Skyline

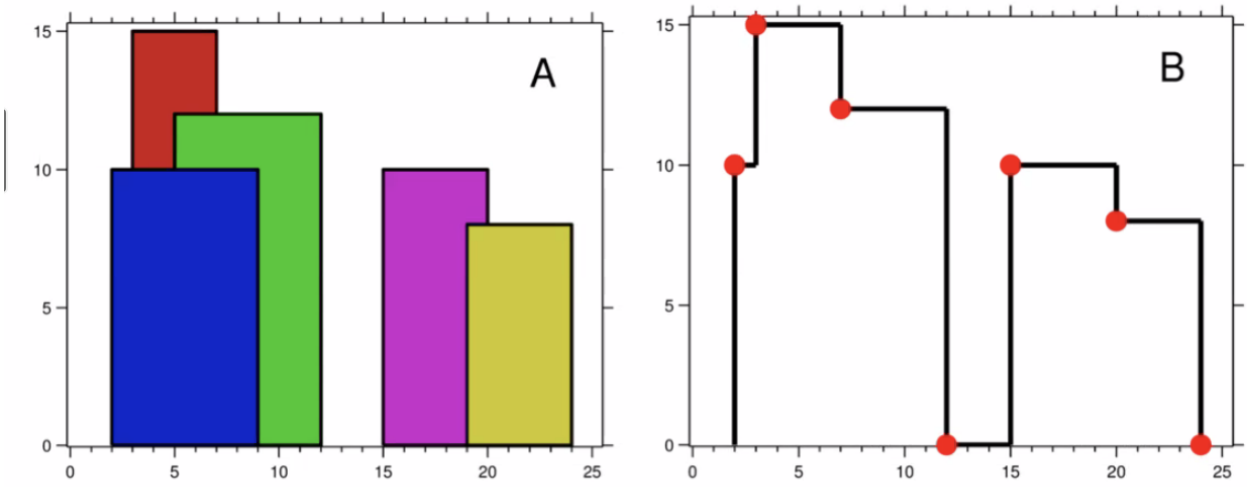
Mapping Points in 2d Space

Mason Heaman

1. Introduction:

The Skyline algorithm was designed to assist an architect in drawing a city skyline. I have designed an algorithm that, given an ordered triple denoting building location, will compute coordinates mapping the skyline view of a city. For a visualization of this algorithm see Figure 1.1.

Figure 1.1



Section A represents the given input. Shapes are drawn using the ordered triple (L, H, R), providing the left x coordinate, height, and right x coordinate respectively. Section B represents the calculated coordinates for the skyline of A.

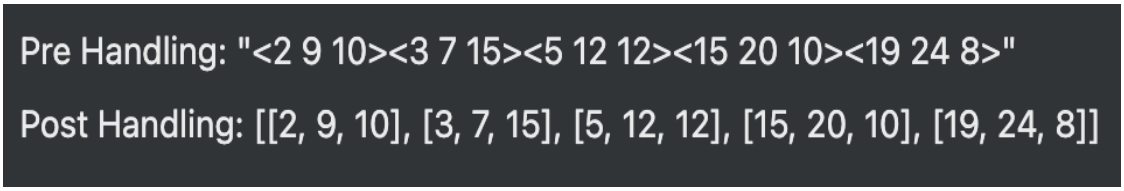
2. Plausibility, Efficiency, and Resources

This algorithm will be made possible using a Divide and Conquer approach similar to that of mergesort. The list of building points will be recursively split into individual buildings, which will then be merged using several different sub-cases (See Section 3). This divide and conquer approach will yield a Time Complexity of T(nlogn), once again similar to mergesort. In terms of resources, the implementation and running of this algorithm can be handled by a single developer with a single workstation. Given the implementation instructions in Section 3 and the availability of a supervisor for advice, a single developer should be able to solve the problem in 1-3 working days.

3.1 Implementation

The first step of the implementation lies in handling the input file. The input will be given in an ordered triple, where values are separated by spaces and buildings are enclosed by <>'s. A helper function should be implemented to read this data from a file and return it as a two-dimensional integer array. Data before and after handling are shown in figure 3.1.

Figure 3.1



After data has been correctly converted to a two-dimensional integer array, The main Skyline function can be created. This function will take three arguments: the beginning index of the int[][] (int); the ending index of the int[][](int); and the int[][] itself. This function is the recurrence that drives the skyline merging and will handle three cases. Before implementing these cases, create an ArrayList to hold the solution.

Case 1: low is greater than high

This means that all of the skylines have been merged, return the solution.

Case 2: low is equal to high

This means that we are on the last building in the list, add two elements to the solution (l, h) and (r, 0) to finish the skyline at ground level.

Case 3: low is less than high

This means that there is more than one building to be handled and we can use a divide and conquer approach to split and merge the buildings efficiently. Find the midpoint of the int[][] and recursively call the Skyline method twice: from low-mid, and mid+1-high. Then call the Merge function. See section 3.2 for merging instructions.

3.2 Merging

The merging function will similarly be handled using 3 cases. To prepare for the merge, initialize two integer variables: h1 and h2 to keep track of height values. Initialize them to 0. Additionally, get the first buildings from the higher and lower lists and create an int[x,y] to hold the new building coordinates.

Case 1: the low building's l value is less than the high building's

Set the new building's values to the low building's l and h values. If the low building's h is less than h2, update the new building's y value to h2. Set h1 to the h value of the low building.

Case 2: the high building's l value is less than the low building's

Set the new building's values to the high building's l and h values. If the high building's h is less than h1, update the new building's y value to h1. Set h2 to the h value of the low building.

Case 3: the building's l values are equal

Set the new building's x value to the shared x value of the low and high buildings. Set the new building's y value to the calculated maximum of the two building's y values.

After the new building's coordinates have been filled by one of the three cases, add the new building to the solution. Loop through the low and high skylines until one of the arrays becomes empty, then fill the remaining buildings from the non-empty array.

3.3 Removing Duplicates

While still included in the merge function, removing duplicates is an important process that benefits from a deeper explanation. Duplicates are points that have the same h value as their predecessor, meaning that they do not add to the skyline.

To find duplicates, loop through the elements added to the solution. If the h value is the same as the previous value, remove that point from the solution and continue to the next point.

If a duplicate value is not found at the current point, break from the loop as a duplicate value will only be directly after the point being checked.

3.3 Returning the Solution

After duplicates have been removed, the skyline has been effectively merged and can be returned. The merging function along with the driving Skyline function will result in a fully solved Skyline from the input values.