機

- 給你一個初始就有一些元素的集合，請支援以下操作：
  1. 在集合中刪除最大值
  2. 查詢集合中最大值

怎麼做？

使用 heap 刪除最大值，並回傳最大值！

Heap 的正確使用時機

- 給你一個初始就有一些元素的集合，請支援以下操作：
  1. 在集合中刪除最大值
  2. 查詢集合中最大值

怎麼做？

使用 heap 刪除最大值，並回傳最大值！

你聽過 sort 嗎？

Heap 的正確使用時機

- 給你一個初始就有一些元素的集合，請支援以下操作：
  1. 在集合中刪除一個數
  2. 查詢集合中最大值

怎麼做？

Heap 的正確使用時機

- 給你一個初始就有一些元素的集合，請支援以下操作：
  1. 在集合中刪除一個數
  2. 查詢集合中最大值

怎麼做？

heap 甚至無法直接做。那維護一個 bool 陣列，deleted[i] 表示位置 i 上的東西是否已被刪掉，然後 heap 裡的每個節點多存一個值代表位置，在查詢前不停 pop 掉 heap 的最大值直到那一個最大值未被刪掉，要想辦法找到每個元素的位置那要開個 unordered_map<int, vector<int>> .....

Heap 的正確使用時機

- 給你一個初始就有一些元素的集合，請支援以下操作：
  1. 在集合中刪除一個數
  2. 查詢集合中最大值

怎麼做？

如果可離線？倒過來做加入！

如果不可離線？剛剛做的那些事情在 sort 好的 array 上面也能做！

Heap 的正確使用時機

- 不要為了用資料結構而用資料結構！
- 能坐著就別站著，能躺著就別坐著。
- 資料結構是手段，不是唯一解法，更非目的
- 不要讓工具限制住了你的想像力。

Heap 的正確使用時機

- 給你一個陣列，對於每一個陣列中所有長度為 k 的 subarray，求出 k 個數字中的最大值。
- Note : subarray 是一段連續的數字

怎麼做？

- 把前 k 個數字塞進去 heap
- 每次刪一個、加一個
- 想辦法處理刪除
- O(NlogN)

Heap 的正確使用時機

- 給你一個陣列，對於每一個陣列中所有長度為 k 的 subarray，求出 k 個數字中的最大值。
- Note : subarray 是一段連續的數字

怎麼做？

- 記得 deque 嗎？
- 把還有可能是最大值的東西留在 deque 裡
- 每次看 deque 的 front 是否還合法
- O(N)

