Computer Science Courses

Project 1: Dynamic Connectivity

- Project starts on: Friday, Sep 11, 2015
- Project due date: **Thursday Sep 24, 2015**

> Please read the description very carefully. You will stand to lose significant amount of points if you do not follow instructions perfectly in your projects.

Purpose: Understand disjoint set union/find uses for connectivity in networks

Input: A 2D grid of size m x n and connection information between neighboring cells in the grid

Output: A list of connected subsets, their respective size and extent, plus the root of the subset (i.e., root of the UnionFind tree)

Project skeleton

Your project skeleton can be download from Piazza.

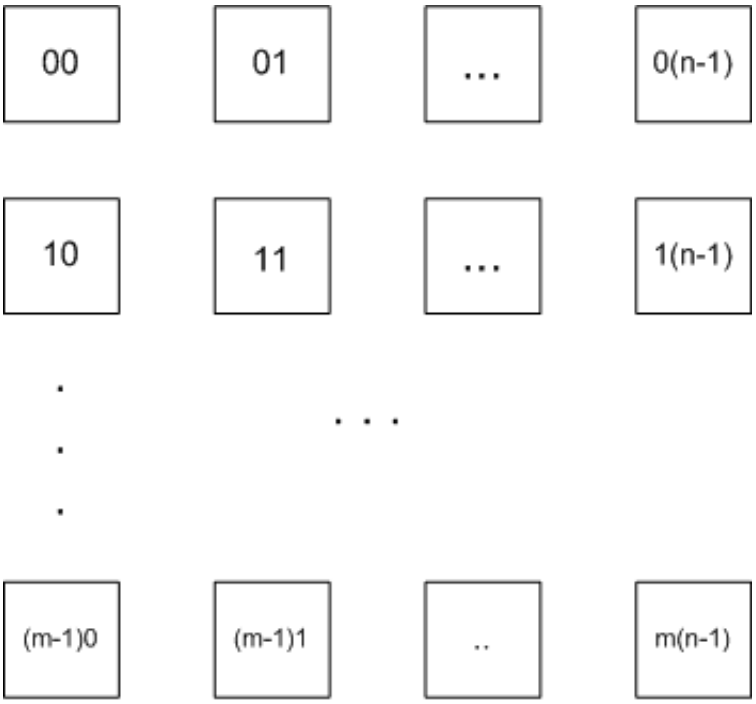
The project skeleton consists of the following files:

- **Union-Find** data structure : `WeightedQuickUnionUF.java` dependencies `StdIn.java` and `StdOut`. The implementation is generic and adapted from http://algs4.cs.princeton.edu/home/ [http://algs4.cs.princeton.edu/home/]. To better understand Union-Find, you are encouraged to start with running the sample driver inside `WeightedQuickUnionUF.java` using input file `tiny.txt`.

> Do not change the original function names and input parameters in project skeleton. You are not allowed to refactor the project skeleton. In order to get full credit, you must use the same function names, same input and output parameters.

Description

Consider a grid of size m-by-n, where m is the number of rows and n the number of columns. The position of a cell is identified by its row and column positions e.g., grid[row][col]

