Last TPOP Practical Challenge

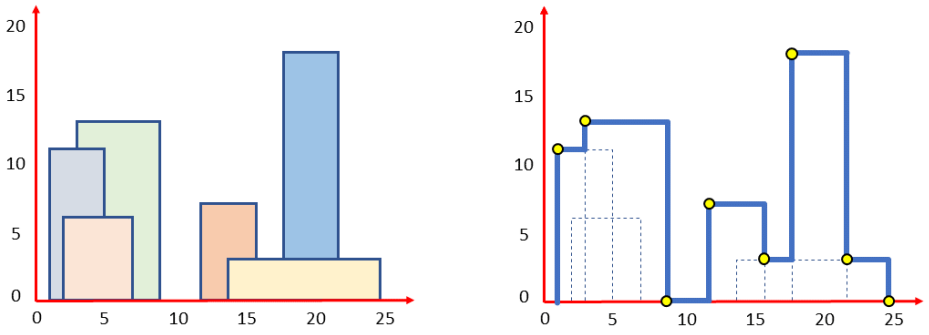
Lilian Blot

March 7, 2019

1 The Problem

We consider a problem related to the drawing of geometric figures. The problem is concerned with the removal of hidden lines, that is lines obscured by other parts of a drawing. This is similar to computing the projected shadows of geometric figures. We call this the upper envelop.

Given the exact locations and shapes of n bars in a 2-dimensional graph, give an algorithm that computes the projected shadow or upper envelop (in 2 dimensions) of these bars, eliminating hidden lines. As an example, an input is given in Figure 1a; the corresponding output is given in Figure 1b. We assume that the bottom of all bars lie on a fixed horizontal line ($H = 0$).



(a) (b)

Figure 1: The upper envelop Problem: (a) a set of bars, and (b) the resulting upper envelop.

A bar B_i is represented by the triple (L_i, H_i, R_i) where L_i and R_i denote the left and right x-coordinates of the bar respectively, and H_i denotes the bar's height. The following constraints must be respected:

- $0 \leq L_i < R_i$
- $H_i > 0$

The input is a list of triples; one per bar. The output is the upper envelop specified as a list of pairs (x_i, h_i) where x_i is the x-coordinates and h_i is the height of a node. The nodes are represented by yellow circles in Figure 1b. An example of inputs and outputs for the problem shown in Figure 1 are given in Figure 2.

Bars

L	H	R
1	11	5
2	6	7
3	13	9
12	7	16
14	3	25
19	18	22

(a)

Envelop

x	h
1	11
3	13
9	0
12	7
16	3
19	18
22	3
25	0

(b)

Figure 2: The upper envelop Problem: (a) the input, and (b) the output. Note the output is represented by yellow circle in Figure 1b.

2 Brute Force

The straightforward algorithm for this problem uses induction on n (number of bars). The base step is for $n = 1$ and the upper envelop can be obtained directly from B_1. As an induction step, we assume that we know S_{n-1} (the upper envelop for $n-1$ bars), and then must show how to add the n^{th} bar B_n to the upper envelop.

The process of adding a bar to the upper envelop can be examined by looking at Figure 3, where we add the bar $B_n = (L_n, H_n, R_n) = (6, 7, 24)$

to the upper envelop $(1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 25, 0)$. We scan the upper envelop from left to right stopping at the first x-coordinate x_1 that immediately precedes L_n (in this case $x_1 = 3$) and then we extract the part of the envelop overlapping with B_n as a set of strips $(x_1, h_1), (x_2, h_2), ..., (x_m, h_m)$ such that $x_m < R_n$ and $x_{m+1} \leq R_n$ (or x_m is last). In this set of strips, a strip will have its h_i replaced by H_n if $H_n > h_i$ (because the strip is now covered by B_n). If there is no x_{m+1} then we add an extra strip (replacing the last 0 in the old upper envelop) $(x_m, H_n, R_n, 0)$. Also, we check whether two adjacent pairs (x_k, h_k), (x_{k+1}, h_{k+1}) have the same height $(h_k = h_{k+1})$; if so, we remove the pair (x_{k+1}, h_{k+1}). This process can be viewed as a merging of B_n and S_{n-1}.