Heap 的正確使用時機

- For k = 4

5	2	4	1	3	6	7

- Deque

5						

Heap 的正確使用時機

- For k = 4

5	2	4	1	3	6	7

- Deque

5	2					

Heap 的正確使用時機

- For k = 4

5	2	4	1	3	6	7

- Deque

5	~~2~~					

Heap 的正確使用時機

- For k = 4

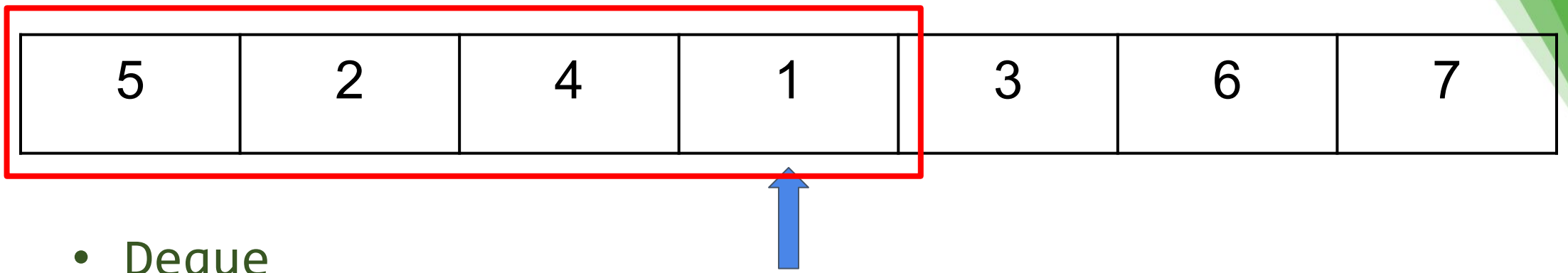
5	2	4	1	3	6	7

- Deque

5	4					

Heap 的正確使用時機

- For k = 4



- Deque

Heap 的正確使用時機

- For k = 4

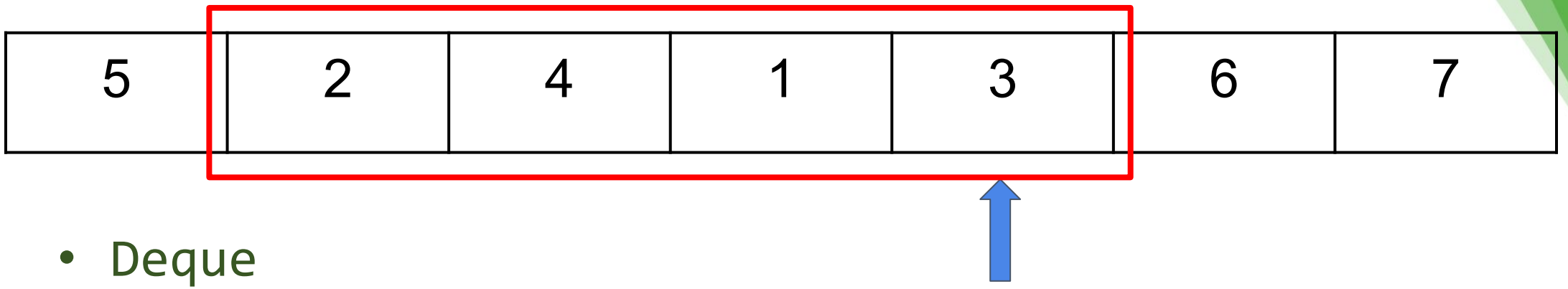
5	2	4	1	3	6	7

- Deque

5	4	1				

Heap 的正確使用時機

- For k = 4



- Deque

Heap 的正確使用時機

- For k = 4

5	2	4	1	3	6	7

- Deque

~~5~~	4	~~3~~				

Heap 的正確使用時機

- For k = 4

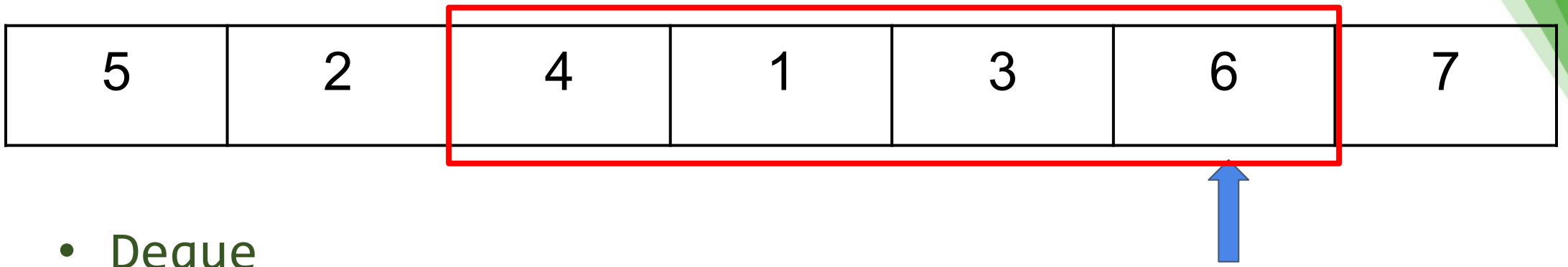
5	2	4	1	3	6	7

- Deque

~~5~~	~~4~~	~~3~~				

Heap 的正確使用時機

- For k = 4



- Deque

Heap 的正確使用時機

- For k = 4

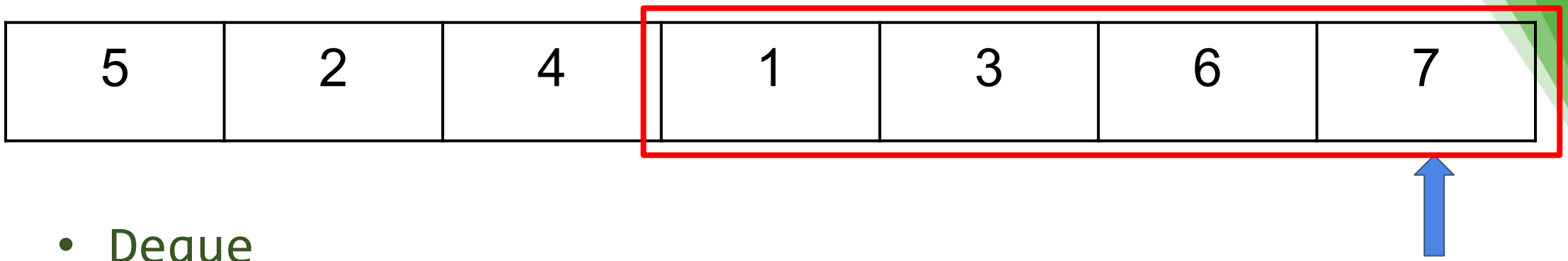
5	2	4	1	3	6	7

- Deque

~~5~~	~~6~~					

Heap 的正確使用時機

- For k = 4



- Deque

Heap 的正確使用時機

- 1161 - 4.虛擬番茄online | TIOJ
- 花一點時間看題目

Heap 的正確使用時機

- 1161 - 4.虛擬番茄online | TIOJ
- 枚舉一個維度，對於另一維度保留前 k 小的值
- 如何知道第 k 小的值是多少？
- 大小為 k 的 heap
- 對，這題真的要用 heap 啦
- 為什麼用 heap?
  - 動態更新
  - 有加入、有刪除
  - 只有刪除最大值，沒有其他東西