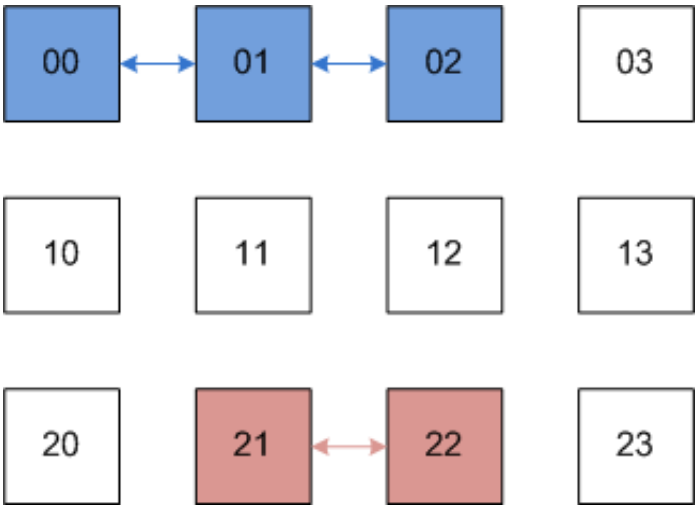


mxn grid without connections indexed by row, col positions

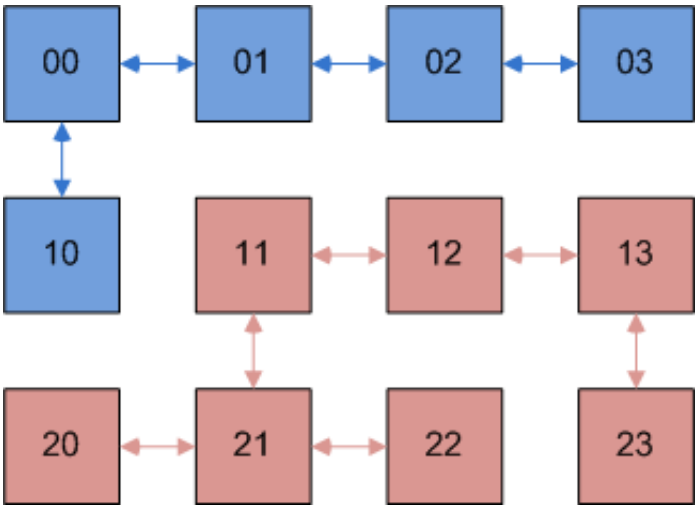
The passage connection between two neighboring cells is modeled by pair of coordinates e.g. `(0,0) , (0,1)` that is, there is a connection between grid[0][0] and grid[0][1]. For example, consider a grid of size 3 by 4, with the connection information:

```
((0,0),(0,1)), ((0,1),(0,2)),((2,1),(2,2)), ((0,1),(1,0)), ((2,0),(2,1)), ((0,3),(0,2)), ((2,1),(1,1)),
((1,1),(1,2)), ((1,2),(1,3)), ((1,3),(2,3))
```

After processing the first three pairs from input, the grid should look like:



After processing the remaining connection information, the final grid should look like:



Passages between cells that are not adjacent is not allowed. You need not check the input for mistakes.

Referring to the final grid, there are two subsets of cells, one colored blue, the other red. Furthermore, the size of the two subsets are 5 and 7 respectively. As to extent, the blue subset is in a rectangle with the NE corner 0,0 and the SW corner 1,3. the red subset lies in the bounding box NE = 1,0 and SW = 2,3.

You have to determine the smallest such bound for each final subset and print it using NE and SW coordinates.

Note: For the connection information to be legal, the two cells indicated by the information must be adjacent cells. Two cells are adjacent when they share a common side on the grid.

Brute force determination of the subsets and their size would be quadratic or worse. Instead, use the disjoint set union/find with quick union to obtain an efficient algorithm. The algorithm to be implemented has applications in, for example, penetration of porous materials. In contrast to depth-first search, memory requirements scale better in the disjoint union/find approach as the recursive depth does not grow with the problem size.

For this project, we will consider a subset to be a series of connected grid cells. The subset will be represented by the root of the subset i.e. the parent that would be returned by using the Find() call. The root is dependent on the order of the connection information for a given set, and can randomly be any position on the grid, within its set. To facilitate automated checking, you must implement the union operation as follows:

1. Given the edge (i,k), (r,s), then (i,k) is the first argument, and (r,s) is the second argument of the union call.
2. If the sets are different and of equal size, then the representative of the set of the second argument will point to the representative of the set of the first argument.

In this project you are going to implement the following methods:

Tasks

You are to implement the five (nearly) empty methods inside Project1.java:

1. `Project1`: constructor, creates grid of size m,n and creates a UnionFind structure of size m*n
2. `read_input`: Read input from user (grid size and passage connection pairs) and stores passage information in a connection list. Input for the connection pairs will be as follows.

3. `map`: converts a cell coordinates (row,col) into an integer using row major mapping*
4. `unmap`: inverse of `map`, i.e. it converts an integer value into cell coordinates (row,col)
5. `process_connections`: scans connections and populates UnionFind structure that permits determining adjacency
6. `retrieve_connected_sets`: retrieve all connected sets from the UnionFind structure, size and root
7. `output_boundaries_size`: outputs the minimum bounding rectangle of each set (i.e: NE = (min row, min col), SW = (max row, max col)) and size of each connected set

Expected Output for this program will be to first print each representative root (i.e. parent) of the sets ordered from grid[0][0] to grid[m-1][n-1] increasing by column then row, followed by the size of that set. Equivalently, the roots are sorted by the row-major order. The final output printed will be the bounding rectangle for each set, ordered in the same way as each root that was printed.

- In row-major mapping, elements of the rows are contiguous, see https://en.wikipedia.org/wiki/Row-major_order [https://en.wikipedia.org/wiki/Row-major_order] for more information.

