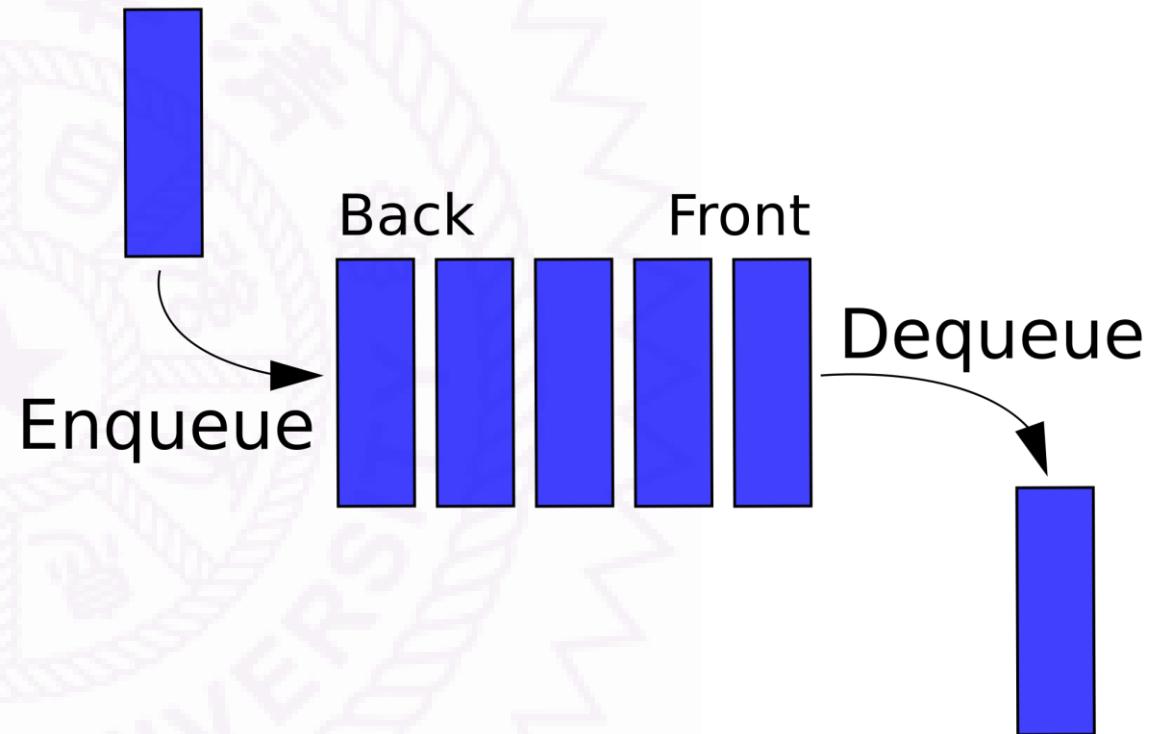


廣度優先搜尋

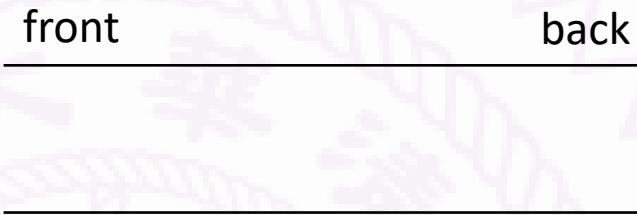
日月卦長

Queue

- push/ emplace(X) 加入資料
- front() 看最前面的資料
- pop() 刪掉最前面的資料
- First in first out



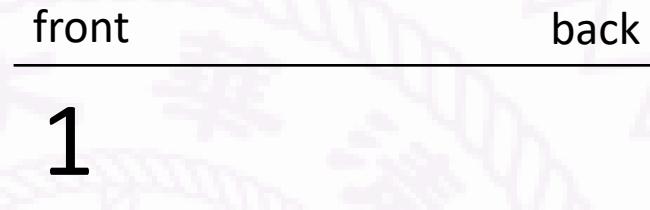
std::queue



```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
int main() {
    queue<int> Q;
    Q.emplace(1);
    Q.emplace(2);
    Q.emplace(3);
    Q.pop();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

```
#include <deque>
#include <iostream>
#include <vector>
using namespace std;
int main() {
    deque<int> Q;
    Q.emplace_back(1);
    Q.emplace_back(2);
    Q.emplace_back(3);
    Q.pop_front();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

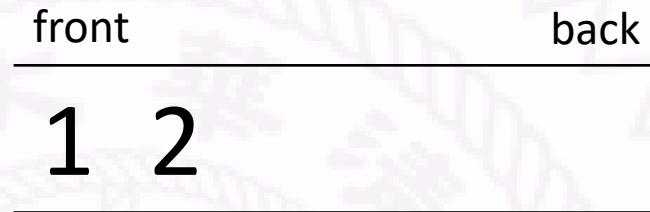
std::queue



```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
int main() {
    queue<int> Q;
    Q.emplace(1);
    Q.emplace(2);
    Q.emplace(3);
    Q.pop();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

```
#include <deque>
#include <iostream>
#include <vector>
using namespace std;
int main() {
    deque<int> Q;
    Q.emplace_back(1);
    Q.emplace_back(2);
    Q.emplace_back(3);
    Q.pop_front();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

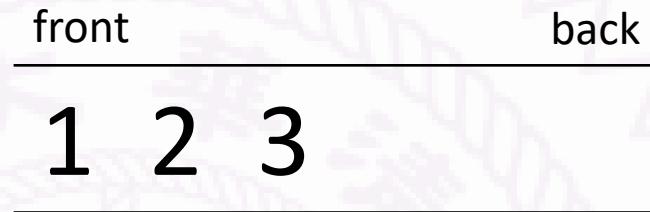
std::queue



```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
int main() {
    queue<int> Q;
    Q.emplace(1);
    Q.emplace(2); [Red box]
    Q.emplace(3);
    Q.pop();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

```
#include <deque>
#include <iostream>
#include <vector>
using namespace std;
int main() {
    deque<int> Q;
    Q.emplace_back(1);
    Q.emplace_back(2); [Red box]
    Q.emplace_back(3);
    Q.pop_front();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

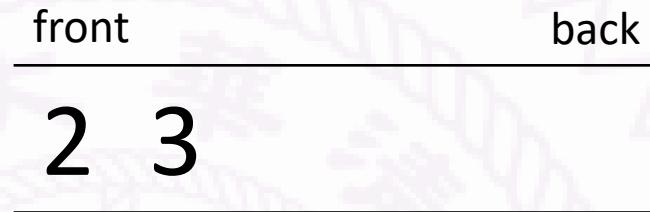
std::queue



```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
int main() {
    queue<int> Q;
    Q.emplace(1);
    Q.emplace(2);
    Q.emplace(3);
    Q.pop();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