存圖

存圖

- 相鄰矩陣
  - 空間複雜度:O(V^2)
  - 查詢兩個點之間是否有邊:O(1)
  - 遍歷一個點周圍的邊:O(V)
  - 增加一條邊:O(1)
  - 刪除一條邊:O(1)
- 相鄰串列
  - 空間複雜度:O(V+E)
  - 查詢兩個點之間是否有邊:O(deg)
  - 遍歷一個點周圍的邊:O(deg)
  - 增加一條邊:O(1)
  - 刪除一條邊:O(deg)

存圖

- 相鄰矩陣
  - 宣告方便
  - 思考方式直覺
  - 詢問是否存在邊很快
  - 刪邊很快
  - 完全圖時適合使用
  - 其他地方不如相鄰串列

存圖

- 相鄰串列
  - 使用 vector 實作
  - 總空間 O(V + E)
    - 在完全圖上跟相鄰矩陣一樣
  - 稀疏圖上使用
  - 快速查詢兩點間有沒有邊？ set

存圖

- 小思考
  - 在沒有 vector、無法動態開陣列的情況下，該怎麼實作相鄰串列呢？

存圖

- 小思考
  - 在沒有 vector、無法動態開陣列的情況下，該怎麼實作相鄰串列呢？
  - 自由發揮，方法應該很多種
  - 有一天 C++ 編譯器壞掉，只能寫 C
    - ~~大學生活~~
  - 對於每個點，紀錄最後一條有這個點的邊的 index
  - 對於每個邊，紀錄兩端點上次出現的邊的 index

DFS ? BFS ?

Sprout

DFS ? BFS ?

- 在基礎的 flood fill 中，DFS 跟 BFS 都可以用來得知有幾個連通塊、每個連通塊的大小等等。
- 但在需要得知層級(距離原點的最短距離)時，BFS 有絕對優勢。若 A 點的 BFS 順序小於 B 點，則 A 距離原點的距離必定不大於 B。
- DFS 看起來沒什麼優勢？
- 為什麼不要都寫 BFS 就好？

DFS ? BFS ?

- BFS 的優勢
  - 最短距離
  - 最少步數
  - 最....
- DFS
  - 找到一組解
  - 不一定是最近
  - 要求最...的時候，使用 BFS 較佳

DFS ? BFS ?

- DFS 的優勢
  - 節省空間
    - 數獨、魔方陣等等
    - 盤面複雜不能用一個數字代表
  - 思考直覺
    - 遞迴式思考
  - 實作簡易
    - 就是遞迴啦
  - 只要找一組解，而不需要最佳解時
  - 只需要確定 Yes / No

DFS ? BFS ?

- DFS
- 魔方陣

6	1	8
7	5	3
2	9	

DFS ? BFS ?

- DFS
- 魔方陣

6	1	8
7	5	3
2	9	1

DFS ? BFS ?

- DFS
- 魔方陣

6	1	8
7	5	3
2	9	~~4~~

DFS ? BFS ?

- DFS
- 魔方陣

6	1	8
7	5	3
2	9	2

DFS ? BFS ?

- DFS
- 魔方陣

6	1	8
7	5	3
2	9	~~2~~

DFS ? BFS ?

- DFS
- 魔方陣

6	1	8
7	5	3
2	9	3

DFS ? BFS ?

- DFS
- 魔方陣

6	1	8
7	5	3
2	9	~~3~~

DFS ? BFS ?

- DFS
- 魔方陣

6	1	8
7	5	3
2	9	4

DFS ? BFS ?

