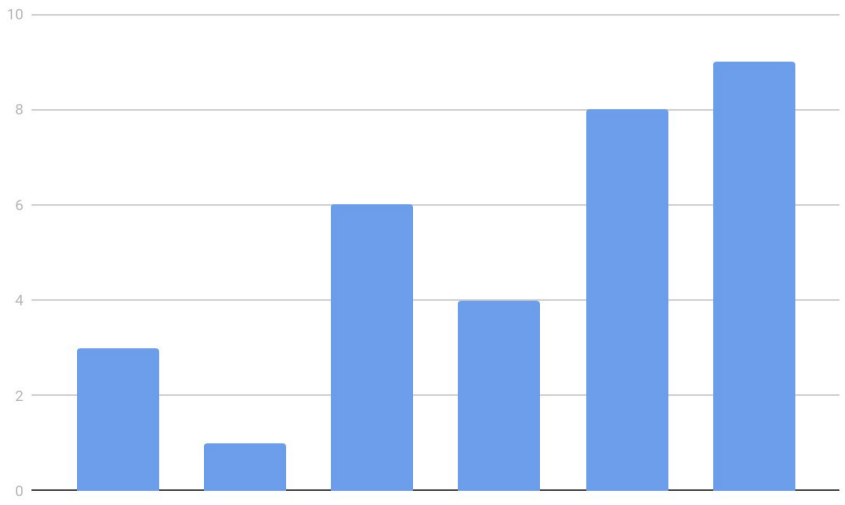
Rust Landscape Water Filling Problem

Definition

Write a program in Rust that calculates the water level in different parts of a landscape.

The landscape is defined as positive numbers.

Examples: 3,1,6,4,8,9



Then it begins to rain. Per hour one unit of rain falls on each segment of the landscape.

Water that falls on segment 3 in the above sample landscape will flow equally into segment 2 and into segment 4 until segment 4 is filled up to the level of segment 3.
Water that falls on segment 5 will flow into segment 4 because segment 6 is higher.
Water that falls on segment 6 will flow into segment 5 (and then further into segment 4) because the right to segment 6 is an infinite wall (same as left to segment 1).

To the very left and to the very right of the landscape are infinite walls, i.e. water does not flow outside of the defined landscape.

The program shall calculate the water levels for each segment after x number of hours of rain.

The user of the program shall be able to define the landscape when running the program and shall also be able to define the number of hours it will rain.

Describe an algorithm (in text form) and prove that the algorithm.

Rules

From the above definition, the following rules could be inferred:

- Water units are divisible, otherwise, the unit of water falling on segment 3 would flow into segment 2 or to segment 4 but never on both.
- Given a segment that is higher than both of his neighbor's water should split evenly to his neighbors depending on the neighbor's height.

Solution

The following solution relies on a property that some kinds of landscapes do exhibit. In the general scenario, the amount of water a segment will be able to receive will depend on its height and its position in the whole landscape. There is a landscape where the amount of water a segment will receive will depend only on the segment's height and on the next highest segment. Just for the sake of naming, we will call this degenerated landscape a *VShape*.

A *VShape* is a landscape where all the rain that falls flows into the same segment or segments. This definition covers landscapes with the same height elements, up-hill, downhill slopes, and V-shaped ones.

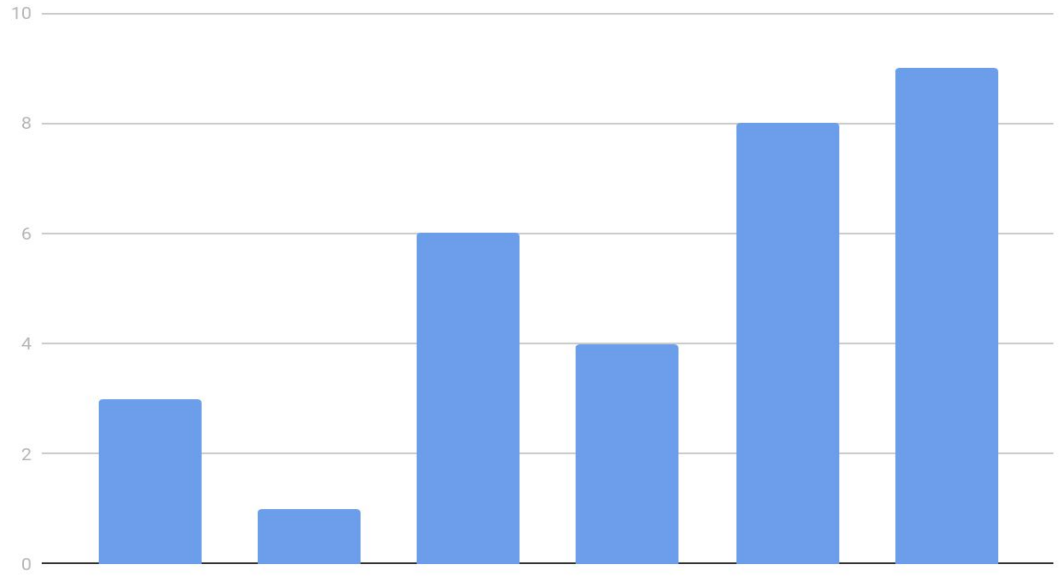
The implemented algorithm is a kind of dive and conquer algorithm. It starts by splitting a given landscape into one or more *VShapes*. Then given some amount of water it will go through every *VShape* trying to fill them. If while filling a *VShape*, it's level of water reaches the height of a neighbor VShape, the two landscapes are joined into a bigger VShape. The algorithm will finish when there's no more rain to distribute.