```
#include <deque>
#include <iostream>
#include <vector>
using namespace std;
int main() {
    deque<int> Q;
    Q.emplace_back(1);
    Q.emplace_back(2);
    Q.emplace_back(3);
    Q.pop_front();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

std::queue



```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
int main() {
    queue<int> Q;
    Q.emplace(1);
    Q.emplace(2);
    Q.emplace(3);
    Q.pop();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

```
#include <deque>
#include <iostream>
#include <vector>
using namespace std;
int main() {
    deque<int> Q;
    Q.emplace_back(1);
    Q.emplace_back(2);
    Q.emplace_back(3);
    Q.pop_front();
    cout << Q.front() << ' ' << Q.back();
    return 0;
}
```

是用 deque 做出來的

```
2 #include <vector>
3 #include <queue> <typename _Tp, typename _Sequence = deque <_Tp>>
4 using namespace std;
5
6 int main() { A standard container giving FIFO behavior.
7     queue<int,>
```

元素存取

queue	
尾巴	back()
頭部	front()

數量資訊

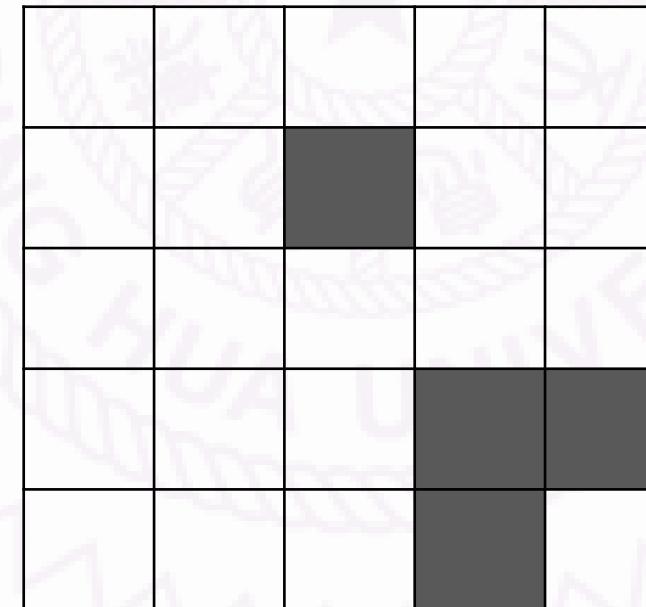
queue	
裡面是不是空的	empty()
裡面有多少東西	size()

洪水算法

Flood fill algorithm

倒水問題

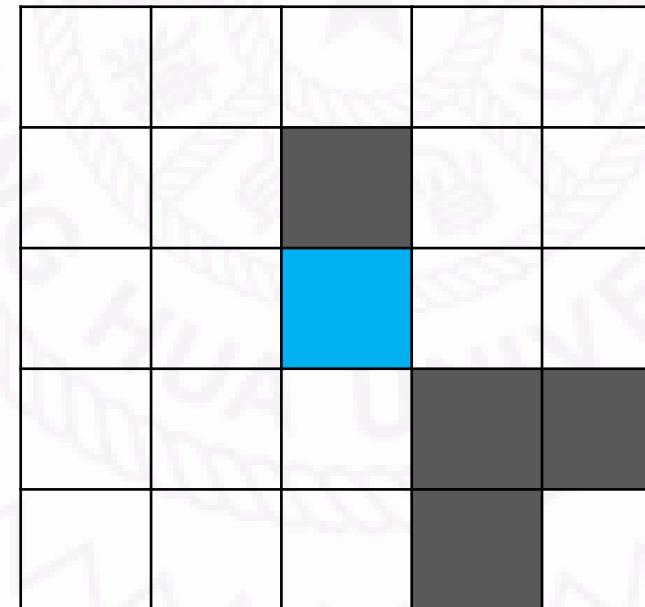
- 在一個 $N \times M$ 的格子圖中，選一個點倒水
- 溢出來水會往某些方向流(看題目規定水怎麼流)
- 有些時候會有障礙物無法淹沒



倒水問題

- 在一個 $N \times M$ 的格子圖中，選一個點倒水
- 溢出來水會往某些方向流(看題目規定水怎麼流)
- 有些時候會有障礙物無法淹沒

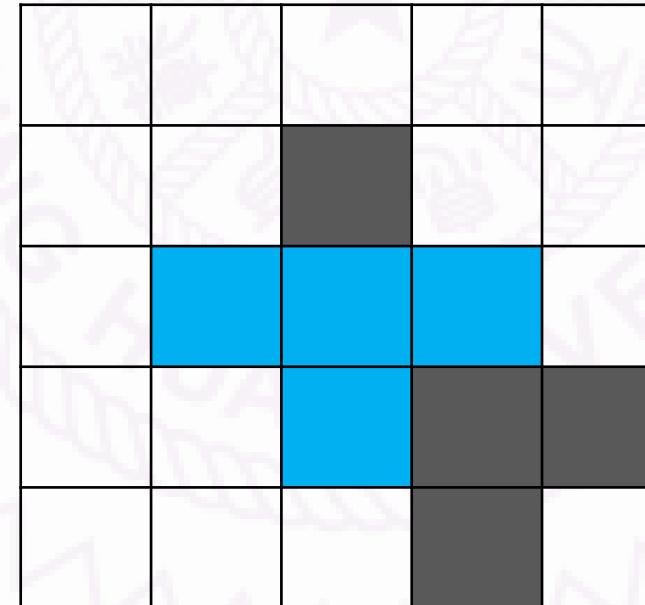
假設每個格子
的水只會往上
下左右淹



倒水問題

- 在一個 $N \times M$ 的格子圖中，選一個點倒水
- 溢出來水會往某些方向流(看題目規定水怎麼流)
- 有些時候會有障礙物無法淹沒

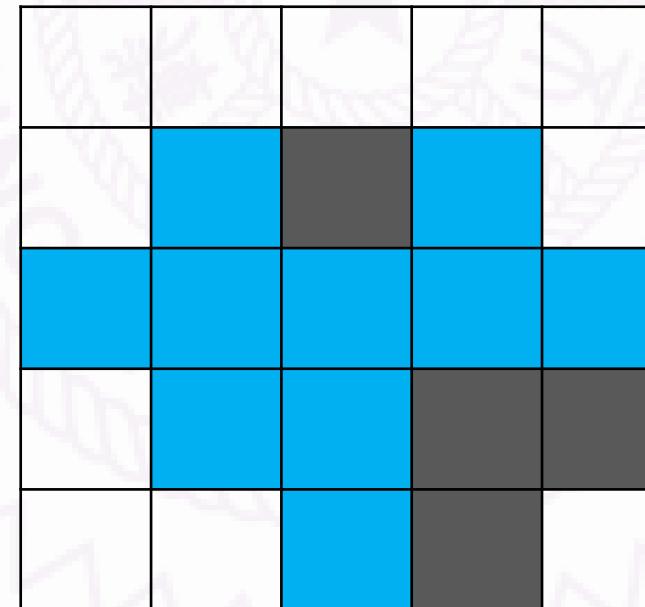
假設每個格子
的水只會往上
下左右淹



倒水問題

- 在一個 $N \times M$ 的格子圖中，選一個點倒水
