Algorithms – Programming Assignment 1

PERCOLATION

B06602035 李晴妍

E-Mail : b06602035@ntu.edu.tw

Phone : 0987883537

1.

(1) Operating system: Windows
(2) Compiler: IntelliJ IDEA
(3) Text editor / IDE: IntelliJ IDEA

2. Describe how you implemented Percolation.java. How did you check whether the system percolates?

一開始是使用 2-dimension boolean array 去實作 percolation system，一開始的 union 會比較簡單，但在計算 neighboring open sites 時要考慮的條件多，也容易因為計算錯誤造成 segmentation fault。所以後來決定用一維矩陣模擬二維矩陣。因為是 n*n grid，所以可以用(row, col)去算 index，如下公式：

index = row * n + col + 1

```
private int calIndex(int row, int col) {
    return row * num + (col + 1);
}
```

一開始 grid 要多開兩個作為 top 跟 bottom。如下面表示：

0 (top)

---

x x x ... x

x x x ... x

. . (n*n)

. .

x x x ... x

---

n*n+1 (bottom)

true 表示 open，false 表示 blocked。open site 會統一存到 union cute 裡。每次 open site 時都要 check neighboring site 是否 open，是的話就 union。

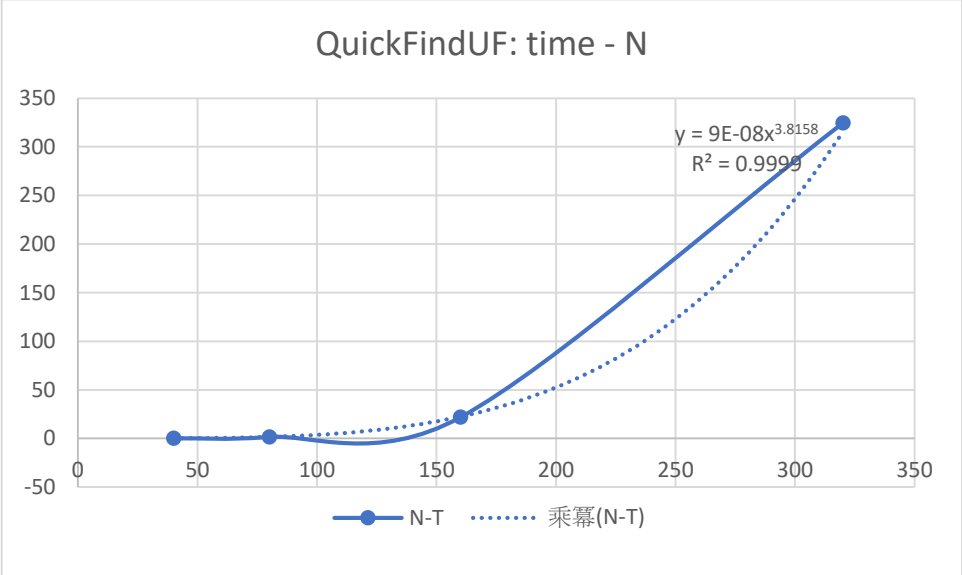
Check the system percolates 是檢查最後 top 跟 bottom 是否在同一個 set 裡。

```
public boolean percolates() {
    return (cute.find(0) == cute.find(end));
}
```