- DFS
- 魔方陣

6	1	8
7	5	3
2	9	4

找到答案！

總共只用了九格空間～

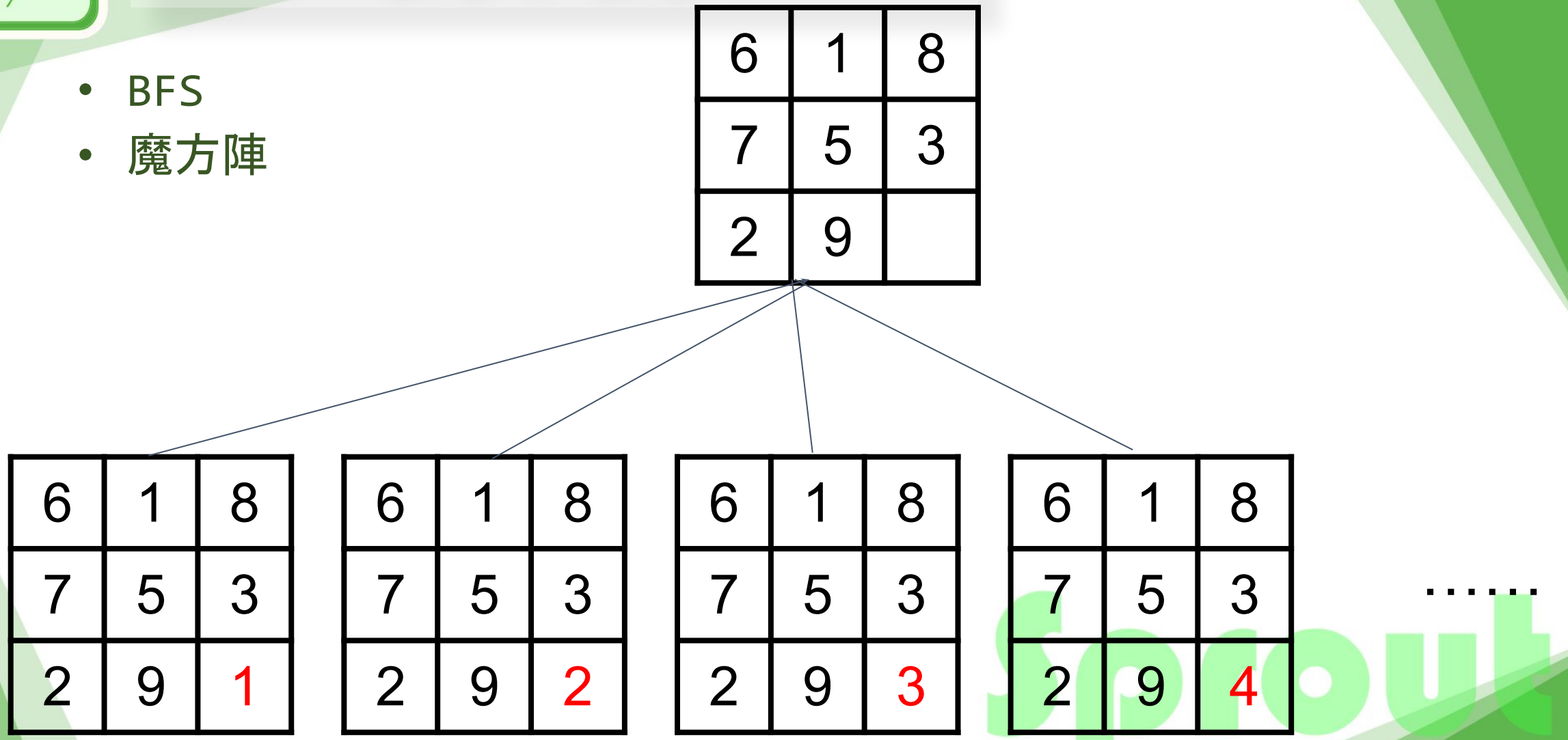
DFS ? BFS ?

- BFS
- 魔方陣

6	1	8
7	5	3
2	9	

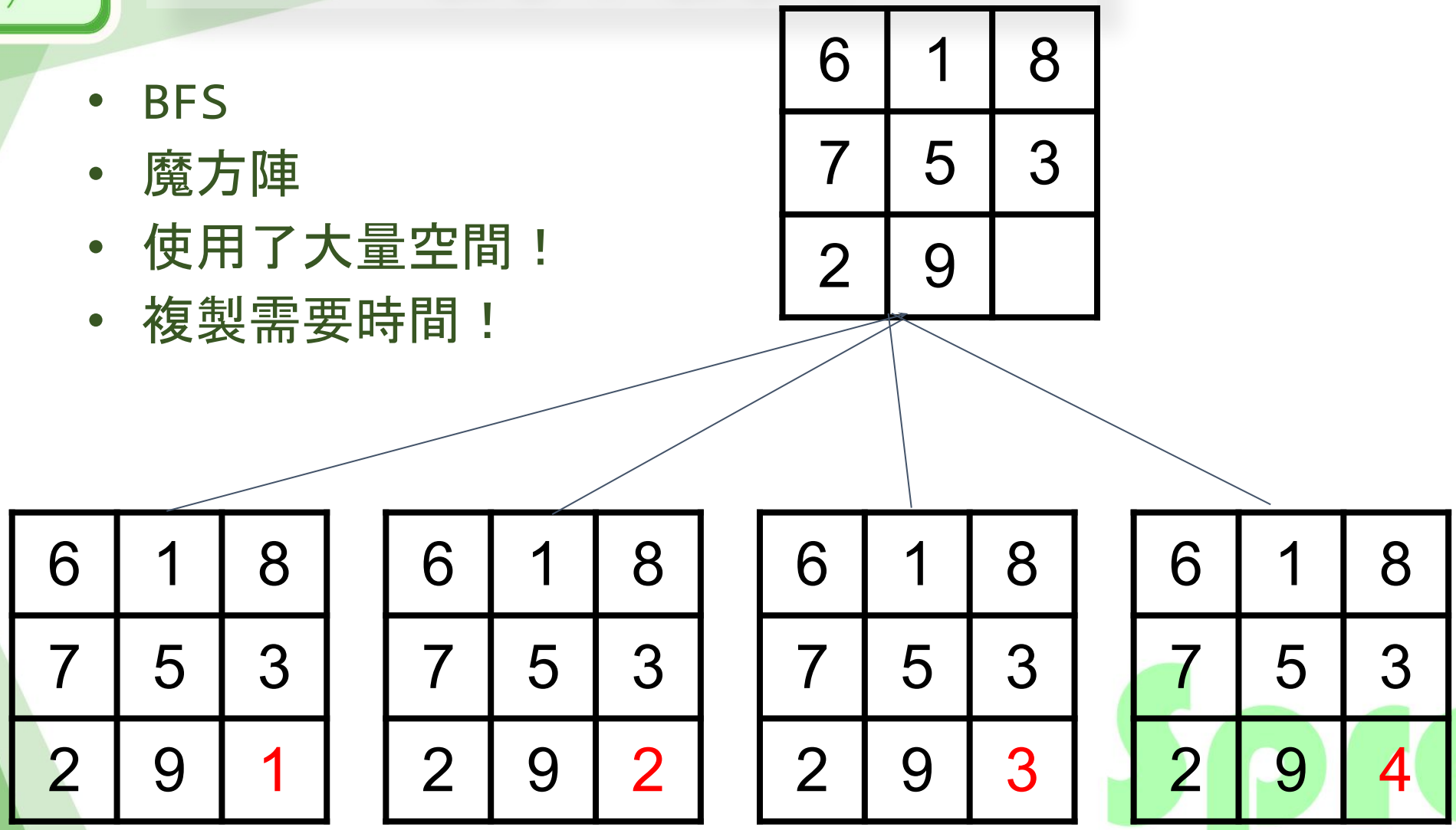
DFS ? BFS ?