- 溢出來水會往某些方向流(看題目規定水怎麼流)
- 有些時候會有障礙物無法淹沒

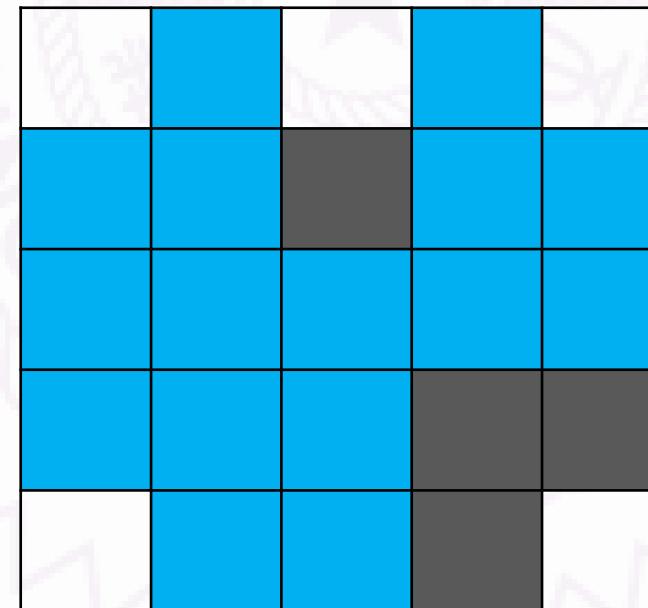
假設每個格子
的水只會往上
下左右淹



倒水問題

- 在一個 $N \times M$ 的格子圖中，選一個點倒水
 - 溢出來水會往某些方向流(看題目規定水怎麼流)
 - 有些時候會有障礙物無法淹沒

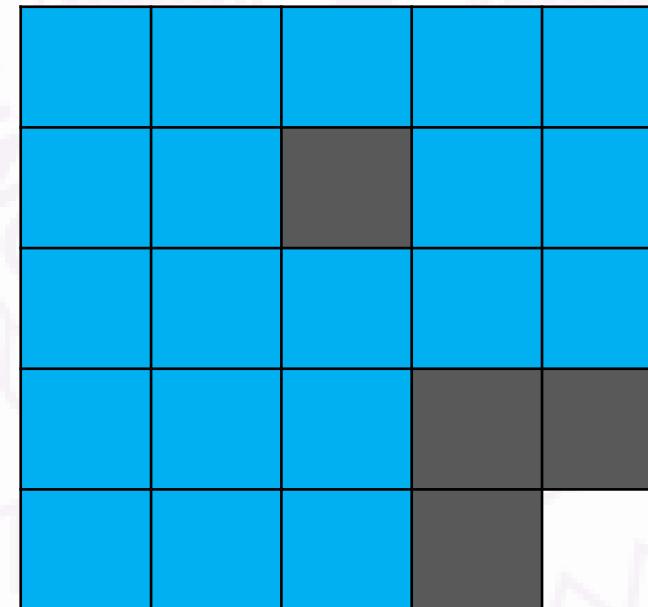
假設每個格子的水只會往上
下左右淹



倒水問題

- 在一個 $N \times M$ 的格子圖中，選一個點倒水
- 溢出來水會往某些方向流(看題目規定水怎麼流)
- 有些時候會有障礙物無法淹沒

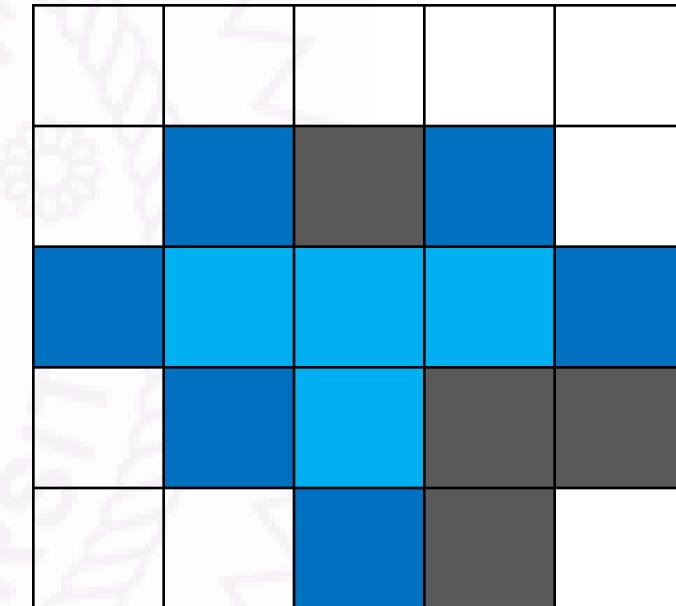
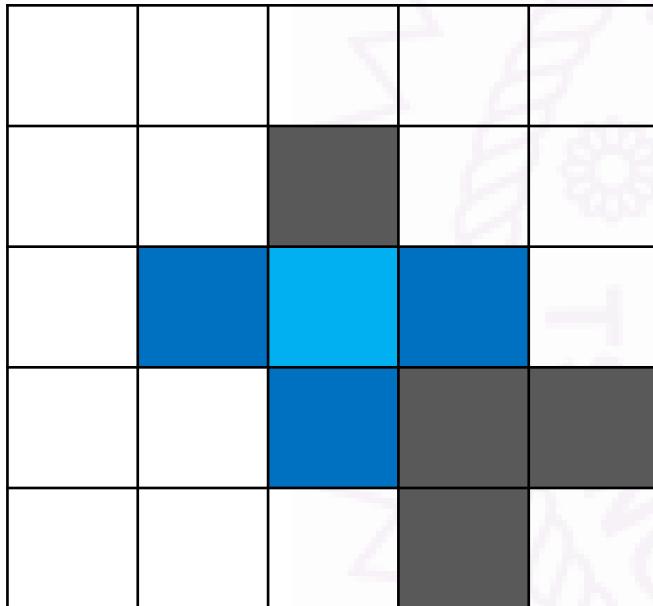
假設每個格子
的水只會往上
下左右淹



暴力法 $O(N^2M^2)$

- 設起始點的level為0
- $L = 1$
- While True
 - 掃描所有的格子 (x,y) :
 - 如果 (x,y) 非障礙物 且 沒設置level 且 上下左右的格子的 level 等於 $L-1$
 - 設 (x,y) 的level為 L
 - 如果都沒有格子的level為 $L-1$
 - Break
 - $L = L + 1$

能淹出水的格子



用 queue 記錄所有邊界格子

Demo

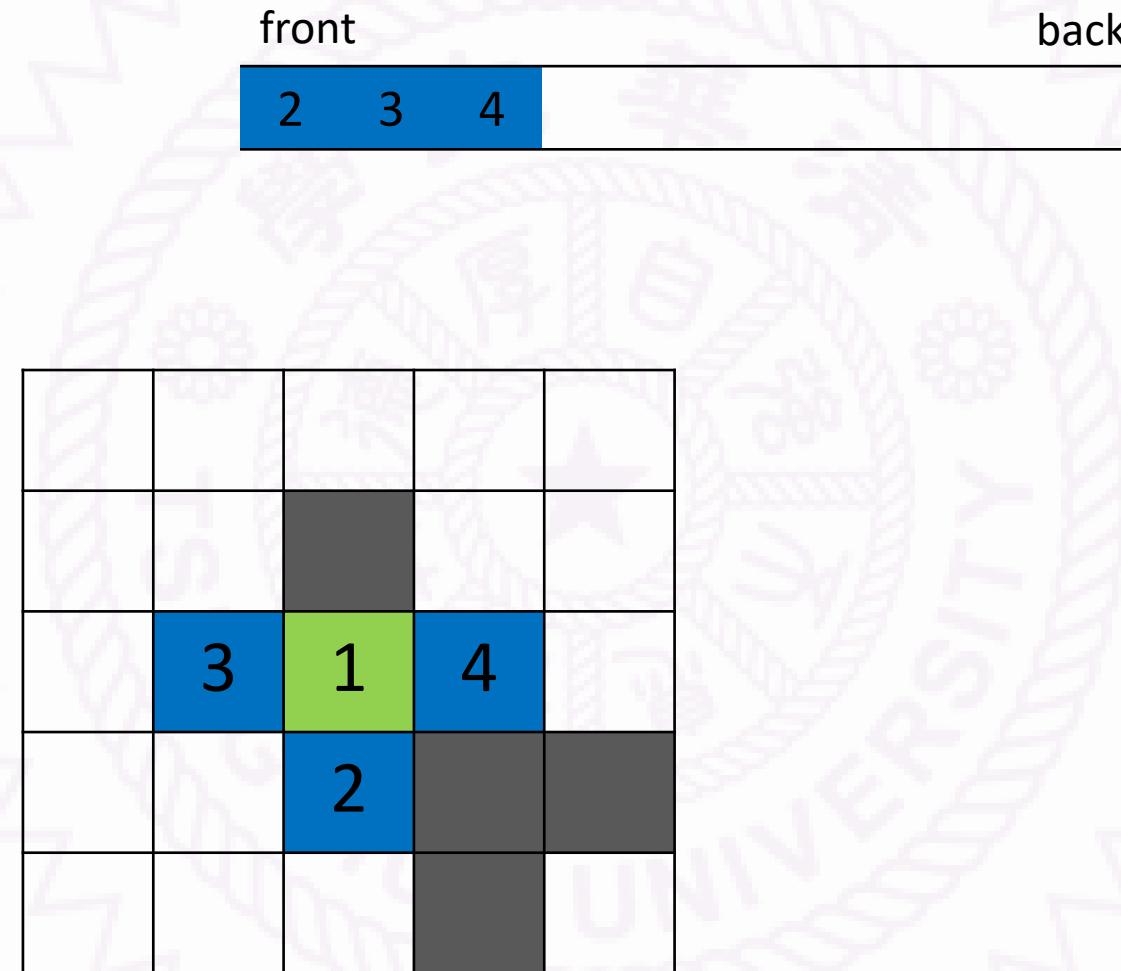
front

back

1

A 5x5 grid of squares. The squares are colored as follows: Row 1: All white. Row 2: Second from left is dark gray; others are white. Row 3: Third from left is dark gray; others are white. Row 4: Second from left is light blue; others are white. Row 5: Last three columns are dark gray; first two columns are white. The central square of the light blue row contains the number 1.

Demo



Demo

front				back
3	4	5	6	
3	1	4		
6	2			
	5			

Demo

front	4	5	6	7	8	back
		7				
8	3	1	4			
	6	2				
		5				

Demo

front	5	6	7	8	9	10	back

		7			9		
8	3	1		4	10		
	6	2					
		5					

Demo

front	6	7	8	9	10	11	back

		7		9	
8	3	1	4	10	
	6	2			
	11	5			

Demo

front	8	9	10	11	back
	7				12
	7		9		
8	3	1	4	10	
12	6	2			
	11	5			

Demo

front	8	9	10	11	12	13	14	back
-------	---	---	----	----	----	----	----	------

		13					
