Tutorial 4: Greedy algorithms

1 Computing the Skyline of a City

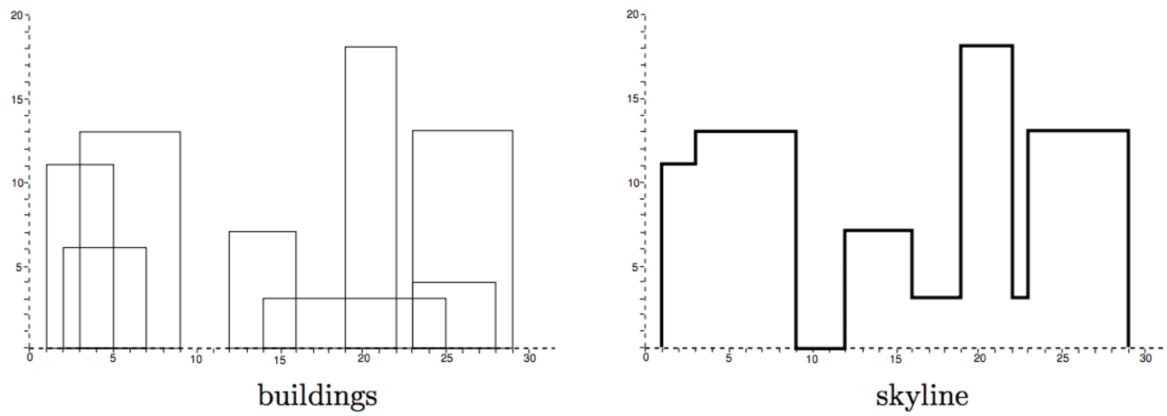

Figure 1: Left and right figures are visual representations of the input and skyline, respectively.

You are given the locations and shapes of a set of rectangular buildings in a city, and you wish to draw the skyline of these buildings in two dimensions. The bottoms of all the buildings lie on the x-axis. Building B_i is represented by a triple (L_i, R_i, H_i), where L_i and R_i denote the left and right x-coordinates of B_i, respectively, and H_i denotes B_i's height. Informally, the skyline of the city is the silhouette of the buildings. Formally, a skyline is defined as follows. For each x coordinate value, take the height of the highest building that intersects x. This draws the skyline and can be represented as a sequence of maximal line segments, $(x_1, x_2, h_1), (x_2, x_3, h_2), (x_3, x_4, h3), ..., (x_{k-1}, x_k, h_k)$, where x_i's are ordered. Since the end point of each line segment x_i is the same as the start point of the next line segment, we can simply represent the skyline as a set of $(x_1, h_1), (x_2, h_2), ..., (x_k, h_k)$ pairs. Consider as an example, the buildings in the left figure left, which can be written as (sorted by their L_i values):

$$Input : (1, 5, 11), (2, 7, 6), (3, 9, 13), (12, 16, 7), (14, 25, 3), (19, 22, 18), (23, 29, 13), (23, 28, 4)$$

The skyline of this building is shown on the right and can be written as:

$$Output : (1, 11), (3, 13), (9, 0), (12, 7), (16, 3), (19, 18), (22, 3), (23, 13)(28, 0)$$

You are given as input a set of n B_i that are **unordered**. You can assume for simplicity that the L_i and R_i points of the buildings are distinct. Design an $O(n \log n)$ divide and conquer algorithm that computes the skyline of the city.

Hint: Consider splitting the input arbitrarily to left and right sets of $n/2$ buildings and think of a linear time merge algorithm to merge the two skylines.

1.1 Solution

Given two skylines S_1 and S_2, already sorted on their x values, we can merge them as follows:

Algorithm 1: Skyline($B_1, ..., B_n$)

```
if n == 1 then
    return (B_1.L, B_1.h) (B_1.R, 0)
S_1 = Skyline(B_1, ..., B_{n/2});
S_2 = Skyline(B_{n/2+1}, ..., B_n);
return MergeSkylines(S_1, S_2)
```

Algorithm 2: MergeSkylines(S_1, S_2 of length $n/2$)

```
// i_1, i_2 are the left most pointers and h_i, h_j are the current heights in S_1, S_2, respectively.
i_1 = 1, i_2 = 1, h_i1, h_i2 = 0;
curH = 0;
curLoc = 0 out = {};
while i_1 ≤ n/2 & i_2 ≤ n/2 do
    if S_1[i_1].x < S_2[i_2].x then
        curLoc = S_1[i_1].x;
        h_i1 = S_1[i_1].h;
        i_1++;
    else if S_2[i_2].x < S_1[i_i].x then
        curLoc = S_2[i_2].x;
        h_i2 = S_2[i_2].h;
        i_2++;
    newH = max(h_i1, h_i2);
    if newH ≠ curH then
        out.add(curLoc, newH);
        curH = newH
```

We move in tandem from left to right in both S_1 and S_2 and at any point keep the highest point in $curH$. When the highest point changes, we add a new point to the skyline. The runtime of this merge subroutine is $O(n)$, giving us a recurrence $T(n) = 2T(n/2) + O(n)$, which is $O(n \log n)$ by the Master theorem.

2 Buying items from the SuperCheapStore

Suppose we would like to buy n items from the SuperCheapStore (SCS) where all items are currently priced at 1\$. Unfortunately, there is no delivery and we have to transport the items home. And to make matters worse:

- we can fit only one item in our truck; and
- it takes one day to drive home and back.

Worst of all, SCS charges us for the storage of undelivered items and the charge for storage of item i grows exponentially as the original price times a factor $c_i > 1$ each day. This means that if item i is picked up d days from now, the charge will be c_i^d dollars. In which order should we pick up our items from SCS so that total amount of charges is as small as possible?

Develop a greedy algorithm to solve this problem assuming that $c_i \neq c_j$ for $i \neq j$. Prove that your algorithm gives an optimal solution. What is the running time of your algorithm?

2.1 Solution

Sort the items in decreasing order of c_i. Pick them up in this order. We claim that the total cost will be minimized. Suppose (by re-indexing) that $c_1 > c_2 > \cdots > c_n$. Then we pick up item i on day i which costs c_i^{i-1}. Item c_1 is picked up on the first trip, so it costs $1 = c_1^0$. The total cost is $c_1^0 + c_2^1 + c_3^2 + \cdots + c_n^{n-1}$.

Consider any different solution S. We will show that its cost can be decreased. Since solution S is different there must be two items i and j where $c_i < c_j$ but we pick up item i before item j. Suppose we pick up item i after d days and item j after $d+k$ days. The cost of these two items in solution S is $c_i^d + c_j^{d+k}$.

Consider a new solution S' where we swap these two items. The cost of these two items in solution S' is $c_i^{d+k} + c_j^d$. The costs for all the other items remain the same. We want to prove that S' costs less, i.e. that

$$c_i^{d+k} + c_j^d < c_i^d + c_j^{d+k}.$$

Rearranging, we want to prove:

$$c_i^d(c_i^k - 1) < c_j^d(c_j^k - 1)$$

This is clear because $c_i < c_j$. Since we can decrease the cost of any solution different from the greedy one, therefore the greedy solution minimizes the cost. The time complexity of this algorithm is $\Theta(n \log n)$ to sort.