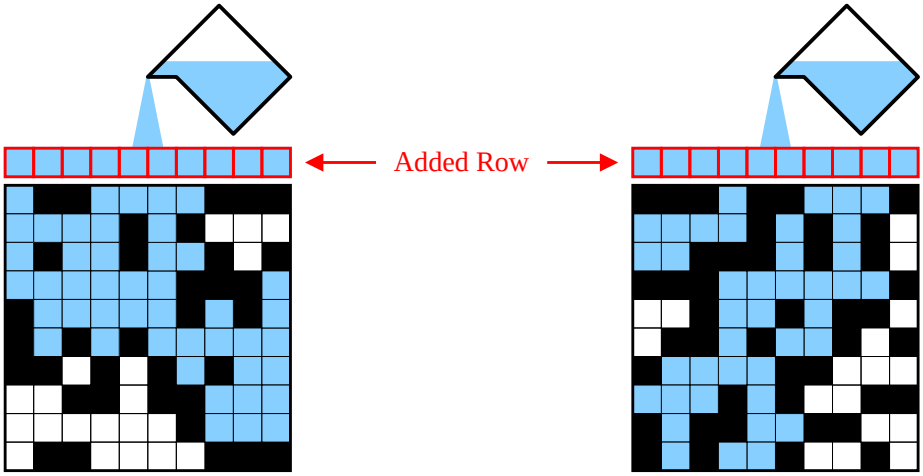
CS29003 Algorithms Laboratory

Assignment No: 8

Last date of submission: 19–March–2019

Union-Find Disjoint-Sets Data Structure

Material scientists in Barviera have built an $n \times n$ mesh of foomium. They pour a liquid called l'eau[1] on the top of the mesh. Some cells of the mesh block l'eau, whereas the other cells allow the passage of l'eau through them. The scientists want to check whether l'eau can seep through the mesh, and go out from the bottom of the mesh. The following figure demonstrates this problem. The blocking cells are shown as solid squares. The mesh on the left side of the figure does not allow seepage of l'eau from top to bottom, whereas the mesh on the right side allows. For small values of n, the problem can be solved by eye inspection only. But the meshes are obtained from x-ray scans of foomium sections, and may contain millions of rows and columns. The Barvieran scientists need you to solve their problem.



Start by adding a dummy row at the top of the mesh, and marking each cell in the added row as non-blocking. This augmented mesh is of dimensions $(n+1) \times n$. A cell in the (augmented) mesh is addressed by the pair (i, j) of its row and column indices. The dummy row corresponds to $i = 0$, the topmost row of the original mesh to $i = 1$, and the bottommost row to $i = n$. Consider the set of all cells:

$$C = \Big\{ (i, j) \mid 0 \leqslant i \leqslant n,\ 0 \leqslant j \leqslant n-1 \Big\}.$$

Maintain a partition $\mathscr{P}$ of C consisting of mutually disjoint non-empty subsets of C, having C as their union. Initially, $\mathscr{P}$ contains all the $n^2 + n$ elements of C as singleton subsets. Later, make a row-major traversal of the mesh. If you find two adjacent non-blocking cells belonging to two different subsets in $\mathscr{P}$, replace these two subsets by their union. Notice that l'eau can move from a cell to an adjacent cell by using a common edge (but not through a common corner). When the traversal is complete, check whether one or more cells in the bottommost row are in the same subset (a member of $\mathscr{P}$) as any cell in the added dummy row.

From this description of the algorithm, it follows that we require the following functions for maintaining the partition $\mathscr{P}$ of subsets of C.

initsets Initialize each element of C as a singleton set.

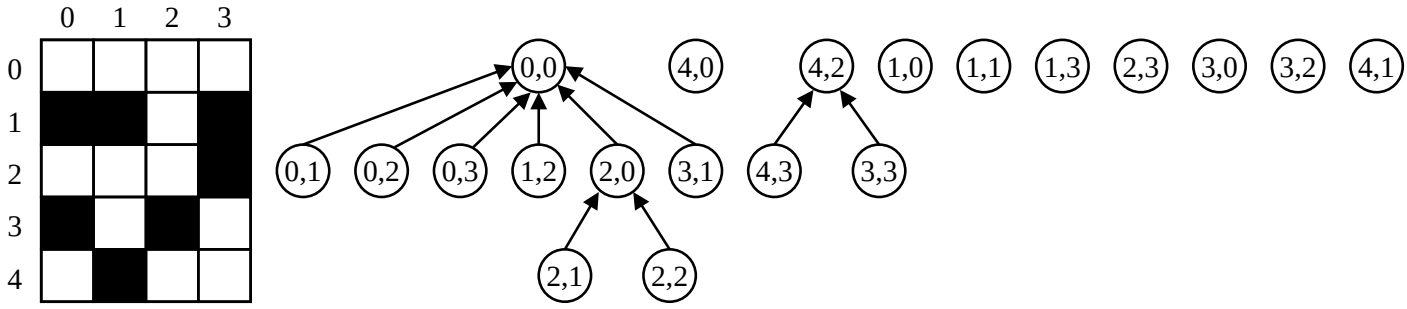
findset Find the identity of the subset to which a given element of C belongs.