- BFS
- 魔方陣



DFS ? BFS ?

- BFS
- 魔方陣
- 使用了大量空間！
- 複製需要時間！



圖上 DFS 小提醒

圖上 DFS 小提醒

- DFS、BFS 的時間複雜度是多少？
- 給你一張點數 < 5000 的圖，請支援以下操作：

  1. 刪邊、加邊
  2. 查詢有幾個連通塊

  （查詢操作小於 5000 個）

圖上 DFS 小提醒

- DFS、BFS 的時間複雜度是多少？
- 考量以下作法：
  - 每次詢問就 DFS
  - DFS 一次是 5000
  - 5000 * 5000 = 2.5 * 10^7
  - 會過嗎？

圖上 DFS 小提醒

- DFS、BFS 的時間複雜度是多少？
- 考量以下作法：
  - 每次詢問就 DFS
  - DFS 一次是 ~~5000~~ O(N + M) / O(N^2)!
  - M 是 5000^2
  - TLE！

圖上 DFS 小提醒

- DFS、BFS 的時間複雜度是多少？
- O(N + M)！
- 別忘記那個 M
- DFS 的過程中，對於每一個點，要檢查他所有的邊
  - 所以是 O(N + M)

- 常常因為 N = 1e5, M < 2e5 之類的限制，忘記 M 的存在

圖上 BFS 小提醒

圖上 BFS 小提醒

- 這段 code，出了什麼問題？

```
void bfs(int s){
    for(int i = 0; i < N; i++) vis[i] = 0;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
        int node = q.front();
        q.pop();
        vis[node] = 1;
        for(int i = 0; i < (int)Adj[node].size(); i++){
            if(!vis[Adj[node][i]]){
                q.push(Adj[node][i]);
            }
        }
    }
}
```

圖上 BFS 小提醒

- 這段 code，出了什麼問題？
- BFS 時，是如何保證時間複雜度的？
- 一個點會進 queue 幾次？

```
void bfs(int s){
    for(int i = 0; i < N; i++) vis[i] = 0;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
        int node = q.front();
        q.pop();
        vis[node] = 1;
        for(int i = 0; i < (int)Adj[node].size(); i++){
            if(!vis[Adj[node][i]]){
                q.push(Adj[node][i]);
            }
        }
    }
}
```

圖上 BFS 小提醒

- 正確寫法
- 差在哪裡？

```
void bfs(int s){
    for(int i = 0; i < N; i++) vis[i] = 0;
    queue<int> q;
    q.push(s);
    vis[s] = 1;
    while(!q.empty()){
        int node = q.front();
        q.pop();
        for(int i = 0; i < (int)Adj[node].size(); i++){
            if(!vis[Adj[node][i]]){
                q.push(Adj[node][i]);
                vis[Adj[node][i]] = 1;
            }
        }
    }
}
```

圖上 BFS 小提醒

- 正確寫法
- 差在哪裡？
- 推進 queue 時就更改 vis
- 難以找到的 bug

```
void bfs(int s){
    for(int i = 0; i < N; i++) vis[i] = 0;
    queue<int> q;
    q.push(s);
    vis[s] = 1;
    while(!q.empty()){
        int node = q.front();
        q.pop();
        for(int i = 0; i < (int)Adj[node].size(); i++){
            if(!vis[Adj[node][i]]){
                q.push(Adj[node][i]);
                vis[Adj[node][i]] = 1;
            }
        }
    }
}
```

