

Calculation of rainfall on an arbitrary two-dimensional surface

Preface

This article describes an algorithm that calculates a rainfall in a finite two-dimensional surface defined by its heights.

Problem: Calculate a volume of water which remained after the "rain" on a 2D "surface" (in **units** - a *dimensionless quantity*).

Input: An array of integer numbers (describes profile's heights of a "surface").

Output: A number, further called "volume"

Terms

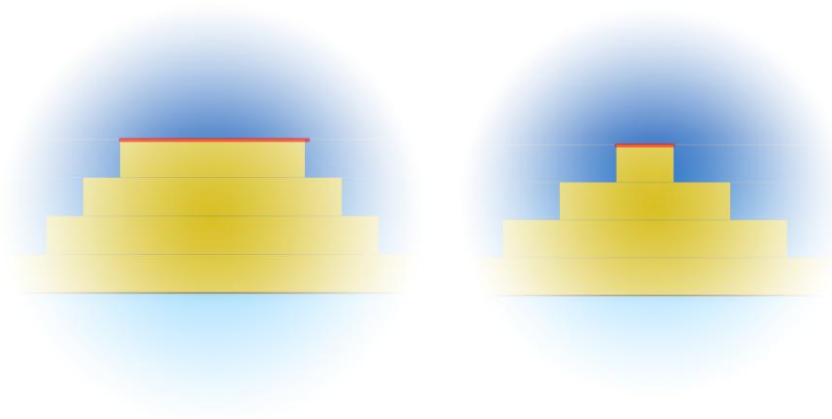
To solve this problem, we will use an object-oriented approach that gives us the opportunity to abstract from pure mathematics and describe the algorithm in terms and concepts that are accessible to understanding of a wide audience.

Plateau

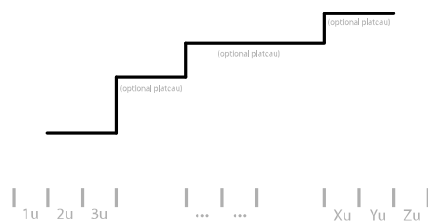


A continuous set of points (heights), that have same value. A plateau may consist of one single point as well.

Examples:

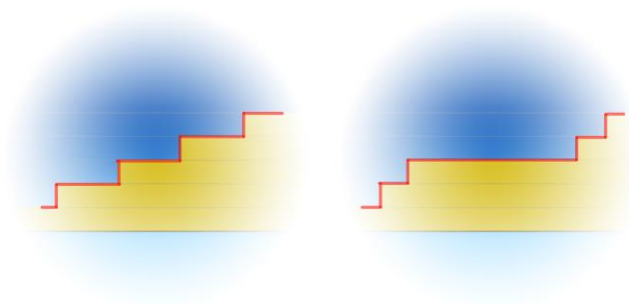


Ascent

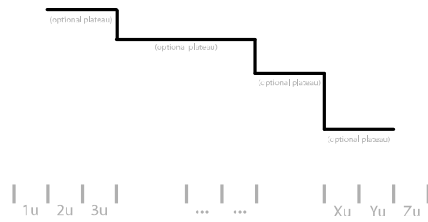


A set of points where each next point has value higher than the previous (may include one or more plateaus in the middle).

Examples:

