Handin 6

Thomas Vinther & Jens Kristian Refsgaard Nielsen

25-10-18

1 Skyline

We wish to make an algorithm that produces the silhouette of a city. Each building in the city is represented by a tripple (l,h,r) where l (r) is the coordinate on the horizontal line where the building begins (ends) and h is the height of the building.

The sihouette of the city is a vector $(x_0, h_1, x_1, \ldots, x_{i-1}.h_i, x_i, \ldots, h_n, x_n)$ where $x_j < x_{j+1}$ and h_j represents the height between point x_{j-1} and x_j.

1.1 a)

A silhouette:

$$X = (x_0, h_1, x_1, \ldots, x_{i-1}.h_i, x_i, \ldots, h_n, x_n) \tag{1.1}$$

Assume, we are given a silhouette and a building (l, h, r) How does one update the silhouette to include the building. We use the following algorithm from hell

```
Time          | Line nr | Pseudocode
--------------+---------+--------------------------------------------------
              |         | addBuilding($X, (l, h, r)$)
1             |    1    | if $l > x_n$ //case 1
    4         |    2    |     add$(0, l, h, r)$ onto the end of $X$
    1         |    3    |     return
1             |    4    | if $r < x_0$ //case 2
    4         |    5    |     add$(l, h, r, 0)$ onto the front of $X$
    1         |    6    |     return
$\log_2(n)$   |    7    | $i = \min\{k | l < x_k\}$ //using binary search
$\log_2(n)$   |    8    | $j = \max\{k | x_k < r\}$ //using binary search
1             |    9    | if $l < x_0$ // case 3
    4         |   10    |     add$(l, h)$ to the front of $X$
1             |   11    | else if $l = x_{i-1}$ && $h > h_i$//case 4
    1         |   12    |     $h_i = h$
1             |   13    | else $x_{i-1} < l < x_i$ && $h > h_i$ //case 5
    2         |   14    |     add$(l, h)$ after $h_i$
n             |   15    | while $i \leq j$
    1         |   16    |     if $h > h_i$
        1     |   17    |         $h_i = h$
    1         |   18    |     $i$++
1             |   19    | if $j = n$ //case 6
    2         |   20    |     add $(h, r)$ on the end of $X$
3             |   21    | else if $j < n$ && $r = x_{j+1}$ && $h > h_{j+1}$ //case 7
    2         |   22    |     $h_{j+1} = h$
3             |   23    | else $j < n$ && $x_j < r < x_{j+1}$ && $h > h_{j+1}$ //case 8
    2         |   24    |     add$(h, r)$ after $x_j$
n             |   25    | cleanUpAisle4($X$)
```

With helping algorithm.

```
Time          | Line nr | Pseudocode
--------------+---------+--------------------------------------------------
              |         | cleanUpAisle4($X$)
n             |    1    | for $i = 1, \dots, n-1$
    1         |    2    |     if $h_i = h_{i+1}$
        2     |    3    |         delete $h_i, x_i$
```

From the analysis done alongside the code we see that the addBuilding algorithm runs in linear time.

Correctness: There is 8 start up cases that we need to take care of.

Case 1: the building we are adding to the silhouette lies completely to the right of the silhouette iff $l > x_n$ so we add $(0, l, h, r)$ onto the end of the silhouette. **Case 2:** is analogous to to case 1, the case where the building is to the left of our silhouette, so we add $(l, h, r, 0)$ onto the front. **Case 3:** before starting case 3, we compute $i = \min\{k | l < x_k\}$ and $j = \max\{k | x_k < r\}$ using binary search, the x values assosiated with these indicies will be the first we have to compare our building to, and case 1 and 2 ensure their existence. In case 3 we have $l < x_0$ in which case we add (l, h) to the front of the silhouette. **Case 4:** now we have $l = x_{i-1}$, which is well defined because if $l = x_0$ we get $i = 1$, and if $l > x_n$ we are in case 1. Having taken care of the formalities we change h_i to h iff $h > h_i$ since our building is on a perfect line with another building in the silhouette we take the height to be the highest of the two. **Case 5:** here we have $x_{i-1} < l < x_i$ pr definition of i, if also $h > h_i$ we add (l, h) after h_i to indicate that the skyline rises with the new building, between x_{i-1} and x_i.