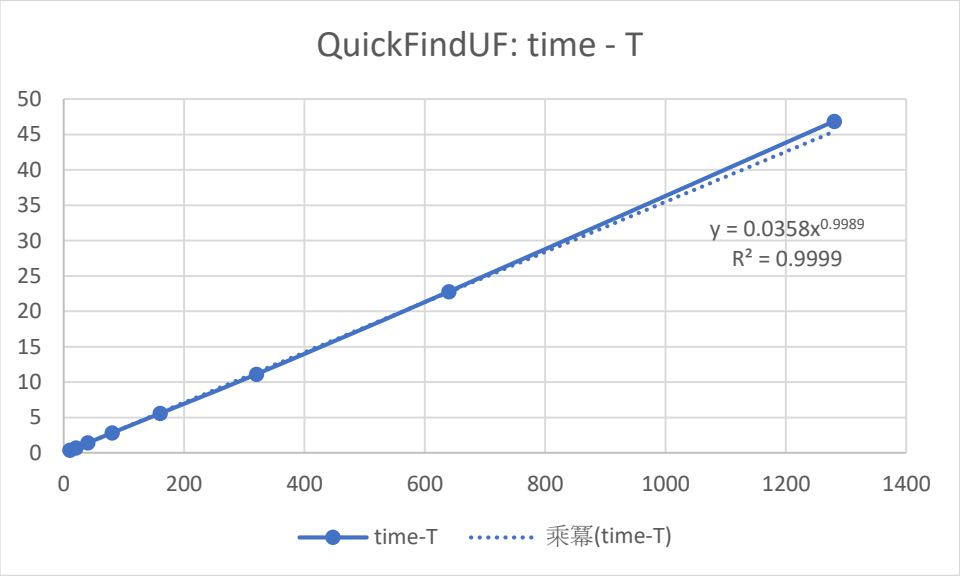
但這樣會造成 backwash，如果從 PercolationVisualizer.java 看的話，會發現是否 full 是從 isFull 這個 function 判斷的。因此解決辦法就是多開一個 union back 使用在 isFull 裡。它跟 cute 不同之處在於它少開一個 bottom，這樣就不會從 bottom 往回 union 造成 backwash。

3. Perform computational experiments to estimate the running time of PercolationStats.java for various values of n and T when implementing Percolation.java with QuickFindUF.java (not QuickUnionUF.java). Use a "doubling" hypothesis, where you successively increase either n or T by a constant multiplicative factor (not necessarily 2).
   To do so, fill in the two tables below. Each table must have 5-10 data points, ranging in time from around 0.25 seconds for the smallest data point to around 30 seconds for the largest one. Do not include data points that take less than 0.25 seconds.

multiplicative factor:	2			QuickFindUF
(keep T constant)				
T =	100	(beta)	(alpha)	
N	time	ratio	log ratio	
10	0.009			
20	0.022	2.4444	1.28951	0.0004621
40	0.116	5.2727	2.39855	1.66663E-05
80	1.551	13.371	3.741	1.17801E-07
160	21.734	14.013	3.80868	8.75679E-08
320	324.44	14.928	3.89993	5.51081E-08



multiplicative factor:			2		QuickFindUF
(keep N constant)					
N =	100		(beta)	(alpha)	
T	time	ratio	log ratio		
10	0.367				
20	0.709	1.9319	0.95001	0.04117755	
40	1.417	1.9986	0.99898	0.035558251	
80	2.821	1.9908	0.99337	0.036302495	
160	5.579	1.9777	0.9838	0.037856754	
320	11.115	1.9923	0.99443	0.035868602	
640	22.797	2.051	1.03634	0.02816646	
1280	46.849	2.0551	1.03917	0.027654619	



4. Using the empirical data from the above two tables, give a formula (using tilde notation) for the running time (in seconds) of PercolationStats.java as function of both n and T, such as

   ~ 5.3*10^-8 * n^5.0 * T^1.5

   Briefly explain how you determined the formula for the running time. Recall that with tilde notation, you include both the coefficient and exponents of the leading term (but not lower-order terms). Round each coefficient and exponent to two significant digits.