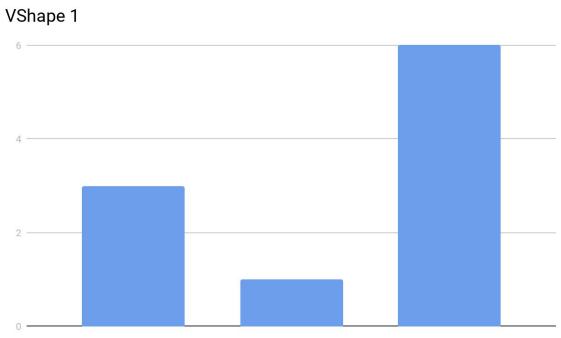
In order to be able to fill a VShape with water an algorithm as the following could be performed:

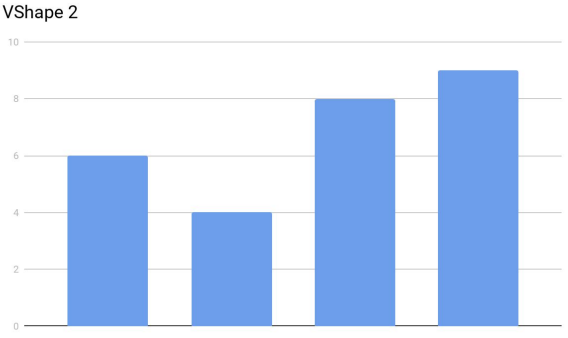
1. Start adding water to the smallest segments until they reach the next higher segment or until there is no more water
2. If when filling the smallest segments we've reached the height of a neighbor *VShape*, stop filling and update the remaining rain to distribute.
3. Goto first step

Applying this algorithm with one hour of rain to the example in the definition will lead to something like the following.

