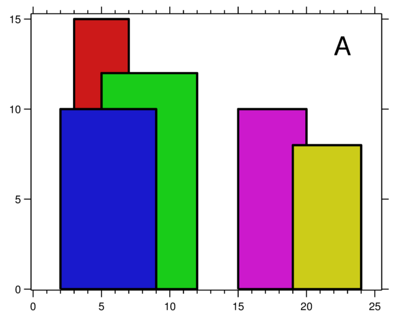
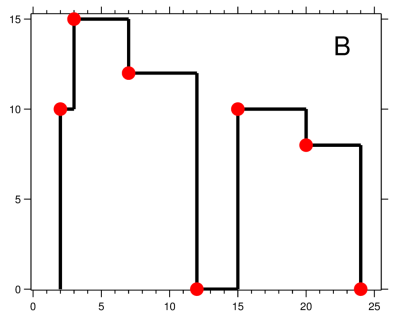
CS29003 ALGORITHMS LABORATORY
(Assignment 2: Divide and Conquer)
Date: Sep 17 – September – 2020

Problem Statement

You have joined as a trainee in an Institution researching on creating devices to help people with defective eyesight. You have been assigned to a group which deals with the binary eyesight defect, where a person can only see 2 colors, black and white. This means that when 2 objects have a greater contrast than a particular threshold, then they can be perceived as different.
They are at the initial phases of the project and they want to understand and stress on the difficulties faced by these people in recognizing objects and the loss of depth perception.
Since you are still a trainee you have been tasked to get the perspective of users on a set of rectangular objects kept on a table.
Imagine that you are looking right at it, and you cannot see the top of these objects (this simulates the irrelevance of depth information). The objects are of various colours and the background is white, and it is known that the defective eye will not be able to differentiate between each of the objects but there is enough contrast to differentiate between the objects and the background.
Using the divide and conquer technique, devise an algorithm to convert the input of a normal vision (Figure 1(a)) to that of the defective vision (Figure 1(b)), and implement it in C++/C .



(a) objects appearing to normal vision



(b) objects appearing to defective vision

Figure 1: Images as seen by the 2 visions

Note: Figure 1(b) does not accurately represent the image perceived by the defective eye. The silhouette would be completely filled in that case.

Input

The number of rectangular boxes N. The geometric information of each rectangular box is represented by a triplet of integers $[L_i, R_i, H_i]$, where L_i and R_i are the x coordinates of the left and right edge of the i^{th} box, respectively, and H_i is its height. It is guaranteed that:

$$0 \leqslant L_i, R_i \leqslant \text{INT\_MAX}$$

$$0 < H_i \leqslant \text{INT\_MAX}$$

$$R_i - L_i > 0$$

You may assume all boxes are perfect rectangles grounded on an absolutely flat surface at height 0.

Output

The output is a list of “key points” (red dots in Figure 1(b)) in the format of

$$[[x_1, y_1], [x_2, y_2], [x_3, y_3], ...]$$

A key point is the left endpoint of a horizontal line segment.
Note that the last key point, where the rightmost box ends, is merely used to mark the termination of the outline, and always has zero height. Also, the ground in between any two adjacent boxes should be considered part of the outline contour.

Naïve Algorithm

This will be an incremental approach, where we will build the outline by merging it with every box in the order which is given. (Remember that the input is not sorted in any order)
Here, the inputs are stored in $boxes$ array described in detail in the Input section above.

```
typedef struct {
   int left; // x coordinate of left side
   int ht; // height
   int right; // x coordinate of right side
} Box;
typedef struct { // A Point in Outline
   int x; // x coordinate
   int y; // height or the y coordinate
} Point;
boxes is the input array of Box datatype
outline = [] // list of Point datatype, so that deletion and insertion costs O(1)
for b_i in boxes: // O(N)
    find j in outline with b_i.left < outline_j.x // O(N)
    //need to combine this point with the outline
    if outline_{j-1}.y>=b_i.ht: // if the height of the previous point is higher than this
        no need to make any changes
    else
        if outline_{j-1}.x==b_i.left: // if the prev point has the same position
            outline_{j-1}.y = b_i.ht // we know that this point has higher height
        else
            insert a new point
            j++

    for points in outline from index j to k where b_i.right<outline_k.x // O(N)
        do remove points from the outline where the height falls below that of b_i
    //in the way we added the left point make a similar combination on the right point
         (b_i.right, 0)
end
```

Time complexity: $O(N^2)$

Can you use Divide and Conquer to devise an algorithm with a complexity of $O(N \log(N))$?

Sample Test Cases

Test Case 1

```
5
2  9  10
3  7  15
5  12  12
15  20  10
19  24  8

Outline :
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

Test Case 2

```
7
0  5  7
5  10  7
5  10  12
10  15  7
15  20  7
15  20  12
20  25  7

Outline :
[[0,7],[5,12],[10,7],[15,12],[20,7],[25,0]]
```

Test Case 3

```
2
0  2  3
2  5  3

Outline :
[[0,3],[5,0]]
```

Test Case 4

```
3
1  2  1
1  2  2
1  2  3

Outline :
[[1,3],[2,0]]
```

Implementation Instructions

We will be providing a header file, which you need to include in your program (C/C++).
All the input and output technicalities have been handled in the process() function of the header file.
You are free to define structures and classes of your own but make sure to keep the end points compatible.
You sample submission should be as follows

```
// 18CS30004\_G03\_Assign2.c/cpp
#include "assign2.h"
Point* findOutLine(Box boxes[], int size, int& outputsize){
    Point* point_array;
    // here solve the problem, where boxes contain the input data and size is the number of
        boxes on the table
    // set outputsize to the number of points in Point array you are yet to return
    return point_array;
}
int main(){
    process();
    return 0;
}
```

Note : Do not include any other header files.

File Naming Convention

Please note that your submissions will not be evaluated unless you follow the below specified file naming convention for the program file. <ROLLNO(IN CAPS)>_Assign<Assign_No>_G<Group_No>.c/cpp

Eg: 18CS30004_G03_Assign2.c / 18CS30004_G03_Assign2.cpp