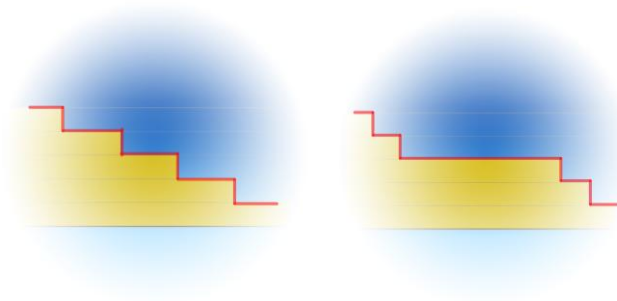


Descent

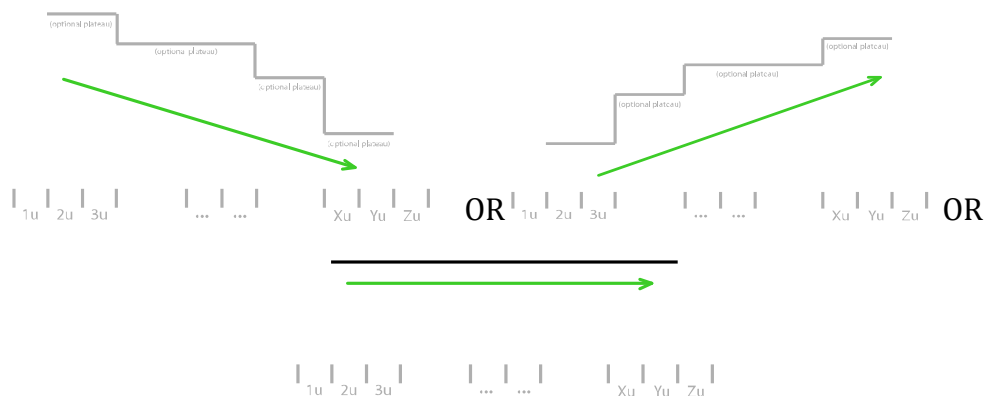


A set of points where each next point has value lower than the previous (may include one or more plateaus in the middle).

Examples:

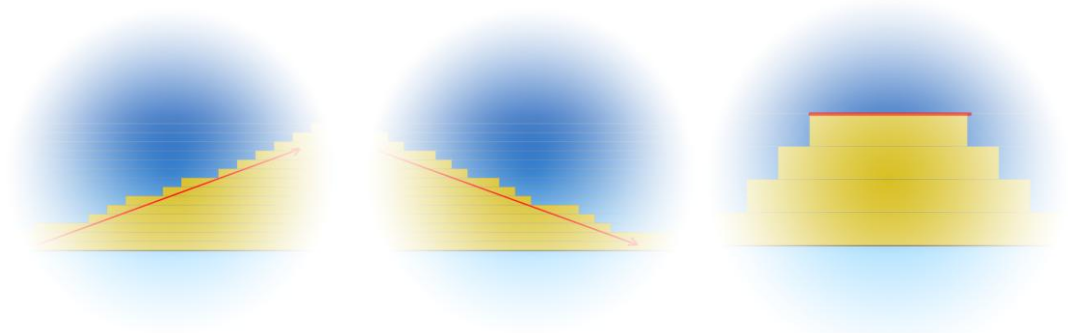


Trend



A set of constantly ascending values (and optionally plateaus in the middle) OR a set of constantly descending values (and optionally plateaus in the middle) OR a set of the same values that constitute a plateau.

Examples:

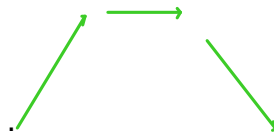


Peak

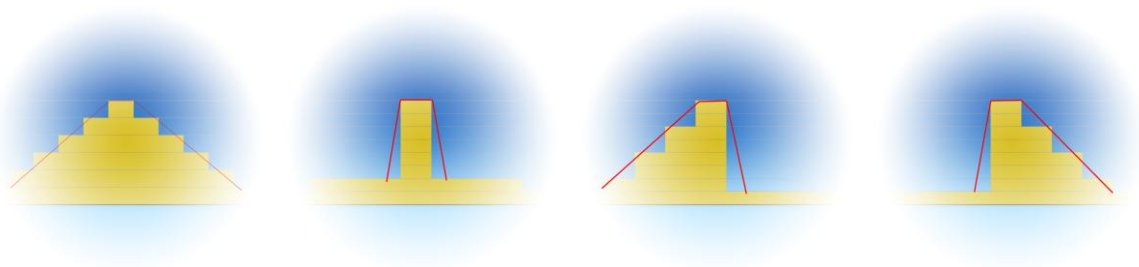


A highest point (a set of points, in the case of a plateau) in a sequence of the values which corresponds to a pattern: when **values are ascending, then form a plateau (one or more equal values), then descending.**