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

Realize each subset in $\mathscr{P}$ as a rooted tree. Each node in the tree corresponds to an element of C, and consists of only two items: the size of the subtree rooted at that node, and a parent pointer. The set of nodes of the

[1] L'eau is the most abundant liquid in Barviera (and elsewhere too). Like the Barvieran king Napoleon, we pronounce it as *low*.

tree is the subset being represented by the tree. The parent pointer of the root is assumed to be NULL. The identity of the tree is a pointer to the root node. For a 4×4 mesh (or 5×4 augmented mesh), the final collection $\mathscr{P}$ can be represented as illustrated in the following figure. Although the cell indices are shown inside the nodes, it is actually not necessary for the cells to store this information (see below why). The figure shows the parent pointers (unless NULL), but the subtree-size fields are not shown.



Let us now come to the implementation details of this data structure.

initsets(n) Create an $(n+1) \times n$ array F of tree nodes (not node pointers). The (i,j)-th element of F stands for the (i,j)-th cell of the mesh. This node can be accessed by accessing $F_{i,j}$, that is, there is no necessity to store the pair (i,j) anywhere. Initially, we only have singleton sets. So set the subtree-size field of each $F_{i,j}$ to one, and the parent pointer of each $F_{i,j}$ to NULL.

findset(F,i,j) In order to identify the tree where a cell (i,j) belongs, we first locate the cell at $F_{i,j}$. We then keep on following the parent pointers until we hit the NULL pointer. The last non-NULL pointer on this path is the desired pointer to the root, and is returned.

mergesets(p,q) Let p and q be the roots of the trees to which two adjacent non-blocking cells (i,j) and (k,l) belong. Suppose that $p \neq q$, and we want to merge these two trees. We look at the sizes of the trees, stored at the root nodes. We make the smaller tree a child of the root of the larger tree by adjusting the parent pointer of the root of the smaller tree. The size field at the root of the larger tree should also be updated accordingly. This heuristic guarantees that the heights of all the trees in $\mathscr{P}$ are restricted to $\mathrm{O}(\log n)$, so subsequent *findset* calls take only this much time.

Part 1: Read n and the data for the $n \times n$ mesh in the row-major order. A blocking cell is entered as 0, and a non-blocking cell as 1. The 4×4 mesh of the above figure is supplied by the user as follows.

0 0 1 0
1 1 1 0
0 1 0 1
1 0 1 1

Add an extra row of non-blocking cells at the top. The user does not enter this row.

Part 2: Implement the three primitives for the disjoint-set data structure, as described above.

Part 3: Write a function *mergeregions* to implement the algorithm for solving the Barvieran l'eau-seepage problem. The function should decide whether l'eau can or cannot seep through the mesh from top to bottom. Implement the algorithm described on the first page.

Part 4: Write a function *findreach* to identify all the cells of the mesh to which l'eau, if poured at the top, can reach. Print these cells (along with all other cells of the augmented mesh) with a special marker (see the sample outputs).

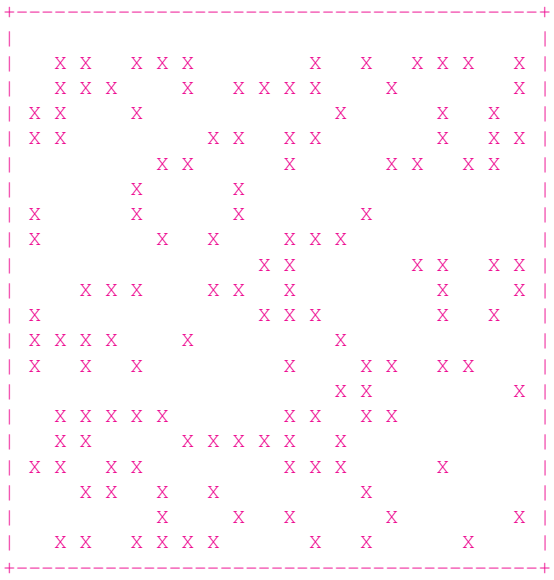
Write a `main()` function that solves Parts 1, 3, 4 in this sequence.