14	7			9			
8	3	1	4	10			
12	6	2					
	11	5					

Demo

		front			back		
		9	10	11	12	13	14
		13					
	14	7		9			
	8	3	1	4	10		
	12	6	2				
		11	5				

Demo

front	10	11	12	13	14	15	16	back
-------	----	----	----	----	----	----	----	------

	13		15	
14	7		9	16
8	3	1	4	10
12	6	2		
	11	5		

Demo

		front			back	
	11	12	13	14	15	16
		13		15		
	14	7		9	16	
	8	3	1	4	10	
	12	6	2			
		11	5			

Demo

front	12	13	14	15	16	17	back

	13		15	
14	7		9	16
8	3	1	4	10
12	6	2		
17	11	5		

Demo

front	13	14	15	16	17	back
-------	----	----	----	----	----	------

	13		15		
14	7		9	16	
8	3	1	4	10	
12	6	2			
17	11	5			

Demo

front	14	15	16	17	18	19	back
-------	----	----	----	----	----	----	------

18	13	19	15	
14	7		9	16
8	3	1	4	10
12	6	2		
17	11	5		

Demo

front	15	16	17	18	19	back
18	13	19	15			
14	7		9	16		
8	3	1	4	10		
12	6	2				
17	11	5				

Demo

front	16	17	18	19	back
	16	17	18	19	20
	18	13	19	15	20
	14	7		9	16
	8	3	1	4	10
	12	6	2		
	17	11	5		

Demo

front	17	18	19	20	back
	17	18	19	20	
18	13	19	15	20	
14	7		9	16	
8	3	1	4	10	
12	6	2			
17	11	5			

Demo

front	18	19	20	back	
	18	19	20		
	14	7		9	16
	8	3	1	4	10
	12	6	2		
	17	11	5		

Demo

front	back
19	20
18	13 19 15 20
14	7 9 16
8	3 1 4 10
12	6 2
17	11 5

Demo

front	back
20	
18	13
14	7
8	3
12	6
17	11
19	
15	9
1	4
2	
5	

Demo

18	13	19	15	20
14	7		9	16
8	3	1	4	10
12	6	2		
17	11	5		

front

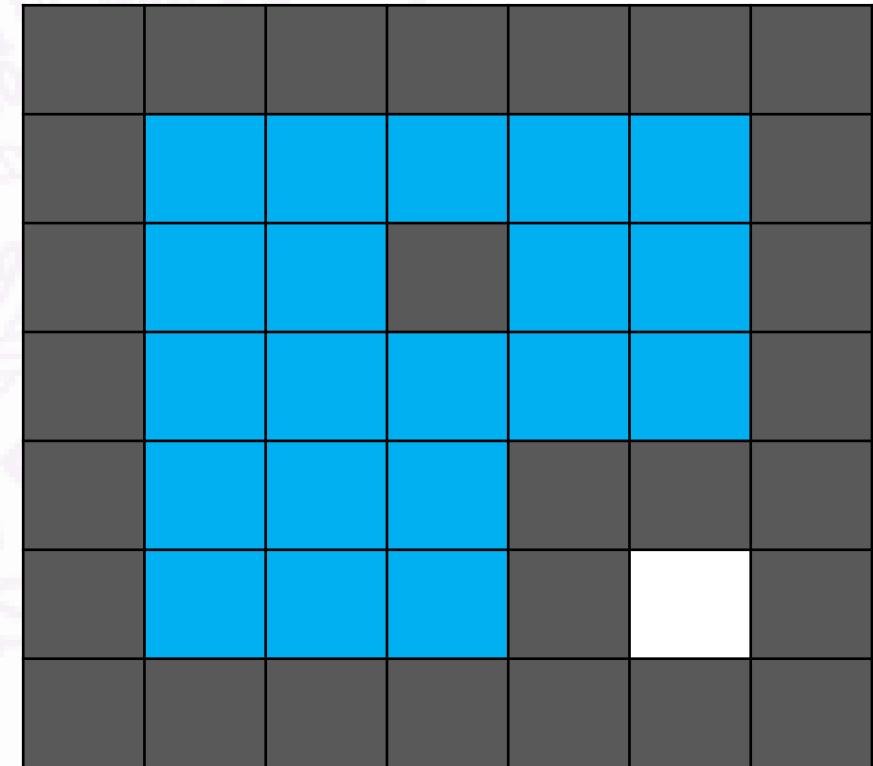
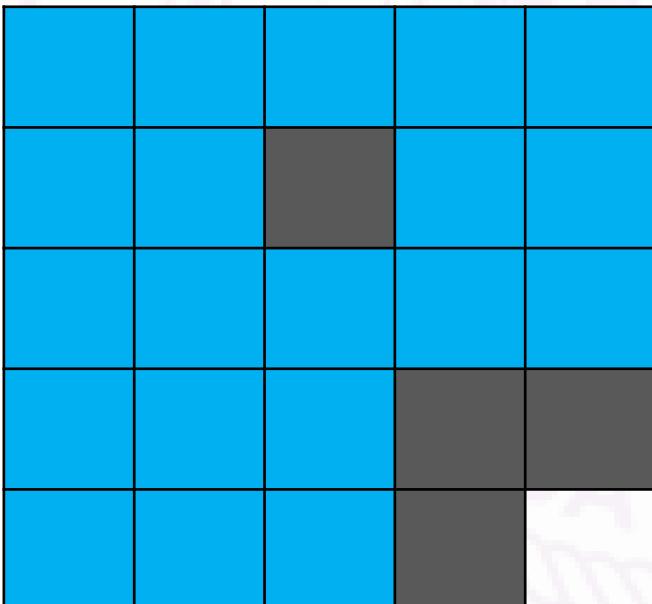
back

實作細節

- 有障礙物？就當作該格已經被填過了
- 同時有多個格子要倒水？
 - 一開始把所有有水的格子丟到queue中

實作細節

- 邊界好麻煩怎麼辦？
 - 弄一個外框，然後把外框都標記成障礙物



實作細節

