

CS 25100 (12283,LE2), Data Structures and Algorithms, Fall 2016

Project 1: Waterflow with Delays

- **Due:** Friday, September 16, 11:59pm
- **Late Penalty:** 20% per day (see [late policy on programming projects](#))
- Skeleton Code: [project1.zip](#)

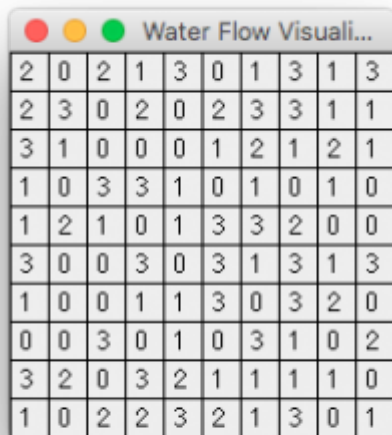
Introduction

Consider an n by m grid representing a geographical area or a porous surface. The top-left corner of the grid is position $[0][0]$. Water is present at time 0 in all cells in row 0 and flows through the surface. Each grid cell has a flow factor which is an integer between 0 and 3. The value 0 means that water cannot flow through the cell. A positive flow factor represents the delay with which the water arriving at the cell at time t flows to its three neighboring cells (horizontally and vertically). For a cell at position $[i][j]$, the neighboring cells water can flow to are $[i+1][j]$, $[i][j-1]$, and $[i][j+1]$ (assuming they exist).

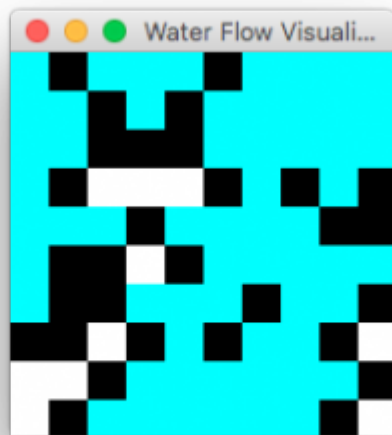
Let `flowGrid` be the array of size n by m storing the flow factors:

- if `flowGrid[i][j]=0`, water arriving at time t at cell $[i][j]$ does not leave the cell.
- if `flowGrid[i][j]=1`, water arriving at time t at cell $[i][j]$ flows to the neighbors of the cell `flowGrid[i][j]` at time $t+1$.
- if `flowGrid[i][j]=2`, water arriving at time t at cell $[i][j]$ flows to the neighbors of the cell `flowGrid[i][j]` at time $t+2$.
- if `flowGrid[i][j]=3`, water arriving at time t at cell $[i][j]$ flows to the neighbors of the cell `flowGrid[i][j]` at time $t+3$.

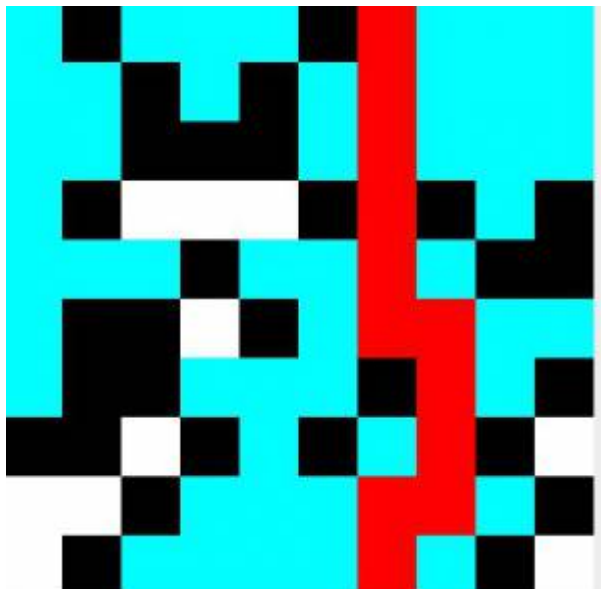
The three images below show (from left to right): a `flowGrid` of size 10 by 10 (at time 0, water is in all cells in row 0); the flow at time $t=9$ (cells reached by water are indicated in blue), and the final flow (white cells are never reached by water).



