HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

Fall 2017

T-301-REIR, Reiknirit

# X1: Percolation

Edda Steinunn Rúnardóttir
kt. 241095-2909
Group 1

September 1, 2017

TA: Hannes Kristján Hannesson

## 1 Introduction

Percolation is defined as “the process of a liquid slowly passing through a filter”. An emulation of such system was modelled via a grid of symmetrical size and a predefined algorithm. The system was considered to percolate when (non-diagonal) adjacent indices in grid were classified as open and provided a passable path from top row to bottom row. The model was initialized such that the entire system was closed and no connections were present between squares.

The model relied on categorizing adjacent squares in grid into subsets. This facilitated determining whether a passable path was present from top to bottom row. A union find algorithm was used to classify squares into subsets which proved a powerful tool for modelling the square’s connections and facilitated boolean checks for connections. The algorithm provided the methods union that unifies two element into the same subset as well as the other elements in their subsets and connect which takes in two indices and determines whether they belong to same subset.

The percolation model was implemented for a statistical analysis on it’s efficiency and nature - especially the threshold $p*$, denoting percolation probablity in respect to site vacancy probability. In addition to exploring the threshold value, time complexity and efficiency are explored by experimenting on running time of program and it’s growth in respect to certain values such as number of threshold tests, different union find algorithms and size of grid.

### 1.1 Setup and Methods

The experiment methods were conducted on a MacBook Air with OS X 10.9.5, a 1,3 GHz Intel Core i5 processor, 4GB RAM and Intel HD Graphics 5000. Eclipse was the development environment chosen for modelling the percolation system and conducting experiments on it, programmed in Java.

The estimation of threshold $p*$ was given by providing average, standard deviation (via Std-Stats library) and a 95% confidence interval for $p*$ for an experiment method called Percolation-Stats.java that conducted $T$ times an experiment on a percolation system model (implemented as Percolation.java) and stored each experiment’s threshold. Results of experiments conducted were documented and explored with respect to grid size, number of experiments and threshold estimations.

### 1.2 Implementation

The percolation model consisted of two major factors; a boolean 2D array and a union find 1D array. The boolean array simply determined whether a square was open or closed. The union find array contained two more elements than the boolean array because it contained a virtual first element and last element. The virtual first element was initialized in the same subset as every element in the first row of system, while the last virtual element was initialized in the same subset as the bottom row. This structure facilitated checking whether a path was present from top to bottom (i.e. whether the system percolates) as the union find array offered a boolean method *connected* that checks if the two virtual elements were present in the same subset - the system is considered to percolate successfully if so.

## 2 Results

Running time analysis experiments were conducted on system to determine time complexity in respect to variables and explore system's efficiency. The analysis was conducted both with Quick-Find algorithm (see table 1) and with Weighted Quick-Union algorithm (see table 2).

An initial value was chosen for each variable (N and T) and an experiment was run. Then the value was doubled and used for next experiment and so on. This pattern facilitated to determine time complexity pattern. The Stopwatch.java method was used for measuring the time elapsed for each experiment. The experiments conducted were independent for each value, that is, one value was set constant while testing the other.

Generally the Weighted Quick-Union algorithm is considered more efficient and this was proven by experiment for value N, as N's time complexity was reduced from $N^4$ to $N^2$. Time complexity with respect to $T$ however remained linear with both algorithms, but overall complexity was reduced from $N^4T$ to $N^2T$ (see calculations in below chapters) so all in all, the Weighted Quick Union proved a much more efficient algorithm than the Quick-Find algorithm.

### 2.1 Results With Quick-Find

Table 1: Records of running time with individual variables N and T and a Quick-Find algorithm

(a) Running time with various values of N ($T$ fixed as 30)

| $N$ | Running time (s) |
|---|---|
| 15 | 0.02 |
| 30 | 0.04 |
| 60 | 0.14 |
| 120 | 1.28 |
| 240 | 19.8 |
| 480 | 275.3 |

(b) Running time with various values T ($N$ fixed as 30)

| $T$ | Running time (s) |
|---|---|
| 15 | 0.029 |
| 30 | 0.035 |
| 60 | 0.058 |
| 120 | 0.08 |
| 240 | 0.118 |
| 480 | 0.18 |

**Time complexity** Running time was determined by applying to both experiment sets the following equation for time complexity: $T(X_i) = aX_i^b$ whereas $b$ was determined as $T(X_i)/T(X_{i-1})$ where $X_i$ is value for variable X in a single experiment $_i$ and $T(X)$ denotes running time for variable X .First individual values were checked. The tilde notation with respect only to $N$ is approximately : $\sim 5 * 10^{-9} * N^4$. The tilde notation with respect only to $T$ is approximately : $\sim 9 * 10^{-4} * T$. Both equations were approximately consistent with experiments.