```
void bfs(int x, int y) {
    std::queue<std::pair<int,int>> Q;
    Q.emplace(x, y);
    while(Q.size()) {
        std::tie(x, y) = Q.front();
        Q.pop();
        if(grid[x][y]) continue;
        grid[x][y] = true;
        Q.emplace(x+1, y);
        Q.emplace(x, y+1);
        Q.emplace(x-1, y);
        Q.emplace(x, y-1);
    }
}
```

嶄新的複雜度

- 每一格最多只會被丟進queue一次
 - 進入queue內元素不超過格子個數= $O(NM)$
- 每個格子向鄰近的格子溢出只需要常數的操作 $O(1)$
- 整體複雜度: $O(NM)$

同時擴散

- 觀察我們queue的執行結果，可以發現queue裡面「梯數」一定是非嚴格遞增的
- 只要在queue裡面多紀錄每個格子的「梯數」，然後每回合一口氣把同一個梯數的一起做完

同時擴散

		0		

同時擴散

	1	0	1	
		1		

同時擴散

	2		2	
2	1	0	1	2
	2	1		
		2		

同時擴散

	3		3	
3	2		2	3
2	1	0	1	2
3	2	1		
	3	2		

同時擴散

```
void bfs(int x, int y) {  
    std::queue<std::pair<int, int>> Q;  
    Q.emplace(x, y);  
    int L = 0;  
    while (Q.size()) {  
        for (int Num = Q.size(); Num--;) {  
            std::tie(x, y) = Q.front();  
            Q.pop();  
            if (grid[x][y])  
                continue;  
            grid[x][y] = true;  
            Level[x][y] = L;  
            Q.emplace(x + 1, y);  
            Q.emplace(x, y + 1);  
            Q.emplace(x - 1, y);  
            Q.emplace(x, y - 1);  
        }  
        L += 1;  
    }  
}
```

想一下

- 剛剛的淹水演算法中，用的資料結構是**queue**
- 如果用**stack**代替**queue**，會發生什麼事呢？

Demo

stack : 1

STK

1

		1		

Demo

stack : 2,3,4

STK	2	3	4	
-----	---	---	---	--

3	1	4		
	2			

Demo

stack : 2,3,5,6

STK	2	3	5	6
-----	---	---	---	---

			5	
	3	1	4	6
		2		

Demo

stack : 2,3,5,7

STK 2 3 5 7

			5	7
	3	1	4	6
		2		

Demo

stack : 2,3,5,8

STK 2 3 5 8

				8
			5	7
3	1	4	6	
	2			

Demo

stack : 2,3,5,9

STK 2 3 5 9

			9	8
			5	7
	3	1	4	6
		2		

Demo

stack : 2,3,5,10

STK 2 3 5 10

		10	9	8
			5	7
	3	1	4	6
		2		

Demo

stack : 2,3,5,11

STK 2 3 5 11

	11	10	9	8
			5	7
	3	1	4	6
		2		

Demo

stack : 2,3,5,12,13

STK	2	3	5	12	13
-----	---	---	---	----	----

13	11	10	9	8
	12		5	7
	3	1	4	6
		2		

Demo

stack : 2,3,5,12,14

STK 2 3 5 12 14

13	11	10	9	8
14	12		5	7
	3	1	4	6
		2		

Demo

stack : 2,3,5,12,15

STK [2 3 5 12 15]

13	11	10	9	8
14	12		5	7
15	3	1	4	6
		2		

Demo

stack : 2,3,5,12,16

STK 2 3 5 12 16

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16		2		

Demo

stack : 2,3,5,12,17,18

STK 2 3 5 12 17 18

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16	18	2		
17				

Demo

stack : 2,3,5,12,17,19

STK 2 3 5 12 17 19

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16	18	2		
17	19			

Demo

stack : 2,3,5,12,17,20

STK 2 3 5 12 17 20

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16	18	2		
17	19	20		

Demo

stack : 2,3,5,12,17

STK 2 3 5 12 17

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16	18	2		
17	19	20		

Demo

stack : 2,3,5,12

STK 2 3 5 12

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16	18	2		
17	19	20		

Demo

stack : 2,3,5

STK 2 3 5

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16	18	2		
17	19	20		

Demo

stack : 2,3

STK 2 3

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16	18	2		
17	19	20		

Demo

stack : 2

STK

2

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16	18	2		
17	19	20		

Demo

STK

13	11	10	9	8
14	12		5	7
15	3	1	4	6
16	18	2		
17	19	20		

Stack 版本