A peak pattern looks like this: 1) trend "UP", 2) trend "plateau" OR a single point, trend "DOWN"

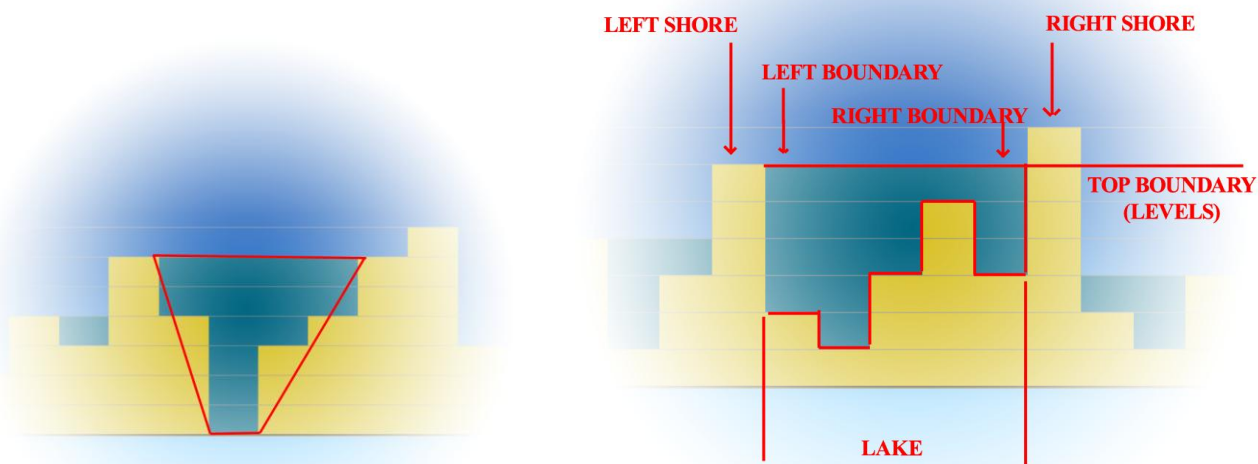


Examples:



Lake

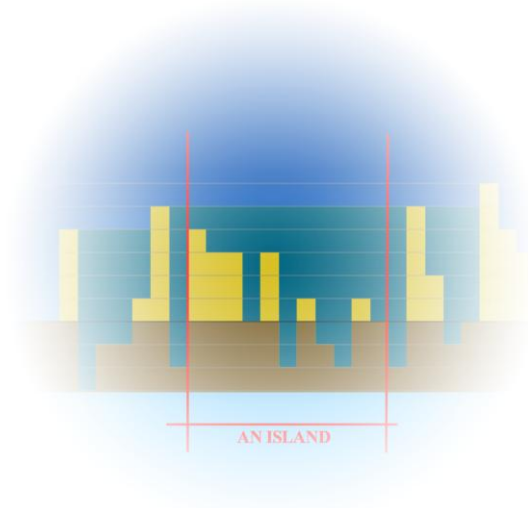
An area between a descent and an ascent, defined by its **left and right boundaries**, which's **top boundary (the level)** is limited by the lower of two peaks of that trends. **Any lake may exist only between exactly two peaks - left and right.**



Island

Island is an arbitrary subset of the peaks (one or more) from the all available peaks.