Submit a single C/C++ source file. Do not use global/static variables.

Sample outputs

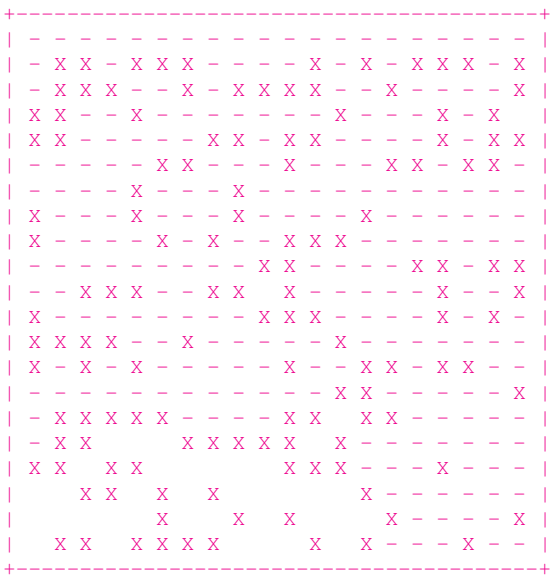
```
n = 20

1 0 0 1 0 0 0 1 1 1 1 0 1 0 1 0 0 0 1 0
1 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 1 1 1 0
0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1
0 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0
1 1 1 1 1 0 0 1 1 1 0 1 1 1 0 0 1 0 0 1
1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1
0 1 1 1 1 0 1 0 1 1 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0
1 1 0 0 0 1 1 0 0 1 0 1 1 1 1 1 0 1 1 0
0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1
0 0 0 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1
0 1 0 1 0 1 1 1 1 1 0 1 1 0 0 1 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0
1 0 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 1 1 1
1 0 0 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1
0 0 1 0 0 1 1 1 1 1 0 0 0 1 1 1 0 1 1 1
1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1 1 0
1 0 0 1 0 0 0 0 1 1 1 0 1 0 1 1 1 0 1 1
```

+++ Input mesh

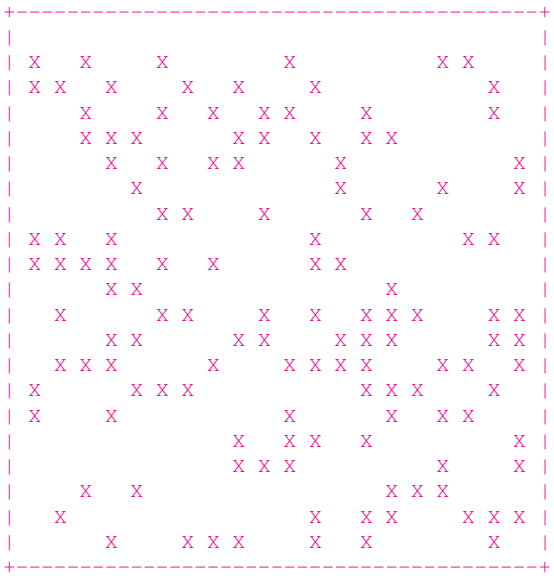


+++ Liquid can seep through the mesh

+++ Reachable cells in the mesh



```
n = 20

0 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1
0 0 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1
1 1 0 1 1 0 1 0 1 0 0 1 1 0 1 1 1 1 0 1
1 1 0 0 0 1 1 1 0 0 1 0 1 0 0 1 1 1 1 1
1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 1 1 1 0
1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0
1 1 1 1 1 0 0 1 1 0 1 1 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
0 0 0 0 1 0 1 0 1 1 1 0 0 1 1 1 1 1 1 1
1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
1 0 1 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 0 0
1 1 1 0 0 1 1 1 0 0 1 1 0 0 0 1 1 1 0 0
1 0 0 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 1 0
0 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 0 1
0 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1
1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 1 0
1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1
1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 0 0 0
1 1 1 0 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1

+++ Input mesh
```



```
+++ Liquid cannot seep through the mesh

+++ Reachable cells in the mesh
```