   QuickFindUF running time (in seconds) as a function of n and T:

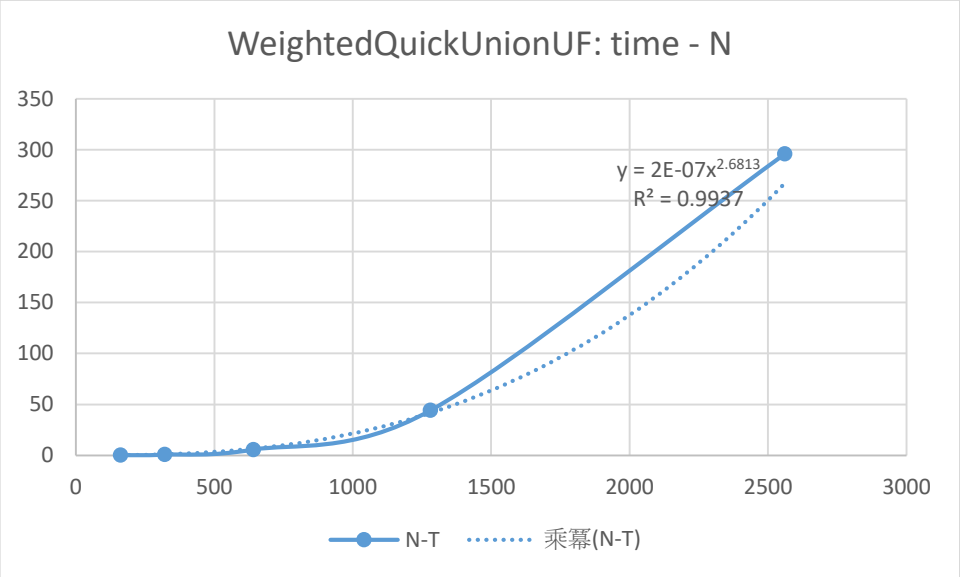
   ~ $\text{Time} = 8.760 \times 10^{-8} \times N^{3.81}$

   ~ $\text{Time} = 0.036 \times T^{0.99}$

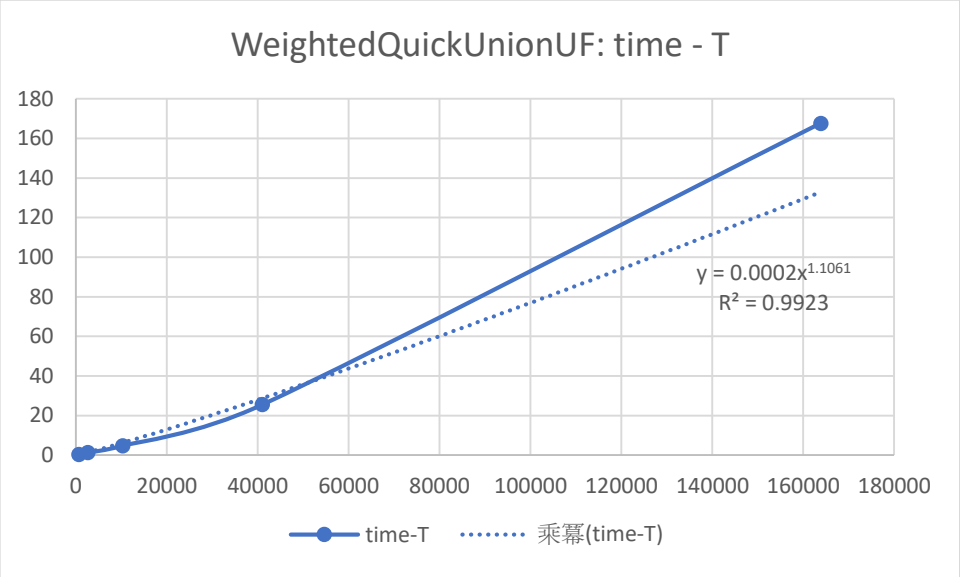
---

5. Repeat the previous two questions, but using WeightedQuickUnionUF (instead of QuickFindUF).

multiplicative factor:			2		WeightedQuickUnionUF
(keep T constant)					
T =	100		(beta)	(alpha)	
N	time	ratio	log ratio		
10	0.007				
20	0.012	1.7143	0.77761	0.001168128	
40	0.025	2.0833	1.05889	0.000502955	
80	0.07	2.8	1.48543	0.000104279	
160	0.211	3.0143	1.59182	6.54227E-05	
320	0.735	3.4834	1.8005	2.26862E-05	
640	5.692	7.7442	2.95312	2.93954E-08	
1280	44.1	7.7477	2.95377	2.92719E-08	
2560	295.8	6.7074	2.74576	1.29657E-07	