In the worst case, this algorithm would need to examine all the $n-1$ triples when adding B_n (this is certainly the case if B_n is so wide that it encompasses all other triples). Likewise, adding B_{n-1} would need to examine $n-2$ triples, and so on. Therefore the algorithm is $O(n) + O(n-1) + ... + O(1) = O(n^2)$.

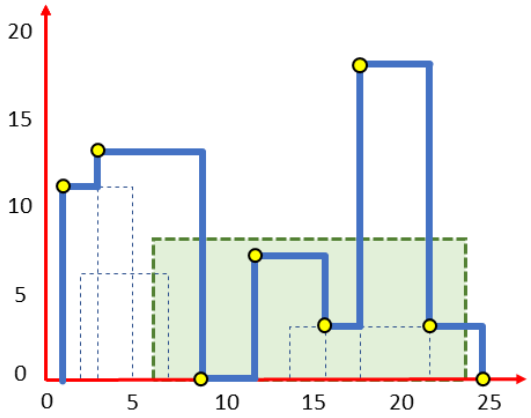


Figure 3: Adding a single bar to a upper envelop.

3 Implementation

You should design a Java class hierarchy that allows the implementation of the Upper Envelop problem using different algorithm (in the same way `List` interface as several implementation of a list (namely `ArrayList` and `LinkedList`). In this practical, we only consider two implementations: Brute

force, and Divide and Conquer. However, your design should allow a developer to implement additional algorithm by inheriting a class(es) and/or an interface(s) from your hierarchy.

- what are the interfaces, abstract classes, and concrete classes
- What are the attribute(s), constructors, methods for each class?
- Ensure the constraints given in the problem are enforced.
- The concrete classes should be able to :
  - Add a single bar to an upper envelop.
  - Add a `Collection` of bar to an upper envelop.
  - Merge two envelops
  - The design should provide a method that return the envelop as a `List` of pairs (x_i, h_i).
  - The design should provide a method to return a diagram representation of an upper envelop via a `String` object as shown in Figure 4.

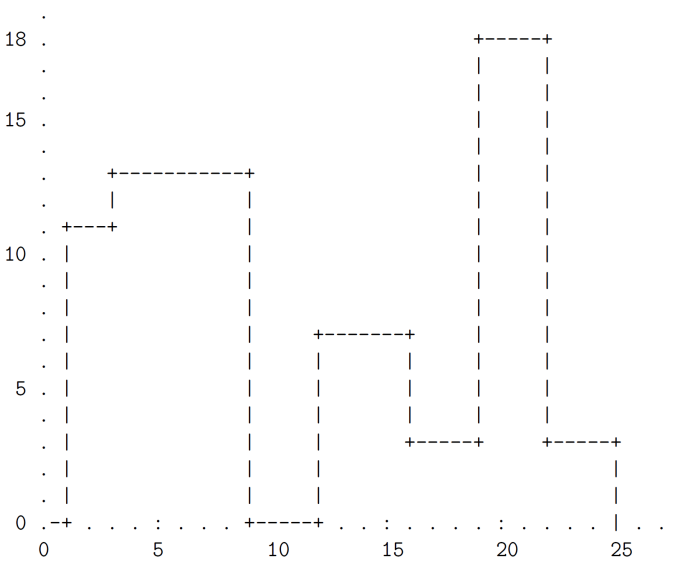


Figure 4: Diagram representation of an upper envelop from Figure 1 using a *String*.

4 A Divide-and-Conquer Algorithm

Is merging two upper envelops substantially different from merging a bar with a upper envelop? The answer is, of course, No.

This suggests that we use divide-and-conquer.

4.1 General idea

Divide the input of n bars into two equal (or nearly equal) sets. Compute (recursively) the upper envelop for each set then merge the two upper envelops. The algorithm has a structure similar to Mergesort. Unlike Mergesort, the input and output are of different data types. In Mergesort, the input and output are arrays of some base type. However, that does not matter much. We simply need to merge two upper envelops (and not two sets of bars).

For instance, given two envelops $A = ((a_1, ha_1), (a_2, ha_2), ..., (a_n, 0))$ and $B = ((b_1, hb_1), (b_2, hb_2), ..., (b_m, 0))$, we merge these lists as the new list: $((c_1, hc_1), (c_2, hc_2), ..., (c_k, 0))$ with $k \leq m + n$. Clearly, we merge the list of as and bs just like in the standard Merge algorithm. But, in addition to that, we have to decide on the correct height in between these boundary values.