Example:



Algorithm Implementation

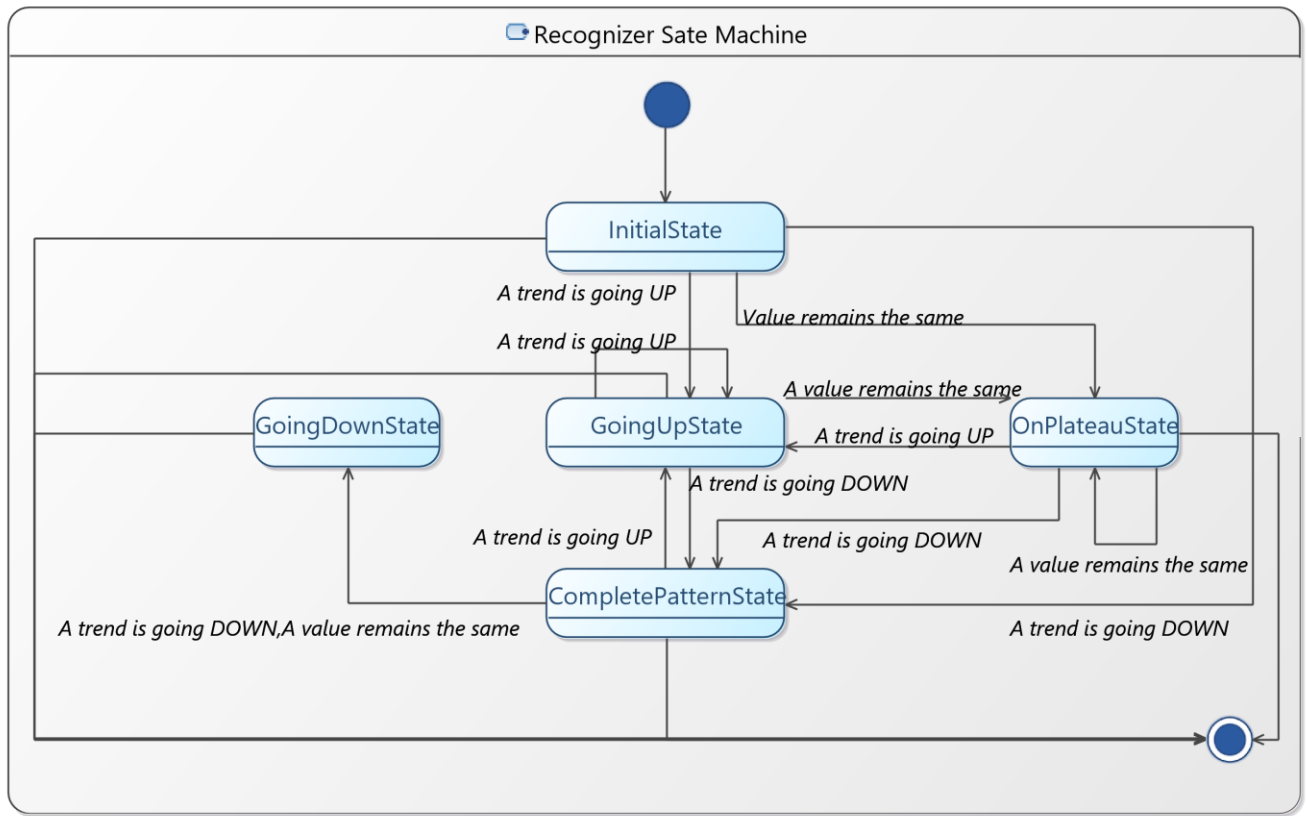
In the mentioned terms, in bird's-eye view, the algorithm may be described in the next steps:

1. **Step 1:** Find all "peaks" defined by the means of their heights (with the "trend up-plateau-trend down" pattern);
2. **Step 2:** Find all the "lakes" that exists among the revealed "peaks";
3. **Step 3:** Calculate the integral sum of volume of each such lake;

Step 1. Find all "peaks" defined by the means of their heights

To find the peaks we will reuse a state machine, that should recognize the up-plateau-down pattern.