Test Cases

```
Input 1:Enter size of grid(m n): 3 3
        Enter number of pairs of connections: 7
        0 0 0 1
        0 1 0 2
        0 0 1 0
        1 1 1 2
        1 1 2 1
        2 0 2 1
        2 1 2 2
Expected Output:
        number of sets: 2
        Parent (0,0) with size 4
        Parent (1,1) with size 5
        Bounds for parent (0,0): 0<=x<=2 0<=y<=1
        Bounds for parent (1,1): 0<=x<=2 1<=y<=2

Input 2: Enter size of grid(m n): 4 4
         Enter number of pairs of connections: 13
         0 0 1 0
         1 0 2 0
         2 0 2 1
         2 0 3 0
         0 1 0 2
         0 2 0 3
         1 1 1 2
         1 2 1 3
         0 1 1 1
         1 2 2 2
         2 2 2 3
         3 1 3 2
         3 2 3 3
Expected Output:
         number of sets: 3
         Parent (0,0) with size 5
         Parent (0,1) with size 8
         Parent (3,1) with size 3
         Bounds for parent (0,0): 0<=x<=1 0<=y<=3
         Bounds for parent (0,1): 1<=x<=3 0<=y<=2
         Bounds for parent (3,1): 1<=x<=3 3<=y<=3
```

```
Input 3: Enter size of grid(m n): 3 4
         Enter size of list: 10
         0 0 0 1
         0 1 0 2
         2 1 2 2
         0 0 1 0
         2 0 2 1
         0 2 0 3
         2 1 1 1
         1 1 1 2
         1 3 1 2
         1 3 2 3
         Set (0,0) with size 5
         Set (2,1) with size 7
         Boundaries for (0,0) are 0<=x<=3 and 0<=y<=1
         Boundaries for (2,1) are 0<=x<=3 and 1<=y<=2
```

Project Specific Instructions

Testing: You will begin by using the sanity test script provided to you. You can find the script on Piazza. Place `sanity_test` and `data` in the same directory as `Project1.java`. Assuming you are in the directory mentioned previously, you must use it as shown below:

```
$sh sanity_test.sh testCaseNumber
The output for this should be:
$"Sanity Test Passed!"
```

Furthermore, you should also create your own test cases and test your program against them. The program takes input from standard input, so you should have to feed in your test cases accordingly. It is highly recommended that you use redirection to test your programs. Talk to your TA if you want to learn how to do it. You could also just look at what `sanity_test.sh` is doing to understand how to use redirection. Move ahead only once you are convinced of the correctness of your work. PLEASE REMEMBER THAT `sanity_test.sh` IS THERE ONLY TO TEST YOUR PROGRAM AGAINST ELEMENTARY TEST CASES. It is up to you to construct all kinds of test cases and validate your work.

Grading

Overview

Tests	Points
Program Compiled and Run	10
Coding Standard	10
Passing All Test Cases	80
Total	100

Details

Program Compiled and Run: 10 pts

We can compile your program and run it successfully, according to our requirements.
We **DO** care about warnings. You will lose points if warnings are raised even though your code compiles and runs.
You should **NOT** change the signatures of the given methods.This can even lead to worse situation: failing most test cases.

Coding Standard: 10pts

Your code should be well structured.
Rule of the thumb: TA can understand any method in less than 10 seconds.
Suggestions:

1. Add Comments. Usually you will have the same amount of comments as code.
2. Friendly Variable Names.
3. Lots of small methods rather than several large methods.
4. **Indentation. You will lose all 10 points if your code is not well indented.**

Passing All Test Cases: 80pts

1. Project1.java 10 pts
2. read_input 5 pts
3. map 15 pts
4. unmap 15 pts
5. process_connections 15 pts
6. retrieve_connected_sets 15 pts
7. output_boundaries_size 5 pts

You are responsible for the robustness of the program. Passing only the provided vanilla test cases may result in very low scores.

Submission

Suggestion: We suggest you create a folder for CS251 projects like CS251_Project0, CS251_Project1 and so on. Store all you project files in that directory. To create such a folder, type "mkdir CS251_Project…"

Note:

1. you should **not** literally copy the following commands.
2. Replace yourLogin with your login ID like "zhan1015".
3. Replace "directoryContainsProject1" with the directory that contains project0 like "CS251_Project1". Do **NOT** cd into project1.

To resubmit, you just need to retype the following commands and type "yes" after the third command.

```
ssh yourLogin@data.cs.purdue.edu
cd directoryContainsProject1
turnin -v -c cs251 -p project1 project1
```

After the third command is executed, the system will give you some feedback about which files and folders have been submitted. If you resubmit a project, the previously submitted files will be overwritten.

cs25100i/fall15/projects/project1.txt · Last modified: 2015/09/11 15:46 by zhan1015