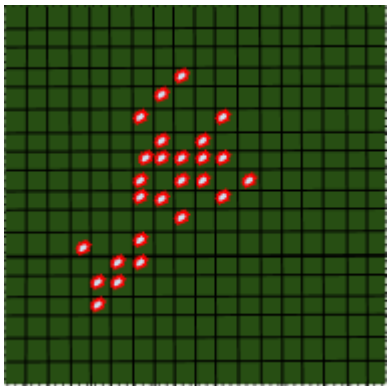
15-110: Principles Of Computation
Lab 4: Burn Baby, Burn
Released: Monday, 7-12
Due: Wednesday, 7-18, 11:59 PM



Introduction:

As we have seen in class, there are many problems which are difficult to solve and hard to understand using our normal iterative approach, but are far easier to approach from a recursive stand point. We will be covering one such problem during this lab. You will write functions which will allow you to analyze a connected group of cells on a 2-D grid. This algorithm is used in the paint bucket tool, common in many image editors.
We will be using this algorithm to calculate the size of a patch of fire inside a forest, where the the burning portions of the forest are represented by elements in a list.

Task 1: Writing the wrapper function for fireSize

Write the function `startFire(forest, row, col)`, which takes the the rectangular 2-D grid `forest` and the integers `row` and `col` as parameters. It will return the number of fire squares within the specified patch, although `fireSize` will take care of that calculation. `startFire()` should create the 2-D list `helperForest`, which is the same size as `forest` and contains the number 0 at all indices (to be used in Task 2). `startFire()` should then ensure that `forest` contains only the strings, `"fire"` and, `"no fire"` as elements. If any element is not one of these two strings, it should be changed to the string `"no fire"`. Once this has been set up, `startFire` should return `fireSize` with `forest`, `row`, and `col` as parameters.

To help you, we have provided the function `makeBlankList(height, width, element)`, which creates a 2-D list of the appropriate height and width. Every element in this new list will be `element`.

Task 2: Calculating the size of the fire

Write the function `fireSize(forest, row, col)` which takes the rectangular 2-D grid `forest` and the integers `row` and `col` as parameters. You should treat `row` and `col` as the indices of

a square in `forest`. `fireSize()` should return the size of the patch of fire which that square is a part of. Squares are considered to be part of the same patch of fire if they are connected (horizontally, vertically, or diagonally).

`fireSize()` should be a recursive function. As a recursive function, you must break it down into two parts: the base case and the recursive case. The code in your recursive case should call `fireSize()` on all the surrounding array indices and should make use of the global variable `helperForest` in order to prevent double counting and infinite loops. Your base case should test for any conditions under which you no longer need to continue calling your recursive case. If those conditions are met, your recursive case should return a simple value.

`fireSize()` should return 0 if it is called on a `row` and `col` that are outside of `forest`.