圖論應用 : 還原路徑

例題: CSES Labyrinth

- 給你一個 n*m 表格形狀的地圖，有些格子是空地，其他格子是障礙物
- 請找出一條從 A 到 B 點的最短路徑，並且輸出移動的方式(用 UDLR 字串代表每一步要往上/下/左/右

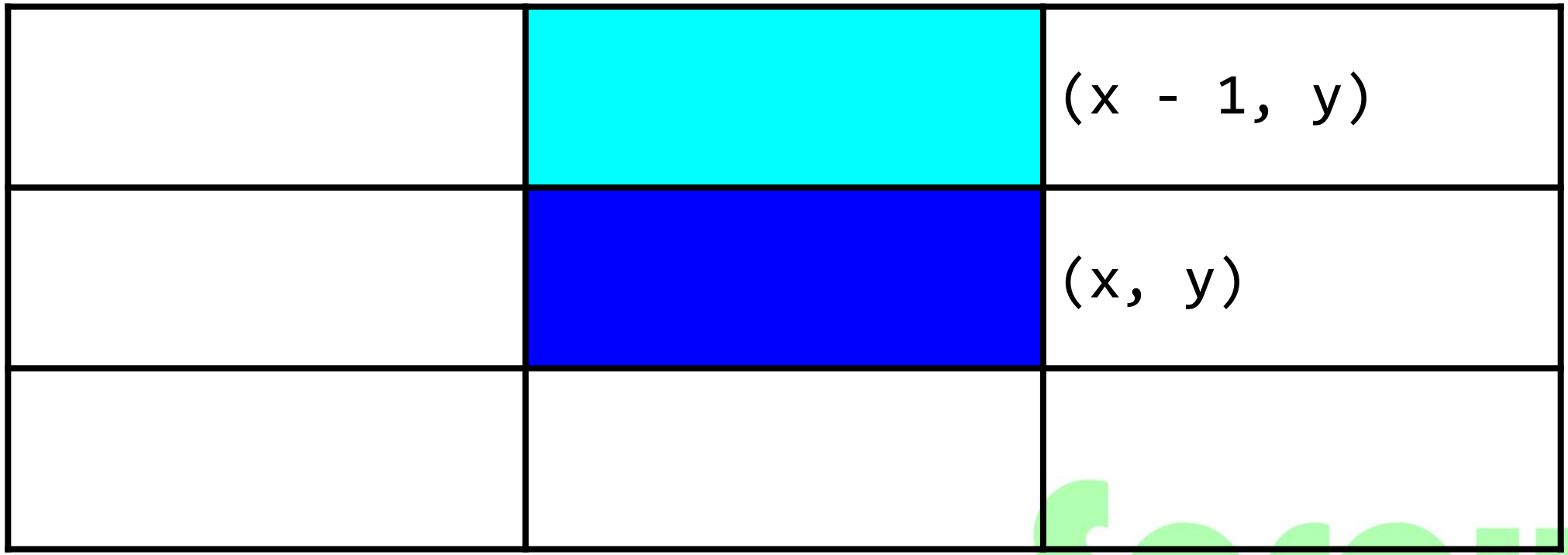
利用陣列列舉方向

```
int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, -1, 0, 1};

for(int i = 0 ; i < 4; i++)
   if(check(x + dx[i], y + dy[i]))
      queue.push(x + dx[i], y + dy[i]);
```

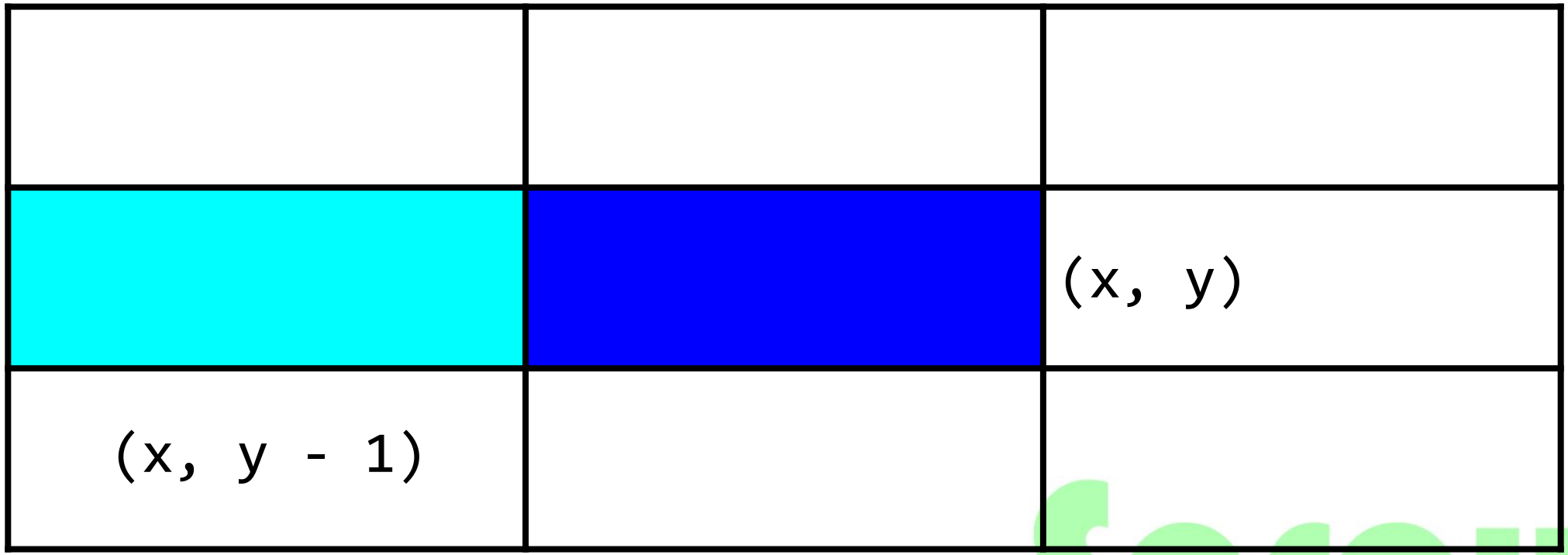
利用陣列列舉方向

```
i = 0 : dx[i] = -1, dy[i] = 0
```