```
void dfs(int x, int y){  
    std::stack<std::pair<int,int>> STK;  
    STK.emplace(x, y);  
    while(STK.size()) {  
        std::tie(x, y) = STK.top();  
        STK.pop();  
        if(grid[x][y]) continue;  
        grid[x][y] = true;  
        STK.emplace(x+1, y);  
        STK.emplace(x, y+1);  
        STK.emplace(x-1, y);  
        STK.emplace(x, y-1);  
    }  
}
```

分類

- 用queue:
 - 廣度優先搜索(Breadth first search,BFS)
- 用stack:
 - 深度優先搜索(Depth first search,DFS)
- 事實上DFS不會像那樣實作

遞迴來實做DFS

- 在進入遞迴的時候，其實函數的資訊都會被記錄在系統提供的 stack 中
- 因此可以用遞迴實作DFS

```
void dfs(int x, int y) {  
    if(grid[x][y]) return;  
    grid[x][y] = true;  
    dfs(x+1, y);  
    dfs(x, y+1);  
    dfs(x-1, y);  
    dfs(x, y-1);  
}
```

+1 -1 很麻煩? 用 for 快速枚舉

```
void bfs(int x, int y) {
    queue<pair<int, int>> Q;
    Q.emplace(x, y);
    int L = 0;
    while (Q.size()) {
        for (int Num = Q.size(); Num--;) {
            tie(x, y) = Q.front();
            Q.pop();
            if (grid[x][y]) continue;
            grid[x][y] = true;
            Level[x][y] = L;
            for (auto [dx, dy] : Dxy)
                Q.emplace(x + dx, y + dy);
        }
        L += 1;
    }
}
```

```
pair<int, int> Dxy[4] =
    {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
```

