CS29003 ALGORITHMS LABORATORY
ASSIGNMENT 9
Date: 26th Sep, 2019

Important Instructions

1. **List of Input Files:** input.txt
2. **Program Files to be submitted:** ROLLNO_A9_P1.c/.cpp, ROLLNO_A9_P2.c/.cpp, ROLLNO_A9_P3.c/.cpp, ROLLNO_A9_P4.c/.cpp
3. Files must be read using file handling APIs of C/C++. Reading through input redirection isn't allowed.
4. You are to **stick to the file input output formats strictly** as per the instructions.
5. Submission through **.zip files are not allowed**.
6. Write your name and roll number at the beginning of your program.
7. Do not use any global variable unless you are explicitly instructed so.
8. Use proper indentation in your code.
9. Please follow all the guidelines. Failing to do so will cause you to lose marks.
10. **There will be part marking.**

Disjoint Sets

Problem Statement

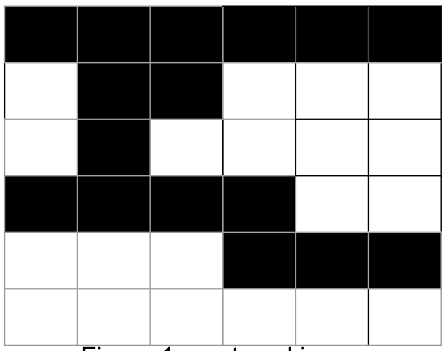


Figure 1a: not working

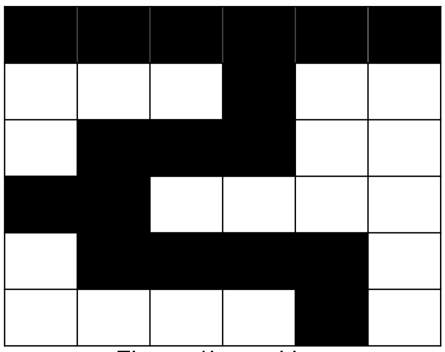


Figure 1b: working

Figure 1: Picture of Chip

An US based company XYZ has built an $n \times n$ cell network chip. The chip has one input-end and one output-end and the electric signal propagates from input-end to output-end through some cells of the chip. Some cells of the chip are able to carry electric signal and others are not. They want to check whether a chip is able to propagate the electric signal from input-end to output-end. The following figures demonstrate this problem.

In both of the figures above, top-most row represents the input-end and bottom row represents the output-end. An electric signal can propagate only through the black cells of the chip. Figure 1a demonstrates an example of a non-functional chip. Figure 1b demonstrates an example of a working chip. A cell in the chip is addressed by the pair (i, j) of its row and column indices. Consider the set of all cells:

$$C = \{(i, j) | 0 \leq i \leq n-1, 0 \leq j \leq n-1\} \tag{1}$$

You should always maintain a partition $\boldsymbol{P}$ of C consisting of mutually disjoint non-empty subsets of C, having C as their union. Initially $\boldsymbol{P}$ contains all the n^2 elements of C as singleton subsets. If it is found that two adjacent black cells belong to two different subsets in $\boldsymbol{P}$, replace these two subsets by their union. Notice that electric signal can move from a black cell to an adjacent black cell by using an common edge (not diagonally). When the traversal is complete, check whether one or more cells in the bottom row are in the same subset containing one of the elements in the topmost row.

We should keep the following function for maintaining the partition $\boldsymbol{P}$ of subsets of C.

Make-set(x)-Initialize each element x of C as a singleton set.

Find-set(x)- Find the identity (root) of the subset to which a given element x of C belongs.

Union- Given two different (so disjoint) subsets S_1, S_2 in the collection $\boldsymbol{P}$, merge them to a single set $S = (S_1 \cup S_2)$. Remove S_1 and S_2 from $\boldsymbol{P}$, and add S to $\boldsymbol{P}$.

Union by Rank and Path Compression Algorithm

Union by rank and *compression* techniques will give you better time complexity compared to other algorithms.

Union by Rank: Always attach the shorter tree to the root of the taller tree. To implement union by rank, each element is associated with a rank. Initially a set has one element and a rank of zero.

If we union two sets and

- Both trees have the same rank – the resulting set's rank is one larger.
- Both trees have different ranks – the resulting set's rank is the larger of the two. Ranks are used instead of height or depth because path compression will change the trees' heights over time.

Path compression: Is a way of flattening the structure of the tree whenever ***Find*** is used on it. Since each element visited on the way to a root is part of the same set, all of these visited elements can be reattached directly to the root. The resulting tree is much flatter, speeding up future operations not only on these elements, but also on those referencing them.

Sample Input File

FILE: *input.txt*

```
6
1 1 1 1 1 1
0 0 1 0 0 1
1 1 1 0 0 1
0 1 0 0 0 0
0 1 1 0 0 0
0 0 1 1 0 0
```

Read the input file for an $n \times n$ electric signal carrying chip in the row-major order. The black cell is entered as 1, and white cell as 0. A cell can be represented as a structure of three fields: data, rank and parent.

Note in all the parts that follow:

- Call the union function over the matrix in row major order, i.e., first $Union(a_{0,0}, a_{0,1})$ will be called, then $Union(a_{0,1}, a_{0,2})$ will be called and so on. For any black cell, you can also do a union with the upper cell.