Similarly to the Mergesort algorithm, a properly implemented divide-and-conquer solution to the upper envelop problem as a $O(n\ log(n))$ complexity.

4.2 To Do

Devise the divide and conquer algorithm, and then implement it in a class that fits in the hierarchy you designed in Section 3. To create your algorithm, you may want to use the hints provided in Section 4.3. Finally, you may want to test manually your algorithm against the example provided in Section 5.

4.3 Hints

Here is an example of how the divide and conquer algorithm works. Given the two inputs:

- $E_1 = ((1, 11), (3, 13), (9, 0), (12, 7), (16, 0))$
- $E_2 = ((14, 3), (19, 18), (22, 0))$

let's compute the output of merging envelops E_1 and E_2. Height of new Strip is always obtained by taking the maximum of the following

(a) Current height from E_1, say h_1.

(b) Current height from E_2, say h_2.

h_1 and h_2 are initialised to 0. *Result* is initialised to an empty list. h_1 is updated when a strip from E_1 is added to result and h_2 is updated when a strip from E_2 is added.

- Compare $(1, 11)$ and $(14, 3)$. Since first strip has smaller left x, add it to result and increment index for E_1.
  $h_1 = 11$, $Result = ((1, 11))$
- Compare $(3, 13)$ and $(14, 3)$. Since first strip has smaller left x, add it to result and increment index for upper E_1.
  $h_1 = 13$, $Result = ((1, 11), (3, 13))$
- Similarly $(9, 0)$ and $(12, 7)$ are added.
  $h_1 = 7$, $Result = ((1, 11), (3, 13), (9, 0), (12, 7))$
- Compare $(16, 0)$ and $(14, 3)$. Since second strip has smaller left x, it is added to result.
  $h_2 = 3$ $Result = ((1, 11), (3, 13), (9, 0), (12, 7), (14, 7))$
- Compare $(16, 0)$ and $(19, 18)$. Since first strip has smaller left x, it is added to result.
  $h_1 = 0$,
  $Result = (1, 11), (3, 13), (9, 0), (12, 7), (14, 3), (16, 3)$
- Since E_1 has no more items, all remaining items of E_2 are added.
  $Result = ((1, 11), (3, 13), (9, 0), (12, 7), (14, 3), (16, 3), (19, 18), (22, 0))$

One observation about above output is, the strip (16, 3) is redundant (There is already an strip of same height). We remove all redundant strips and $Result = ((1, 11), (3, 13), (9, 0), (12, 7), (14, 3), (19, 18), (22, 0))$.

5 Test Samples

This section provides two tables with example of building an upper envelop from a list of bars 1, and merging two upper envelops 2. You should check your algorithm and implementation against these samples.

Table 1: Sample tests for building an upper envelop from a list of bars.

Bars	Envelop
(1,3,4),(2,4,5),(6,1,7),(8,5,11), (8,4,15),(9,3,12),(13,5,16)	((1,3),(2,4),(5,0),(6,1),(7,0), (8,5),(11,4),(13,5),(16,0))
(1,2,9),(13,9,16),(20,7,21),(1,3,3), (10,9,13),(15,2,19),(22,7,23)	((1,3),(3,2),(9,0),(10,9),(16,2), (19,0),(20,7),(21,0),(22,7),(23,0))
(1,11,5),(2,6,7),(3,13,9),(12,7,16), (14,3,25),(19,18,22),(23,13,29), (24,4,28)	((1,11),(3,13),(9,0),(12,7),(16,3), (19,18),(22,3),(23,13),(29,0))

Table 2: Sample tests for merging two upper envelops E_1 and E_2.

E_1	E_2	Merged
((2,5),(6,4),(9,10), (13,0))	((3,12),(12,5),(14,0))	((2,5),(3,12),(12,10), (13,5),(14,0))
((1,3),(3,0),(10,9), (13,0),(15,2),(19,0), (22,7),(23,0))	((1,2),(9,0),(13,9), (16,0),(20,7),(21,0))	((1,3),(3,2),(9,0), (10,9),(16,2),(19,0), (20,7),(21,0),(22,7), (23,0))
((1,11),(3,13),(9,0), (12,7),(16,0))	((14,3),(19,18),(22,3), (23,13),(29,0))	((1,11),(3,13),(9,0), (12,7),(16,3),(19,18), (22,3),(23,13),(29,0))