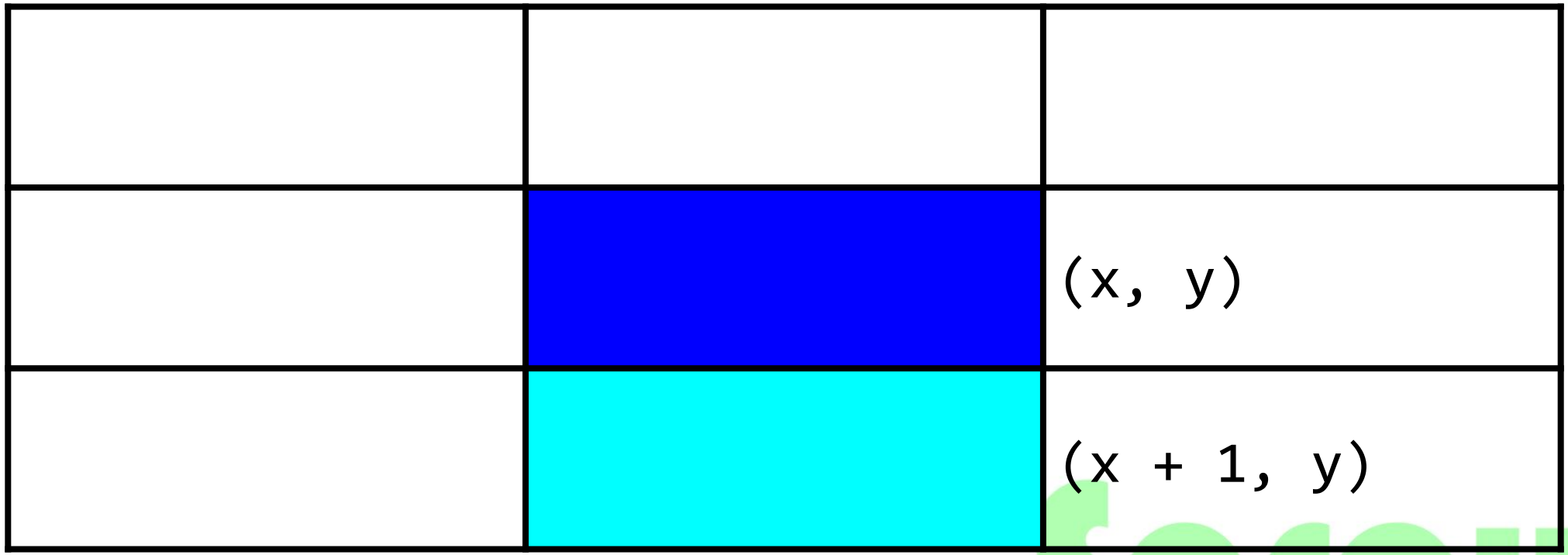
利用陣列列舉方向

```
i = 1 : dx[i] = 0, dy[i] = -1
```