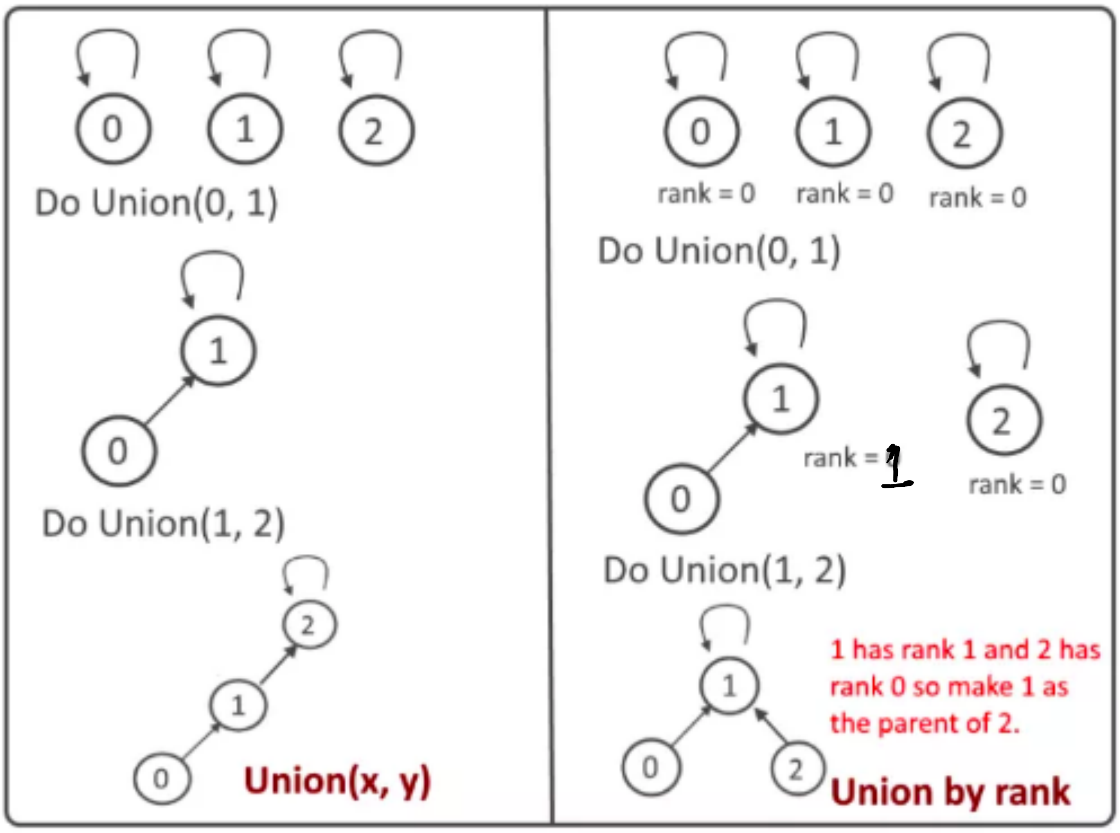
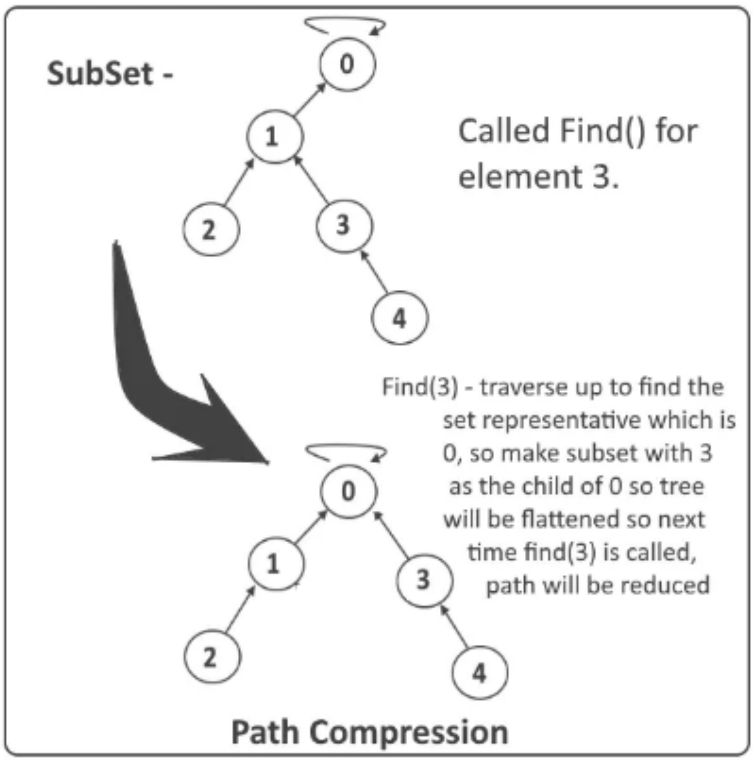


Figure 2: Union by rank



SubSet -
Called Find() for element 3.
Find(3) - traverse up to find the set representative which is 0, so make subset with 3 as the child of 0 so tree will be flattened so next time find(3) is called, path will be reduced
Path Compression

Figure 3: Path Compression

- In $Union(a, b)$, b's root will be appended to a's root, if both a and b have the same rank. Otherwise, you follow Union by rank.

Part-1

Write a function for implementing the *Union by Rank* and *Path Compression* algorithms described above in order to check whether the given chip can carry an electric signal through the black cells from the input-end to the output-end.

Part-2

Write a function for implementing the *Union by Rank* algorithm in order to find a path through which an electric signal can reach to the output-end from the input-end. If such a path exists, then print the coordinates of the cells belonging to it starting from the input-end. [from 0,0]

Part-3

A tree will be formed as a result of the union operation in the *union find* algorithm. Perform a **level order traversal** of the tree thus formed and print the cell as you traverse.

containing the i/p end

Part-4

Write a function to compute the size of the largest connected component of black cells (i.e. 1 in this case) in an $n \times n$ chip matrix.

For example: In the above input file, the largest continuous connected component of 1's add up to a total of ~~14~~. 17