COMPUTATIONAL GEOMETRY
Assignment 3

Aman Choudhary
MTech Coursework, CSA 2020
Sr No: 17920

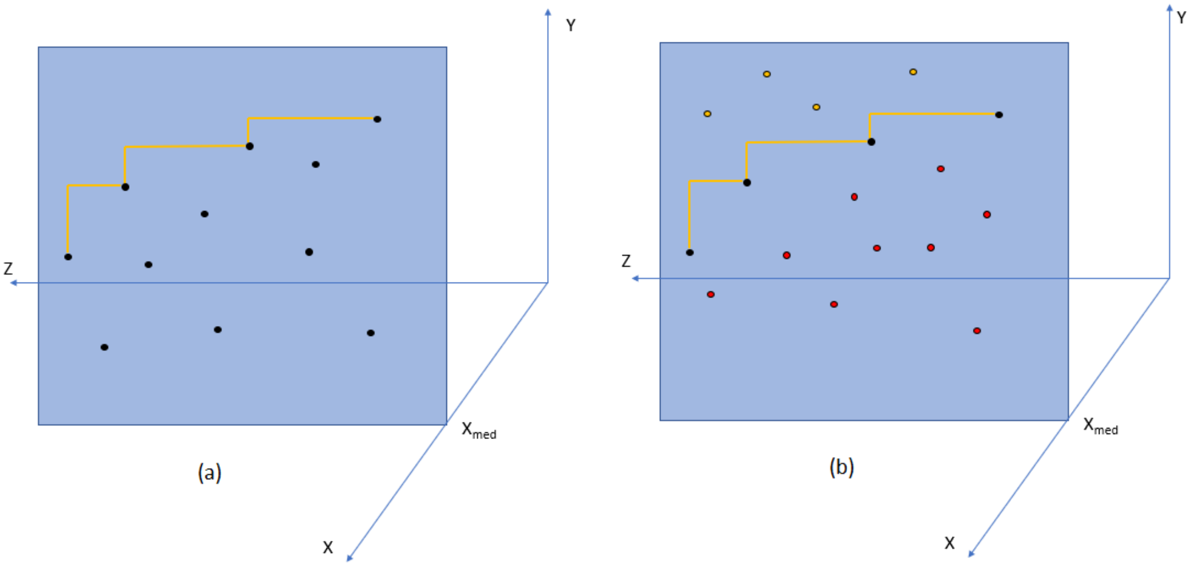
April 09, 2021

1 Problem 1

1.1 Merge Algorithm

Let P be the set of n points in 3D. The algorithm computes a median plane ($x = x_{med}$) such that there are equal number of points of P on either side of the median plane; we call these sets P_l and P_r, respectively. Let S_l and S_r be the skyline points of P_l and P_r respectively. Now, we describe a procedure for merging S_l and S_r into S, which is the set of skyline points of P.

- We first include every point in S_r into S. This is because, the points in S_r are the ones which are dominating the points of P_r. There exists no other point to dominate them. Hence, they will belong to S.
- However, we note that not all points in S_l will belong to S. Every point of S_l is dominated by every point of S_r along the $x - axis$. We need to prune those points in S_l which are dominated by some point in S_r along the $y - axis$ and $z - axis$ as well.
- For this, we first project the set S_r onto the median plane x_{med}. The resultant projections are points on a 2D plane. Let us denote this set by S_r^P, these points are shown by black dots in figure 1(a).
- We now compute a 2D skyline of points in S_r^P. This is denoted by the yellow staircase in figure 1(a).
- Then we project the points in S_l on the same plane, x_{med} and call it S_l^P. The situation is shown in figure 1(b). We note that there are two types of points:
  (a) **The points above the staircase, shown in yellow.** These are the points which are dominated by every point in S_r along $x - axis$. However, no point in S_r dominates these points in either $y - axis$ or $z - axis$. Hence, these points will belong to S. Thus, we add this points shown in yellow to our result.
  (b) **The points below the staircase, shown in red.** These are the points which are dominated by every point in S_r along $x - axis$. Moreover, they are also dominated by some point along the staircase, along the $y - axis$ and $z - axis$. Hence, these points will not belong to S. Thus, we prune such points.



1.2 Proof of Correctness

- We will do this using contradiction. Let us assume that there exists a point p which is a skyline point but $p \notin S$. This means that p must belong to S_l, because we are including every point in S_r to S. Since, p is a skyline point in S_l, there must exist no point p' such that all p' dominates p along all the axes. This means that p lies above the yellow staircase, otherwise it would be dominated along all axes. However, we are making sure that no such is pruned. Hence, p must belong to S, thus reaching a contradiction.
- Let us assume that there exists a point p which is not a skyline point but $p \in S$. This means that p must belong to S_l, because every point in S_r is a skyline point of P. Since, p is not a skyline point, there must exist some point p' such that p' dominates p along all the axes. However, we are making sure that such a point is always pruned. Hence, p cannot belong to S, thus reaching a contradiction.