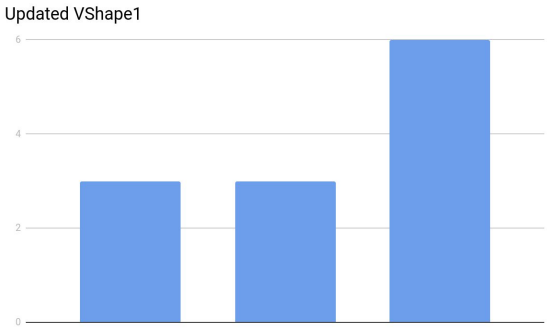
1. Split into *VShapes*

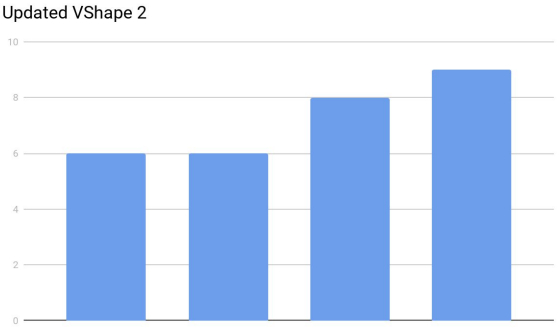




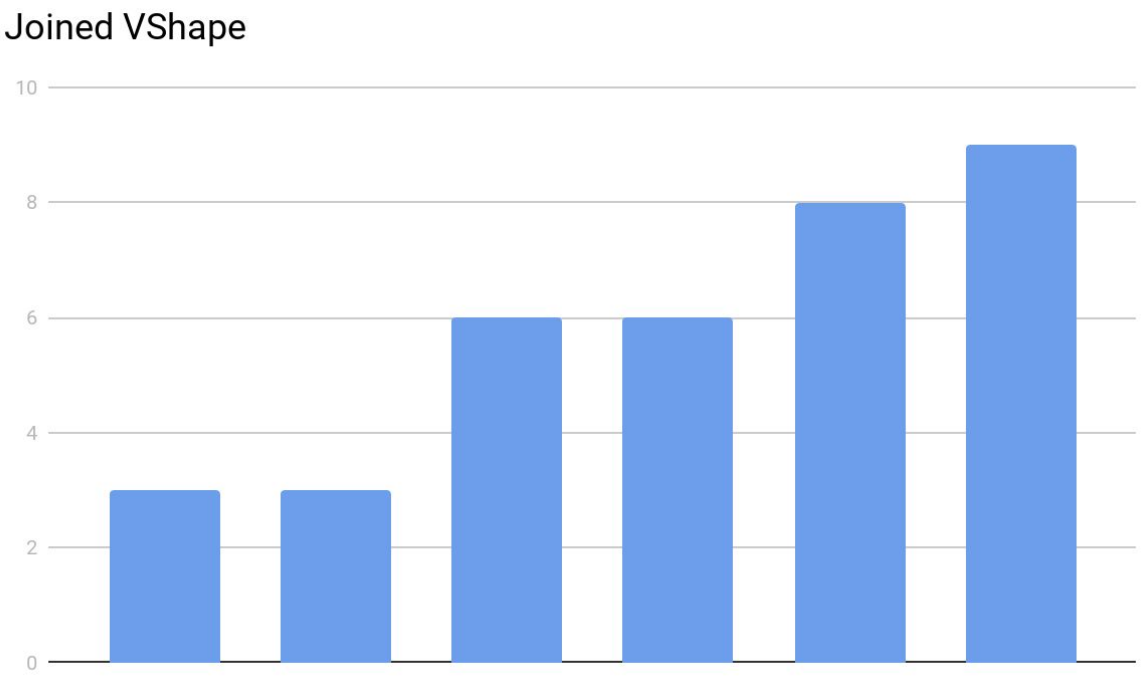


2. Fill the *VShapes* until they reach neighbor *VShape* height

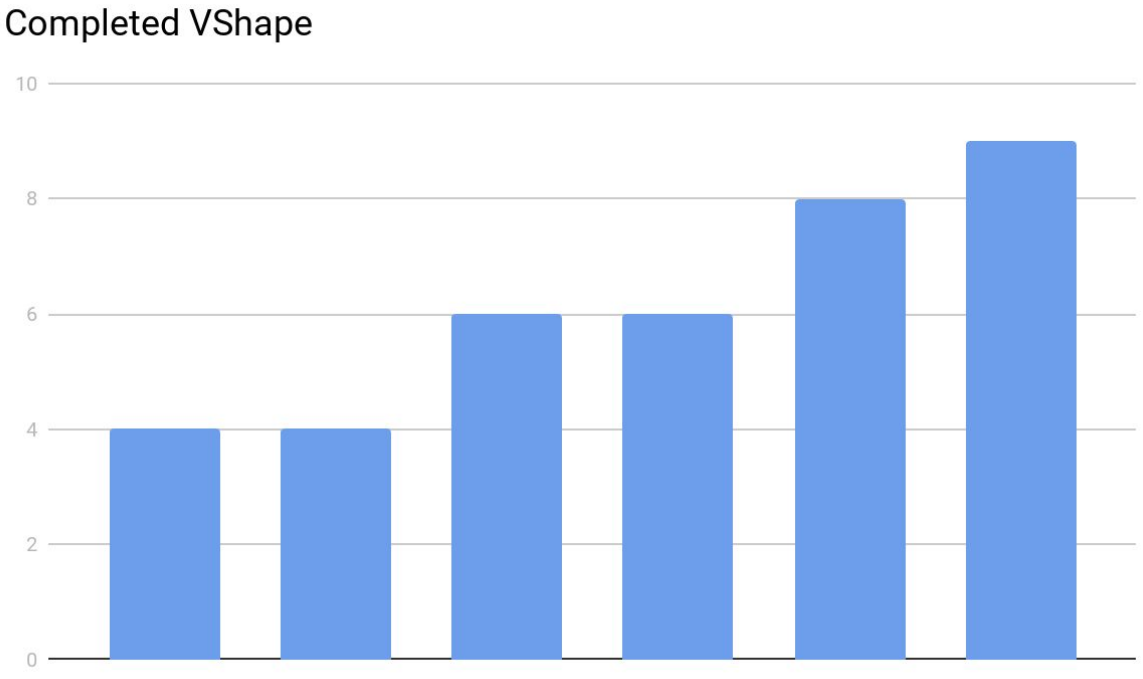




3. Join *VShapes into one*



4. Update the *VShape* with the reminding water



Solution considerations

I've not put too much effort into improving the performance of this algorithm, it's the better performant solution of the ones I've discovered. With more time it should be interesting to study worst-case performance scenarios, chose better data structures, and implement some easy paths.

The current implementation depends on native floating-point arithmetics *f64* and that affects precision with some landscapes.

Other solutions

One algorithm that could also solve this problem will be one like de following:

1. Iterate every segment
    a. If first time in this loop and segment, add one unit of rain to the segment
    b. Look at height differences between right and left neighbors, give them rain until they reach the same height as the segment or the segment is left without rain. Mark that you did changes
    c. If no difference in hight or no rain, mark that you did NO changes
2. If any of the segments changes, goto 1

This algorithm relays on that local rain distribution between neighbors would eventually lead to the distribution of rain across the whole landscape.

Although this implementation would work fine if units were not divisible it does not work as well when using floating-point numbers. This implementation relays a lot on the division operation in order to move units of water from one place to the other and it's quite imprecise when using *f64.*