```
void dfs(int x, int y) {
    if (grid[x][y]) return;
    grid[x][y] = true;
    for (auto [dx, dy] : Dxy)
        dfs(x + dx, y + dy);
}
```

找出步驟數最少的解

BFS 經典應用

經典題

- <https://leetcode.com/problems/water-and-jug-problem/>
- 紿你兩個容器，容量分別為 `jug1Capacity` 和 `jug2Capacity`
- 旁邊有無限供應的水源
- 一開始容器是空的，目標要讓容器的水量總和等於 `targetCapacity`
- 你只能做以下三件事
 - 將某個容器裝滿水
 - 將某個容器的水倒光
 - 將A容器的水倒到B容器中，直到B容器滿了或是A空了為止

題目問法：

- 紿你 jug1Capacity , jug2Capacity , targetCapacity
- 問法一：問你有沒有解(原本題目)
- 問法二：問你有解的話最少需要幾個步驟

範例測資

- Input: jug1Capacity = 3, jug2Capacity = 5, targetCapacity = 4
- Output: true

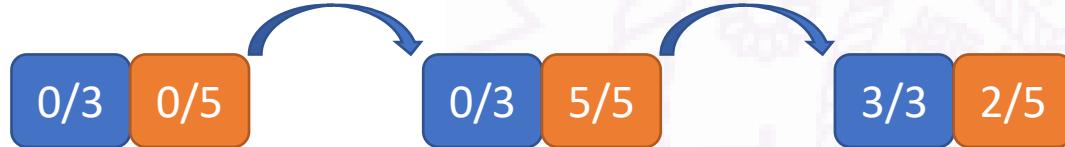
模擬一遍 $\text{targetCapacity} = 4$

0/3 0/5

模擬一遍 targetCapacity = 4



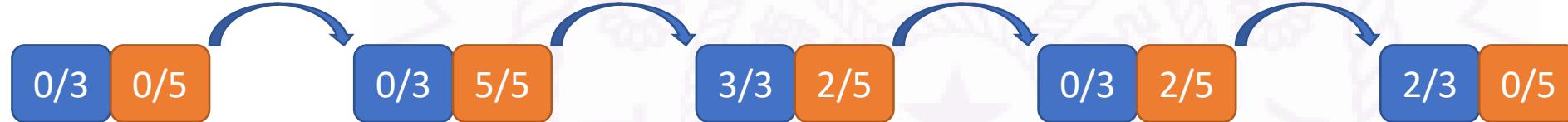
模擬一遍 targetCapacity = 4



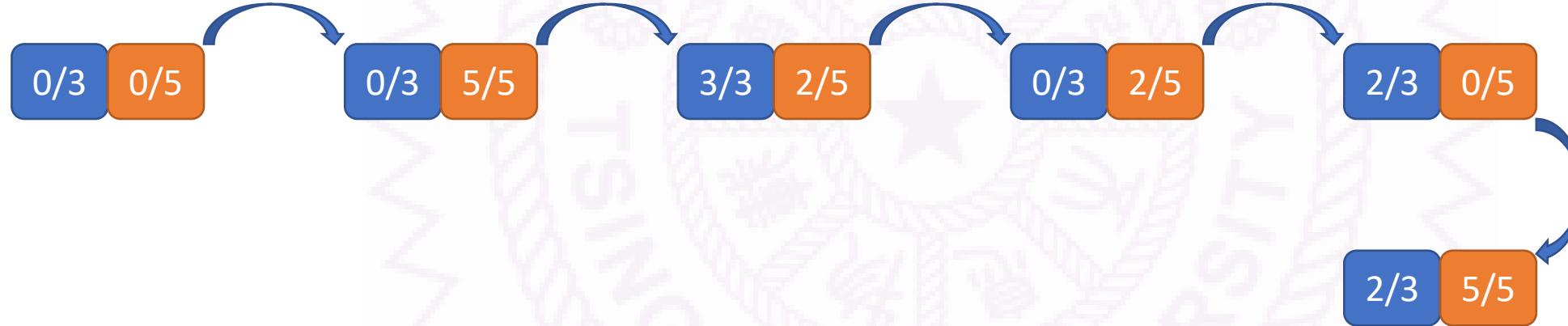
模擬一遍 targetCapacity = 4



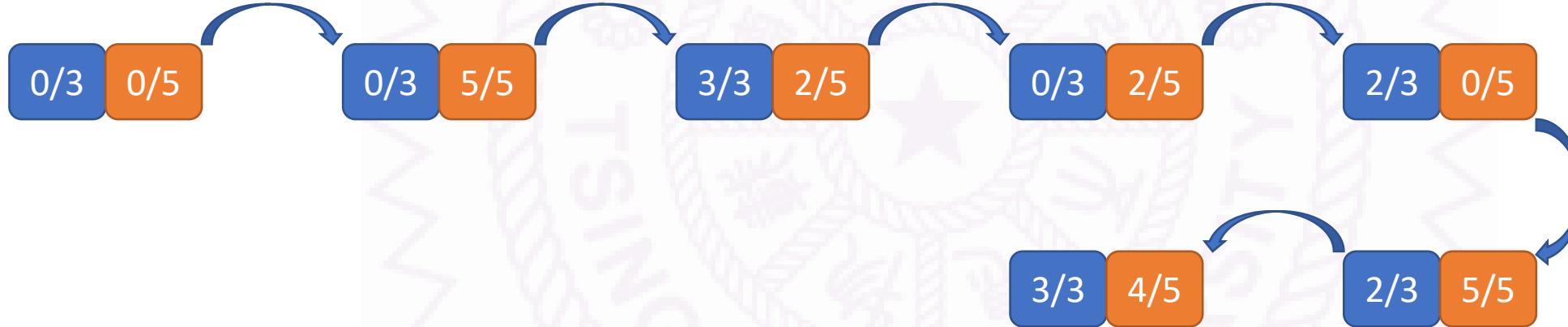
模擬一遍 targetCapacity = 4



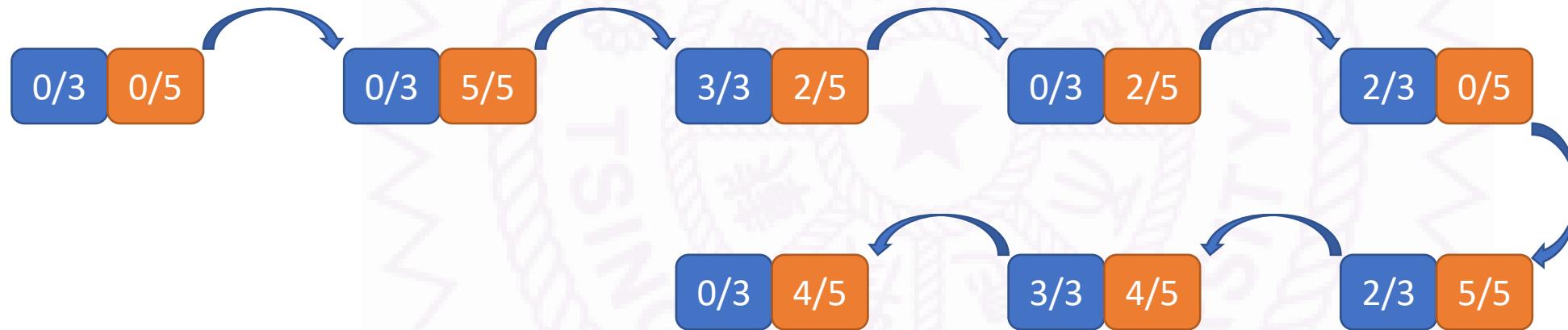
模擬一遍 targetCapacity = 4



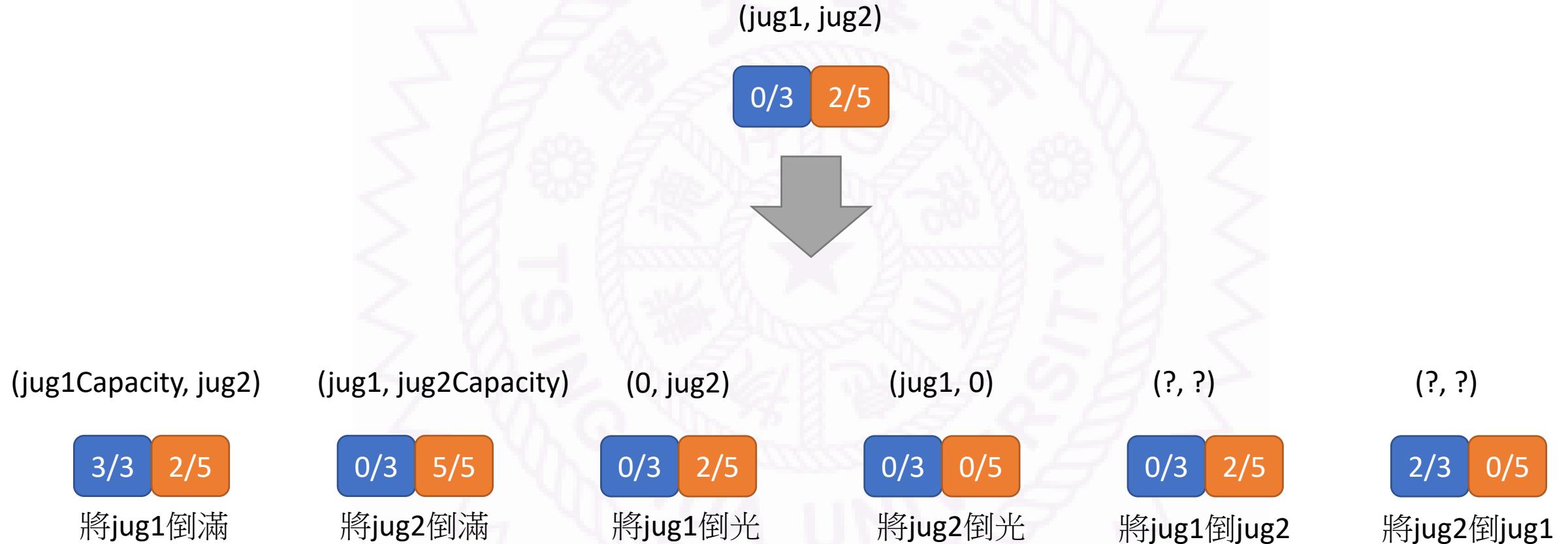
模擬一遍 targetCapacity = 4



模擬一遍 targetCapacity = 4



想法：每個狀態枚舉所有操作

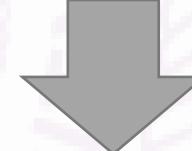


想法：每個狀態枚舉所有操作

