Problem 218: The Skyline Problem

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

A city's

skyline

is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return

the

skyline

formed by these buildings collectively

.

The geometric information of each building is given in the array

buildings

where

buildings[i] = [left

i

, right

i

, height

i

]

:

left

i

is the x coordinate of the left edge of the

i

th

building.

right

i

is the x coordinate of the right edge of the

i

th

building.

height

i

is the height of the

i

th

building.

You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height

0

.

The

skyline

should be represented as a list of "key points"

sorted by their x-coordinate

in the form

[[x

1

,y

1

],[x

2

,y

2

],...]

. Each key point is the left endpoint of some horizontal segment in the skyline except the last point in the list, which always has a y-coordinate

0

and is used to mark the skyline's termination where the rightmost building ends. Any ground between the leftmost and rightmost buildings should be part of the skyline's contour.

Note:

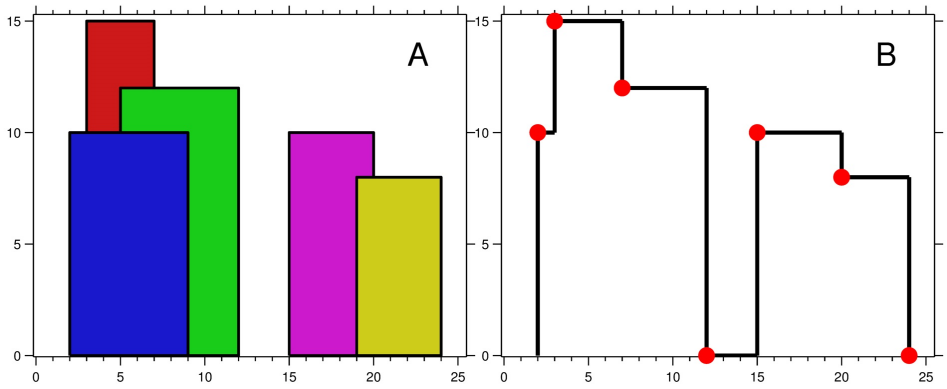
There must be no consecutive horizontal lines of equal height in the output skyline. For instance,

[...,[2 3],[4 5],[7 5],[11 5],[12 7],...]

is not acceptable; the three lines of height 5 should be merged into one in the final output as such:

[...,[2 3],[4 5],[12 7],...]

Example 1:



Input:

buildings = [[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]

Output:

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

Explanation:

Figure A shows the buildings of the input. Figure B shows the skyline formed by those buildings. The red points in figure B represent the key points in the output list.

Example 2:

Input:

buildings = [[0,2,3],[2,5,3]]

Output:

[[0,3],[5,0]]

Constraints:

1 <= buildings.length <= 10

4

0 <= left

i

< right

i

<= 2

31

- 1

1 <= height

i

<= 2

31

- 1

buildings

is sorted by

left

i

in non-decreasing order.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> getSkyline(int[][] buildings) {

}
}
```

Python3:

```
class Solution:
def getSkyline(self, buildings: List[List[int]]) -> List[List[int]]:
```

Python:

```
class Solution(object):
def getSkyline(self, buildings):
"""
:type buildings: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} buildings
* @return {number[][]}
*/
var getSkyline = function(buildings) {

};
```

TypeScript:

```
function getSkyline(buildings: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> GetSkyline(int[][] buildings) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
```

```
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** getSkyline(int** buildings, int buildingsSize, int* buildingsColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func getSkyline(buildings [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun getSkyline(buildings: Array<IntArray>): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func getSkyline(_ buildings: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn get_skyline(buildings: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} buildings
# @return {Integer[][]}
```

```
def get_skyline(buildings)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $buildings
* @return Integer[][]
*/
function getSkyline($buildings) {

}
}
```

Dart:

```
class Solution {
List<List<int>> getSkyline(List<List<int>> buildings) {

}
}
```

Scala:

```
object Solution {
def getSkyline(buildings: Array[Array[Int]]): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_skyline(buildings :: [[integer]]) :: [[integer]]
def get_skyline(buildings) do

end
end
```

Erlang:

```
-spec get_skyline(Buildings :: [[integer()]]) -> [[integer()]].
get_skyline(Buildings) ->
.
```

Racket:

```
(define/contract (get-skyline buildings)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Skyline Problem
* Difficulty: Hard
* Tags: array, tree, dp, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

}
};
```

Java Solution:

```
/**
* Problem: The Skyline Problem
* Difficulty: Hard
* Tags: array, tree, dp, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public List<List<Integer>> getSkyline(int[][] buildings) {

}
}
```

Python3 Solution:

```
"""
Problem: The Skyline Problem
Difficulty: Hard
Tags: array, tree, dp, sort, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def getSkyline(self, buildings: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getSkyline(self, buildings):
"""
:type buildings: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: The Skyline Problem
* Difficulty: Hard
```

```
* Tags: array, tree, dp, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} buildings
* @return {number[][]}
*/
var getSkyline = function(buildings) {

};
```

TypeScript Solution:

```
/**
* Problem: The Skyline Problem
* Difficulty: Hard
* Tags: array, tree, dp, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function getSkyline(buildings: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: The Skyline Problem
* Difficulty: Hard
* Tags: array, tree, dp, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

public class Solution {
public IList<IList<int>> GetSkyline(int[][] buildings) {

}
}
```

C Solution:

```
/*
* Problem: The Skyline Problem
* Difficulty: Hard
* Tags: array, tree, dp, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** getSkyline(int** buildings, int buildingsSize, int* buildingsColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: The Skyline Problem
// Difficulty: Hard
// Tags: array, tree, dp, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

```
func getSkyline(buildings [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun getSkyline(buildings: Array<IntArray>): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func getSkyline(_ buildings: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: The Skyline Problem
// Difficulty: Hard
// Tags: array, tree, dp, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn get_skyline(buildings: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} buildings
# @return {Integer[][]}
def get_skyline(buildings)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $buildings
* @return Integer[][]
*/
function getSkyline($buildings) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> getSkyline(List<List<int>> buildings) {

}
}
```

Scala Solution:

```
object Solution {
def getSkyline(buildings: Array[Array[Int]]): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_skyline(buildings :: [[integer]]) :: [[integer]]
def get_skyline(buildings) do

end
end
```

Erlang Solution:

```
-spec get_skyline(Buildings :: [[integer()]]) -> [[integer()]].
get_skyline(Buildings) ->
.
```

Racket Solution:

```
(define/contract (get-skyline buildings)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```