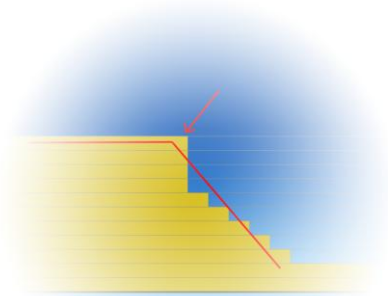
Peak recognizer state machine is following the values trends and collects the indexes of that values in the income heights array. Here is the map of state transitions:



Each state takes its place as a respond to the values trend changing event. Values become "run" in a different (than the current) direction - a new state is activated.

To understand the transitions, let's take a look at the few simple rules underlying these operations:

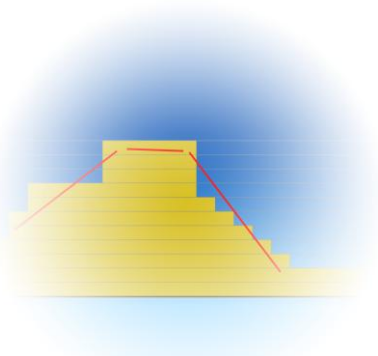
1. Actually, an **every new peak starts only when a new trend is going down after a plateau or a single highest value;**



2. At the same time - the mandatory prerequisite for 1) is that **there should be an ascent or an initial plateau before it**. What does it mean: a descent after a plateau (single point) which is also following after a descent - doesn't necessary marks a peak. See this exclusion example (a descent, after a plateau, after another descent - this is certainly not a peak):



3. This is where the **"Initial State"** comes to be useful - it guarantees that the 2) prerequisite is true at the very beginning. After that the pattern recognition is managed by the corresponding state - **up -(optional plateau) - down**.

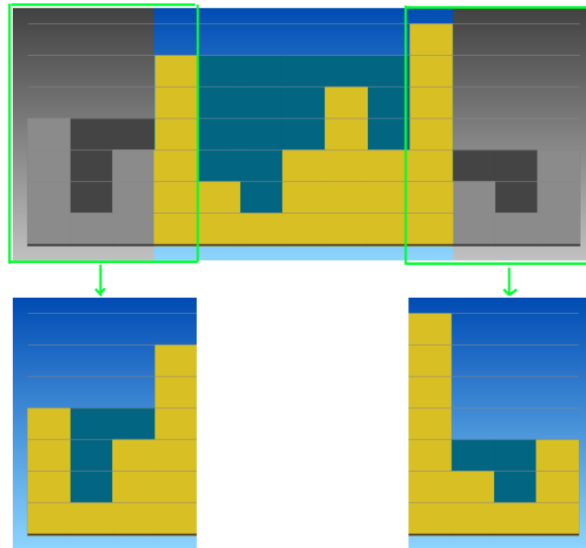


4. And so on - every state is collecting indexes of heights until their trends conforms to the "up-plateau-down" pattern. And after the **plateau-down** stage we may find the machine into the **"CompletePatternState"** which contains the plateau of the similar points (an array of indexes) or a single highest point (if there were no other points at the same level) - **an instance of the Peak object**.
5. Only the **"CompletePatternState"** provides a set of highest values indexes on a particular range of indexes. Being in "CompletePatternState" means that the "up - optional plateau - down" pattern was recognized.

Step 2. Find all the "lakes" that exists among the revealed "peaks"

This is the recursive iterative step, which's main operation are:

1. Find exactly two highest peaks - **L** (highest from the left) and **R** (highest from the right);
2. Flood the entire space between them (including other peaks that are less than the mentioned two) forming a lake;
3. Collect the lake to a **lakes' store**;
4. Break the current island on a two - leftmost peaks (including L), and rightmost peaks (including R);
5. Repeat from the **step 1**) for each such island (that contains more than one peak - a lake can exist only between at least two peaks)



Step 3: Calculate the integral volume of each lake

When all islands are processed, there is only step left - run through the lakes collection and sum all lakes volumes. **A lake volume is a difference among lakes level and lakes bottom at each index of the input data, over which the lake does exist.**