**Therefore, the running time as a function of $N$ and $T$ is : $\sim T(N,T) = N^4T$ .**

### 2.2 Results with Weighted Quick-Union

Table 2: Records of running time with individual variables N and T and a Weighted Quick-Union algorithm

(a) Running time with various values of $N$ ($T$ fixed as 30)

| $N$ | Running time (s) |
|---|---|
| 15 | 0.013 |
| 30 | 0.025 |
| 60 | 0.057 |
| 120 | 0.118 |
| 240 | 0.25 |
| 480 | 0.699 |

(b) Running time with various values of $T$ ($N$ fixed as 30)

| $T$ | Running time (s) |
|---|---|
| 15 | 0.017 |
| 30 | 0.024 |
| 60 | 0.036 |
| 120 | 0.053 |
| 240 | 0.077 |
| 480 | 0.128 |

**Time complexity** Running time was determined by applying to both experiment sets the following equation for time complexity: $T(X_i) = aX_i^b$ whereas $b$ was determined as $T(X_i)/T(X_{i-1})$ where $X_i$ is value for variable X in a single experiment $_i$ and $T(X)$ denotes running time for variable X .First individual values were checked. The tilde notation with respect only to $N$ is approximately : $\sim 8.7 * 10^{-4} * N^2$. The tilde notation with respect only to $T$ is approximately : $\sim 8 * 10^{-4} * T$. Both equations were approximately consistent with experiments.

**Therefore, the running time as a function of $N$ and $T$ is : $\sim T(N,T) = N^2T$.**

### 2.3 Memory Usage

The percolation model uses a 2D array of size $N * N$, a union-find algorithm 1D array as well as three integer values that determine the grid size, the union find algorithm array's size and number of open sites in model.

· **2D array**: 24 bytes (header info) + $N$ bytes + $N$16 bytes (overhead) + $N^2$ bytes

· **1D array**: 24 (header info) bytes + N bytes

· **3 integers**: $4 * 3 = 12$ bytes

**Therefore, a tilde notation of the percolation model's memory usage is : $\sim N^2$.**

## 3 About This Solution

Have you taken (part of) this course before: **No**
Hours to complete assignment (optional): **15+**

### 3.1 Quiz on Collaboration

1. How much help can you give a fellow student taking REIR?

(a) None. Only the TAs can help.

(b) **You can discuss ideas and concepts but students can get help debugging their code only from a TA, or student who has already passed REIR.**

(c) You can help a student by discussing ideas, selecting data structures, and debugging their code.

(d) You can help a student by emailing him/her your code.

**Answer: B. Discussing problems and learning how to solve them together is the best way to remember concepts and gather a better understanding of them. Debugging however cannot be done with a fellow student as code cannot be shared. The TAs or former REIR students are more equipped to help with debugging and can provide better insights (assuming given student that has passed REIR still can understand necessary concept and therefore is mostly equivalent to a TA).**

2. What is the expectation when partnering?

   (a) You and your partner split the assignment between you and solve it individually.

   (b) **You and your partner discuss all the problems together, but code individually.**

   (c) You and your partner discuss the problems and write all the code together.

   **Answer: B. As said above, discussing problems and learning how to solve them together is the best way to remember concepts and gather a better understanding of them. Individual coding however ensures participation and understanding from all partners.**

## 3.2 Known Bugs / Limitations.

The "backwash" problem is present in my solution. That is, if a system percolates and a unconnected square is opened in bottom row, the specified square is considered full while it should not be. This was a limitation which could not be fixed before the due date of assignment.

## 3.3 Help Received

None apart from lesson classes with TA. The TA (Hannes Kristján Hannesson) cleared up what I found unclear in project and helped me to understand how to determine time complexity.

## 3.4 Problem Encountered

No serious problems were encountered, the assignment went OK although extremely time consuming. The backwash however remains a problem, but only due to lack of time. Fortunately the description was straightforward, necessary methods had been explained in classes and TAs where helpful.