Water Flow Visuali...									
2	0	2	1	3	0	1	3	1	3
2	3	0	2	0	2	3	3	1	1
3	1	0	0	0	1	2	1	2	1
1	0	3	3	1	0	1	0	1	0
1	2	1	0	1	3	3	2	0	0
3	0	0	3	0	3	1	3	1	3
1	0	0	1	1	3	0	3	2	0
0	0	3	0	1	0	3	1	0	2
3	2	0	3	2	1	1	1	1	0
1	0	2	2	3	2	1	3	0	1



One of the goal of the project is to determine, for a given flow grid, the earliest time at which flow **leaves row n-1**. If no flow leaves row n-1, we say that no flow path exists. The left image below shows in red the path of a flow leaving at earliest time (it leaves cell [9][6] at time 21). We refer to such a path as an *earliest flow path (EFP)*. Note that there can be multiple earliest flow paths (we specify below how ties are to be broken). The right image shows a flowGrid of size 50 by 50 for which no flow exists.



In addition to detecting whether a given flow grid contains a flow and determining the earliest time flow leaves row $n-1$, you will need to list the cells on an earliest flow path (in the order cells are reached from row 0).

The data structures used for flow detection include 2-dimensional arrays and an appropriate data structure to manage when the flow reaches cells. You will be provided with a visualization method you can use and a method for generating flow grids. You will answer a number of questions requiring experimental or analytical analysis.

Determining Flow

The input is a 2-dimensional array `flowGrid` having n rows and m columns where `flowGrid[i][j]` holds the integer flow factor of location $[i][j]$, $0 \leq \text{flowGrid}[i][j] \leq 3$. We provide a number of input grids for

testing as well as a method `GenerateGrid` (described below). Note that indices start at 0: **the first row is row 0, first column is column 0**.

You need to write a method `determineFlow` determining the entries of array `reachGrid`. Array `reachGrid` is an n by m array where `reachGrid[i][j]` represents the earliest time flow reaches cell `[i][j]`; if no flow can reach the cell, set the value to -1. Once the entries of `reachGrid` are known, you can determine

- the cell in row $n-1$ from which flow leaves the flow grid at the earliest time. In case of a tie, choose the cell with the smallest column index.
- whether there exists no flow out of row $n-1$.

Recall that at time 0, water is present at all cells in row 0. To determine the entries of `reachGrid`, you should use an approach that iteratively determines all cells that can be reached at time 1, then all cells that can be reached at time 2, all cells that can be reached at time 3, etc. Termination happens when no further exploration is possible. In some sense, your computation simulates how the water flows through the grid. Do not use shortest paths algorithms to determine the flow.

You need to choose an appropriate data structure to manage the movement of the flow. You can use linked list-based implementation of data structures (stacks, queues, bags) available in the [Princeton Library](#) associated with the textbook, from the standard Java library, or write your own linked list implementation. **Do not use an array-based data structure such as `ArrayList`**. If you use code from any other source, you need to reference the source.

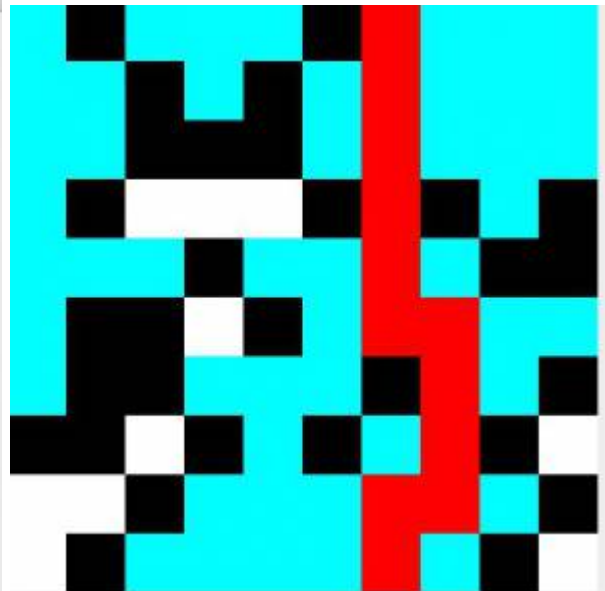
You need to be efficient in the way you manage your chosen data structures and how you explore the flow grid. Use the following guidelines in developing your code:

- flow reaching a cell that was reached by flow at an earlier time has no impact on the flow properties you are computing
- a natural approach is to use three structures, with each one handling a different delay. Whatever data structure you use, you need to minimize the overall number of operations. Ideally, you want to explicitly add and then delete a cell location only once.

Generating the Earliest Flow Path

In addition to method `DetermineFlow`, you need to write a method `earliestFlowPath` which returns the cells on a flow path leaving row $n-1$ at the earliest time. Use array `reachGrid` to generate the path, starting with a cell in row $n-1$ and moving towards row 0. The right image below shows the EFP in red: it contains 12 cells, starting at cell `[0][6]` and ending at cell `[9][6]`.

Water Flow Visualization									
2	0	2	1	3	0	1	3	1	3
2	3	0	2	0	2	3	3	1	1
3	1	0	0	0	1	2	1	2	1
1	0	3	3	1	0	1	0	1	0
1	2	1	0	1	3	3	2	0	0
3	0	0	3	0	3	1	3	1	3
1	0	0	1	1	3	0	3	2	0
0	0	3	0	1	0	3	1	0	2
3	2	0	3	2	1	1	1	1	0
1	0	2	2	3	2	1	3	0	1



Consider the example we used earlier. The earliest time at which flow leaves the grid is time 21. It leaves from cell [9][6]: it arrives there at time 20 and the cell has a flow factor of 1. How do we decide on where the flow came from? the flow could have come from the cell to the left, to the right, or above. Method `earliestFlowPath` should consider each one of these three cells and choose the right one. Always give preference to the smallest column index.

Water Flow Visualization									
0	-1	0	0	0	-1	0	0	0	0
2	4	-1	1	-1	4	1	2	1	2
4	7	-1	-1	-1	6	4	4	2	3
7	-1	-1	-1	-1	-1	6	-1	4	-1
8	9	11	-1	13	10	7	10	-1	-1
9	-1	-1	-1	-1	11	10	11	14	15
12	-1	-1	18	17	14	-1	14	15	-1
-1	-1	-1	-1	18	-1	18	17	-1	-1
-1	-1	-1	21	19	20	19	18	19	-1
-1	-1	26	24	21	21	20	19	-1	-1

Using the entries of `reachGrid` shown above, we can determine that the flow came to cell [9][6] from cell [8][6]:

- Cell [9][5] is reached at time 21 and flow leaves at time 23 (= `reachGrid[9][5] + flowGrid[9][5]`)
- Cell [9][7] is reached at time 19 and flow leaves at time 22
- Cell [8][6] is reached at time 19 and flow leaves at time 20

Use a linked list to store the earliest flow path determined (as specified in the skeleton code). You should NOT use array based linked list like `ArrayList` to store the result.

Data sets and code provided

Data sets

We provide in the skeleton a number of small flow grids you can use to test your methods. The first row contains n and m (the dimension of the grid). You can assume that all inputs will be well-formed.

We also provide a method `GenerateRandomGrid.java` to generate flow grids of a specified size and a probability of p that a cell is blocked (has value 0). You need to use this method in your experimental work.

Code Skeleton and Methods provided

You need to start your project based on given skeleton code. You can download the skeleton code from here: [Skeleton](#).

You will want to visualize the final arrays showing the flow. You can use visualization provided in the skeleton code or write your own. To enable visualization, make sure you set the following variable to be true. (The default value is true).

```
private boolean visual = true;
```

If you want to disable visualization for debugging purpose, just set the value to be false.