```
jug2New = min(jug1+jug2, jug2Capacity)  
jug1New = min(jug1+jug2, jug1Capacity)
```

(jug1, jug2)

0/3 2/5



(jug1+jug2 - jug2New, jug2New) (jug1New, jug1+jug2 - jug1New)

0/3 2/5

將jug1倒jug2

2/3 0/5

將jug2倒jug1

TLE 的程式碼

```
bool canMeasureWater(int jug1Capacity, int jug2Capacity, int targetCapacity) {
    queue<pair<int, int>> Q;
    set<pair<int, int>> Visited;
    Q.emplace(0, 0);
    while (Q.size()) {
        auto [jug1, jug2] = Q.front(); Q.pop();
        if (Visited.count({jug1, jug2})) continue;
        if (jug1 + jug2 == targetCapacity) return true;
        Visited.emplace(jug1, jug2);
        Q.emplace(jug1Capacity, jug2); // jug1 倒滿
        Q.emplace(jug1, jug2Capacity); // jug2 倒滿
        Q.emplace(0, jug2); // jug1 倒光
        Q.emplace(jug1, 0); // jug2 倒光
        int jug2New = min(jug1 + jug2, jug2Capacity);
        int jug1New = min(jug1 + jug2, jug1Capacity);
        Q.emplace(jug1 + jug2 - jug2New, jug2New); // jug1 倒入 jug2
        Q.emplace(jug1New, jug1 + jug2 - jug1New); // jug2 倒入 jug1
    }
    return false;
}
```

孤岛营救问题

- <https://www.luogu.com.cn/problem/P4011>
- 有一張 $N \times M(N, M \leq 10)$ 的格子圖，圖的格子有四種：
 - 道路，可以通過
 - 障礙物，不能通過
 - 鎖住的門，身上有對應的鑰匙就可以通過
 - 鑰匙，通過時可以撿起來放身上
- 總共有 $P \leq 10$ 種門和鑰匙，編號 i 的鑰匙可以打開編號 i 的門
- 你每一步只能往上下左右其中一個方向走
請問從 $(1,1)$ 走到 (N, M) 最少須要走幾步

 第 8 层

等级 1

生命 1340

攻击 32

防御 36

魔防 2

经验 31

金币 68

 5

 0

 0

 0



是不是有點像
魔塔？

孤岛营救问题

- 像普通格子圖那樣 BFS 沒辦法考慮鑰匙和門的問題
- 有對應的鑰匙門就可以當作不存在
 - 設 $dis(x, y, keys)$ 表示走到座標 x, y ，且擁有集合 $keys$ 中的所有鑰匙所花費的最少步驟
 - 這就能用 BFS 算了
- 集合可以用二進位表示，我們枚舉時學過

讀取資料

```
int n, m, p;
int Gate[11][11][11][11];
int Keys[11][11];
void read() {
    int k, s;
    cin >> n >> m >> p;
    memset(Gate, -1, sizeof(Gate));
    for (cin >> k; k--;) {
        int x1, y1, x2, y2, g;
        cin >> x1 >> y1 >> x2 >> y2 >> g;
        Gate[x1][y1][x2][y2] = Gate[x2][y2][x1][y1] = g;
    }
    for (cin >> s; s--) {
        int x, y, key;
        cin >> x >> y >> key;
        Keys[x][y] |= (1 << key);
    }
}
```

枚舉格子

```
auto for_each_neighbor = [](int x, int y, auto &&callback) {
    pair<int, int> Dxy[4] = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
    for (auto [dx, dy] : Dxy) {
        int nx = x + dx, ny = y + dy;
        if (nx >= 1 && nx <= n && ny >= 1 && ny <= m)
            callback(nx, ny);
    }
};
```

關鍵程式碼

```
int dis[11][11][1 << 11];
int solve() {
    queue<tuple<int ,int, int>> q;
    memset(dis, 0x3f, sizeof(dis));
    dis[1][1][Keys[1][1]] = 0, q.emplace(1, 1, Keys[1][1]);
    while (q.size()) {
        auto [x, y, key] = q.front(); q.pop();
        for_each_neighbor(x, y, [&, x = x, y = y, key = key](int nx, int ny) {
            int gate = Gate[x][y][nx][ny];
            if (gate == 0 || (gate > 0 && !(key & (1 << gate)))) return;
            if (int nkey = key | Keys[nx][ny]; dis[nx][ny][nkey] > dis[x][y][key] + 1) {
                dis[nx][ny][nkey] = dis[x][y][key] + 1, q.emplace(nx, ny, nkey);
            }
        });
    }
    int ans = *min_element(dis[n][m], dis[n][m] + (1 << (p + 1)));
    return ans > 1e6 ? -1 : ans;
}
```

範例測資

Input

4	4	9	2
9	2	1	0
1	2	2	0
1	2	1	0
2	3	3	0
2	3	3	0
2	3	4	0
2	3	3	0
3	3	3	0
3	4	4	0
4	2	4	0
2	1	2	0
2	1	1	0
4	2	2	0
2	2	2	0
4	1	1	0
9	1	2	0

Output

14