multiplicative factor:			4		WeightedQuickUnionUF
(keep N constant)					
N =	100		(beta)	(alpha)	
T	time	ratio	log ratio		
10	0.019				
40	0.042	2.2105	0.57219	0.005088135	
160	0.115	2.7381	0.72659	0.002878819	
640	0.354	3.0783	0.81106	0.001875121	
2560	1.254	3.5424	0.91236	0.000974456	
10240	4.771	3.8046	0.96388	0.000650388	
40960	25.522	5.3494	1.20969	6.72038E-05	
163840	167.68	6.5701	1.35795	1.3917E-05	

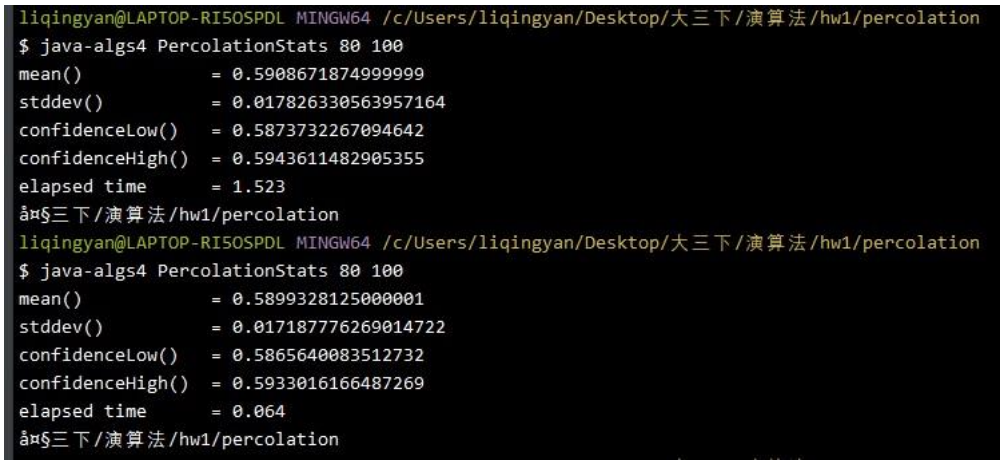


WeightedQuickUnionUF running time (in seconds) as a function of n and T:

$$\sim \quad \text{Time} = 2.93 \times 10^{-8} \times N^{2.95}$$

$$\sim \quad \text{Time} = 6.70 \times 10^{-5} \times T^{1.2}$$

QuickFindUF & WeightedQuickUnionUF 速度比較：
(上：QuickFindUF，下：WeightedQuickUnionUF)



6. Known bugs / limitations.

   必須是 n*n 的矩陣，假如今天不是 n*n 的矩陣，整個 code 都需要重寫。

7. Describe whatever help (if any) that you received. Don't include readings, lectures, and precepts, but do include any help from people (including course staff, lab TAs, classmates, and friends) and attribute them by name.

   我從工海三楊千瑩接受幫助，她告訴我一維矩陣的這個方法。

8. Describe any serious problems you encountered.

   最開始卡在二維矩陣的部分，很難去計算上下左右的 open sites，整個程式變得很冗長，也一直在 debug，後來換成一維矩陣就簡單很多了，只需要多一個 index 的轉換，就解決了這個問題。所以程式有很多種工具跟寫法，可以多試試看。

9. List any other comments here. Feel free to provide any feedback on how much you learned from doing the assignment, and whether you enjoyed doing it.

   程式本身並不難，但最後做出來的結果從 visualizer 看卻很厲害。沒想到看上去很難的東西其實可以很簡單的實作出來，所以這次作業其實很有趣也很有成就感。比較困難的地方應該在理解題目上面，因為除了本身作業的簡介，還需要懂得怎麼使用它已經寫好的 code。這部分反而是我花最多時間的地方。