Make sure to read all the comments we provide in the skeleton code, as they give more detailed relevant instructions. For example, you can use the visualization to display the delay time grid, reach time grid, or the earliest flow path. If you have trouble understanding the skeleton code, please contact your TAs

Analysis Questions

Your **typed** report must answer each question briefly and clearly. Data presented in any experimental analysis should be presented in tables or charts. All material must be readable and clear and prepared in a professional manner. You can include visuals generated by your program or drawn by other means (always include in your .pdf).

Analytical Questions

1. Consider a flowGrid of size n by m grid cells. What is the *minimum* number of cells that an earliest flow path can have? Show and explain what such a path would look like. What is the *maximum* number of cells an earliest flow path can have? Show (include a visual) and explain what such paths would look like. Express all bounds in terms of n and m .
2. Choose a cell in row 0 and one in row $n-1$. What is the maximum number of cell-disjoint flow paths that can exist between the two cells? Cell-disjoint means that no two cell on different paths touch in a way that allows a flow from one path to the other. Explain and illustrate your answer.

3. Explain and justify the choice of the data structure used to determine the flow.
4. In terms of n and m and in the asymptotic sense, what is the running time of your algorithm (worst case)?
5. Assume the flow values are 0 and 1 only and $n=m$ (for simplicity) . During method `DetermineFlow`, what is the maximum number of cells that can be in your chosen data structure at a time t ? Explain your answer in terms of n and show a grid that achieves this maximum. State at what time it happens.

Experimental Analysis and Questions

Let p be the probability that a cell is blocked (i.e., its flow factor is 0). For all cells allowing flow, assume that the flow values of 1, 2, and 3 are equally likely. Method `GenerateRandomGrid.java` generates such grids.

1. For grids of size 200 by 200 and $p = 0.2, 0.3, 0.4, 0.5, 0.6, 0.7$ and grids generated randomly, what is the probability of having a flow out of row 199? Run your code on 100 randomly generated grids and report/plot the average. Describe your results. Provide either a table or a graph.
2. Perform the same experiment for grids of size 50 by 200 and 200 by 50 and $p = 0.2, 0.3, 0.4, 0.5, 0.6$. Describe your results. Provide either a table or a graph.

Grading

- **65** Methods `determineFlow` and `earliestFlowPath`, choice of data structures, efficiency and correctness of generating `reachGrid` and the earliest flow path. Do not use array-based list like `ArrayList`.
- **25** Report: Responses to the analytical question. Experimental results, any supporting visualizations, discussion and explanations.
- **10** Code Quality

Submission Instructions

You are responsible for ensuring your submission meets the requirements. If not, points will be deducted from code quality allocation. Your project will **NOT** be graded before the deadline. Your score on Vocareum will remain 0 until we grade your project.

Code

- In `WaterFlow.java`, you must correctly fill your name, login ID, your project completion date, and your PSO section number.
- You should submit a single folder named as "project1" contains the followings
- project1
 - All `.java` files from the skeleton
 - Any extra `.java` files you created or used (such as the Princeton libraries)
 - Your report (see below)

Report

- File name: <username>.pdf
- Your report must be typed.
 - Use LaTeX, or OpenOffice Write, or Microsoft Word.
 - State your name on top of the report.
 - Handwritten reports, including digitized versions, will not be accepted.
 - Figures and diagrams may be hand drawn and digitized for inclusion into the .pdf file representing your report.
- Your report must be submitted as a PDF.
 - Other file formats will not be accepted.
- Your report must be submitted using Vocareum.
 - Submissions via any other means, to any other venue, will not be accepted.

All work must be submitted in Vocareum following the submission instructions at [Vocareum Submission Guidelines](#)

From:

<http://courses.cs.purdue.edu/> - **Computer Science Courses**

Permanent link:

<http://courses.cs.purdue.edu/cs25100:fall16-le2:project1>

Last update: **2016/09/07 15:06**