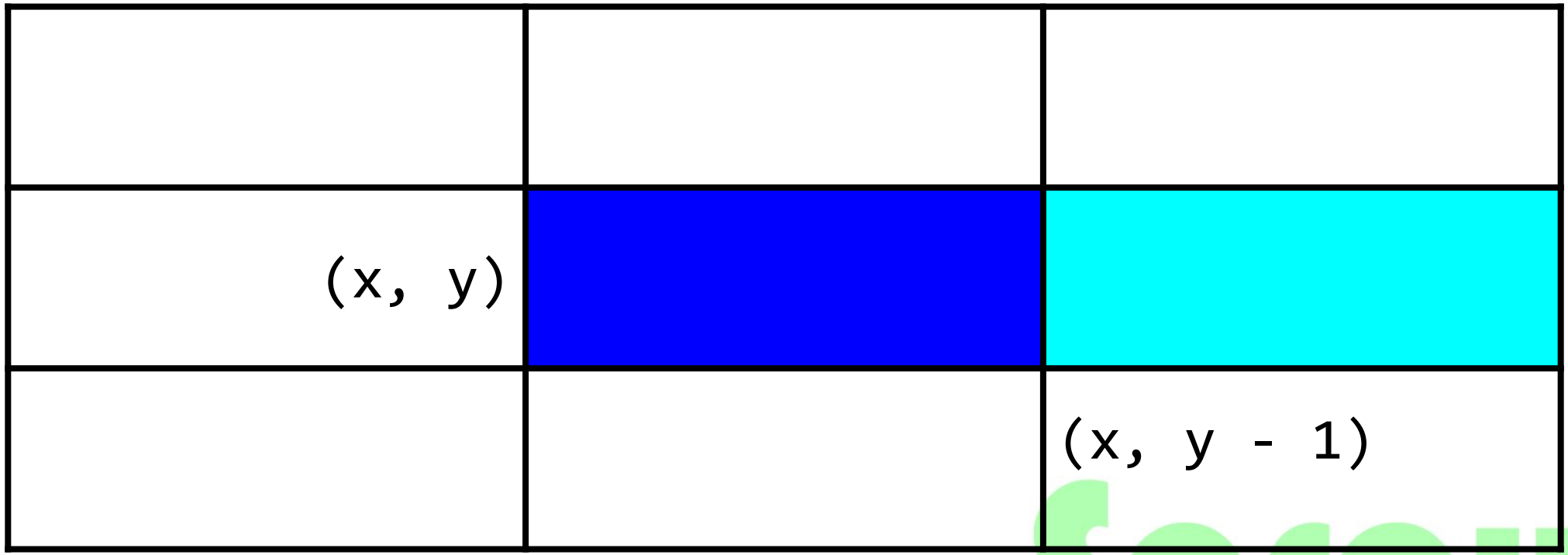
利用陣列列舉方向

`i = 2 : dx[i] = 1, dy[i] = 0`



利用陣列列舉方向

```
i = 3 : dx[i] = 0, dy[i] = 1
```



利用陣列列舉方向

- 把每個方向對應的 x 變化量與 y 變化量存在陣列中
- 在枚舉陣列 index 時，就等於枚舉完所有方向了
- 不只四方向適用
  - 八方向也行
  - 三維的六方向、26 方向也行
    - 希望不需要寫到 26 方向
- 要進入圖論環節囉！

怎麼還原路徑？

- 找到最短距離就直接使用 BFS 就好了
- BFS 擁有的性質：第一次走到一個點的時候，該點紀錄的距離就是最短距離
- 也就是說如果是點 x「發現」點 y 的話，起點到點 y 的最短路徑就是：起點到點 x 的最短路 + x 走到 y
- 這個時候就可以把點 y 的「來源」設定成 x

怎麼還原路徑？

- 找到最短距離就直接使用 BFS 就好了
- 讓我們再看一次 BFS 的程式
- 試著紀錄「每個點的最短路徑的來源」

node -> Adj[node][i]

```
void bfs(int s){
    for(int i = 0; i < N; i++) vis[i] = 0;
    queue<int> q;
    q.push(s);
    vis[s] = 1;
    while(!q.empty()){
        int node = q.front();
        q.pop();
        for(int i = 0; i < (int)Adj[node].size(); i++){
            if(!vis[Adj[node][i]]){
                q.push(Adj[node][i]);
                vis[Adj[node][i]] = 1;
            }
        }
    }
}
```

怎麼還原路徑？

- 找到最短距離就直接使用 BFS 就好了
- 讓我們再看一次 BFS 的程式
- 試著紀錄「每個點的最短路徑的來源」
- 最後從終點開始，沿著來源走就會到起點，再把這個路徑反轉就能得到答案！

Live Coding 時間

結合 heap 和 BFS - A* 演算法

BFS 的缺點

- 常常，BFS 都需要看過幾乎整張圖才能找到答案，搜尋的範圍也不會因為終點不同而有所改變。
- 在現實生活的路徑搜尋問題中（像是 Google Maps），對應的圖非常龐大，因此要盡可能減少不必要的搜尋。

往最可能是答案的方向走

- A* (唸作 A star) 演算法是一個啟發式 (Heuristic) 搜尋演算法
- 他會給每個點一個分數，並且按照分數高低決定搜尋的順序。

往最可能是答案的方向走

- 這個分數是:經過該點的路徑到終點時至少要多長
- 可以分解成:到該點目前的最短路徑（f）

  +從該點到終點的路徑長的估計值（g）
- g 必須要不大於實際的距離:像是曼哈頓距離

怎麼實作呢？

- 紀錄每一個點從起點來的距離（f）
- 使用一個 heap 來儲存（f+g，點座標），每次取得首項最小值的點去更新

延伸問題：有沒有辦法不使用 Heap 呢？

Live Coding

謝謝大家！

Sprout