At this stage we look at the intermediate buildings, and update their height if needed.
The remaining cases 6-8 are analogous to 3-5.
The cleanUpAisle4 algorithm overkills the problem and runs through the entirety of X and if $h_i = h_{i+1}$ we are in a situation with superflous information, so we delete h_i, x_i, this because we are in a situation $(\dots, x_{i-1}, h_i, x_i, h_i, x_{i+1}, \dots)$ so we can choose to delete the height to the left of x_i or the height to the right, since they are equal, we choose to delete the left for no particular reason. $\square$

1.2 b)

Given 2 silhouettes

$$X = (x_0, h_1, x_1, \dots, x_{i-1}.h_i, x_i, \dots\dots, h_n, x_n) \tag{1.2}$$
$$Y = (y_0, h'_1, y_1, \dots, y_{i-1}.h'_i, y_i, \dots, h'_m, y_m) \tag{1.3}$$

We wish to combine them into one. Assume without loss of generality that $m \leq n$

Time	Line nr	Pseudocode
		CombineSilhouette(X,Y)
m	1	for k = 1 to k=m
n	2	addBuilding(X,(y_{k-1}, h'_k, y_k)))

If addBuilding works, so does CombineSilhouette. The time for this algorithm is $O(mn) = O(n^2)$ since $m \leq n$.

1.3 c)

We wish to write the divide and conquer algorithm to take care of this problem.
The following algorithm takes an input on the form

$$\begin{aligned} X = \big(&(x_{0,0}, h_{0,0}, \dots x_{i,0}, h_{i,0}, \dots, h_{n_0-1,0}, x_{n_0,0}),\\ &(x_{0,1}, h_{0,1}, \dots x_{i,1}, h_{i,1}, \dots, h_{n_1-1,1}, x_{n_1,1}),\\ &\qquad\vdots\\ &(x_{0,m}, h_{0,m}, \dots x_{i,m}, h_{i,m}, \dots, h_{n_m-1,1}, x_{n_m,1})\big) \end{aligned}$$

Which is a collections of $m =: X.size$ silhouettes each of size n_i.

Time	Line nr	Pseudocode
		DnCCombine(X)
1	1	if $X.size = 1$
1	2	return X
m/2	3	for(i = 0 : i<X.size/2 : i++)
$\max\{n_i^2, n_{m-i}^2\}$	4	CombineSilhouette($X_i, X_{X.size-i}$)
1	5	delete $X_{X.size-i}$ from X
T(m/2)	6	DnCCombine(X)

This algorithm is very general and overshoots our purposes a bit, since we wish to use it initially on a collection of buildings,

$$X = \big((l_0, h_0, r_0), \dots, (l_n, h_n, r_n)\big) \tag{1.4}$$

and be able to reuse the algorithm all the way to the top. We also start from behind, compared to the way for example Merge-Sort works, which first divides up the input and the gathers it back together. We do not have to use effort to divide up our problem, because we only intend to use DnCCombine once and if later on we wish to add another building we just use addBuilding. So what happens when we call DnCCombine on a collection like (1.4)? We get a binary tree turned on its head with n nodes at the top each of size 1, in the second layer we get $n/2$ nodes of size 2, and at the third layer we have $n/4$ nodes of size 4, and at the i'th level we have $n/2^{i-1}$ nodes of size 2^{i-1}, until we reach the bottom with 1 node of size n, after $\log_2(n)$ levels. This isnt the usual way the recurrence formula/tree works, but it is *clearly* equivivalent to

$$T(n) = 2T(n/2) + n^2 \overset{\text{Master }Theorem}{\iff} T(n) = \Theta(n^2) \tag{1.5}$$

There exists an Indian man on youtube that has a way to solve this problem in $n\log(n)$ time, but we do not wish to plagerise him ;) $\square$