1.3 Time Complexity

- The 2D skyline of S_r^P can be computed in $O(n \log n)$ time using the **Sweepline Algorithm**.
- The pruning step can be done efficiently by constructing a height-balanced BST of the points in the staircase, based on the . Then for every point p in S_l^P, we find its successor p' in the tree, such that, $p.z < p'.z$. Now, it is already known that $p.x < p'.x$. We only need to check whether, $p.y < p'.y$. If it is, then we prune p.
- In the previous step, we are considering only the successor for p, because for the predecessors, $p.z \geq p'.z$. Moreover the successor has the highest $y - coordinate$ among all points which lie further on the staircase (in direction of +ve $z - axis$). So, p' is most eligible to dominate p.
- Querying the BST for successor takes $O(\log n)$ time. And we may need to do it for $O(n)$ points. Hence, the entire pruning step can be accomplished in $O(n \log n)$ time.
- Thus, the total time complexity of the $Merge\ Algorithm$ is $O(n \log n)$. We can thus write the recurrence for the entire algorithm as,

$$T(n) \ = \ 2T(n/2) \ + O(n \log n)$$

- Solving the above recurrence using Master's theorem gives us total time complexity of the Skyline algorithm to be $O(n \log^2 n)$.

2 Problem 2

2.1 Merge Algorithm

- Before the algorithm starts, we first pre-sort P based on the decreasing order of $z - coordinate$ values. Also, we maintain the invariant that the skyline points of any subproblem (in the divide and conquer algorithm) are reported in decreasing order based on their $z - coordinate$ values. Now, we describe a more efficient procedure for merging S_l and S_r into S, which is the set of skyline points of P.
- The algorithm is similar to merging two sorted lists. Here our two lists are namely S_l and S_r. Here we have three cases:

  1. **Case 1:** $p_l.z >= p_r.z$ In this case, the point p_r does not dominate p_l along the $z - axis$, hence p_l will be included in S. Note that, all points of S_r will be trivially included in S.
  2. **Case 2:** $p_l.z < p_r.z$
     (a) $pl.y < y_{max}$: Here, y_{max} denotes the maximum $y - coordinate$ recorded for among all the points of S_r scanned till now. If the condition is satisfied then there exits some point in S_r whose $y - coordinate$ is y_{max} and it dominates p_l along all axes. Hence it is pruned.
     (b) $pl.y >= y_{max}$: This means that there exits no point in S_r till now, whose $y - coordinate$ dominates that of p_l. Hence we scan ahead.

MERGE_SKYLINE

```
S = φ
p_l = S_l[0]
p_r = S_r[0]
y_max = p_r.y
while p_l != NULL and p_r != NULL
    if p_l.z >= p_r.z               //Case 1
        S = append(S, p_l)
        p_l = p_l.next
    else
        if p_l.y < y_max                //Case 2(a)
            p_l = p_l.next              //Prune p_l
        else                            //Case 2(b)
            S = append(S, p_r)          //Add p_r to Skyline points
            p_r = p_r.next
            y_max = max(y_max, p_r.y)
while p_l != NULL
    if p_l.y >= y_max
        S = append(S, p_l)              //Add p_l to Skyline points
    p_l = p_l.next
```

2.2 Time Complexity

- The initial pre-sorting step takes $O(n \log n)$ time.
- The merge algorithm described above takes $O(n)$ time, as all we are doing are scanning the two lists, S_l and S_r.
- Thus, the total time complexity of the $Merge\ Algorithm$ is $O(n \log n)$. We can thus write the recurrence for the entire algorithm as,

$$T(n) \;=\; 2T(n/2) \;+ O(n)$$

- Solving the above recurrence using Master's theorem gives us total time complexity of the Skyline algorithm to be $O(n \log n)$.

3 Problem 3

Let us assume that there exists a point $p \in R_i^+$ but is not a skyline point. Then it must be true that there exists $q \in P$ which dominates p. Now there are two cases, at the beginning of the i^{th} iteration:

1. **Case 1: $q \in M$, i.e. q is already classified:** If this was the case, then q would have pruned p in the second pass of some iteration, when it was classified. Hence, we reach a contradiction that $p \in R_i^+$.
2. **Case 2: $q \in P_i$, i.e. q is not yet classified:** In this case, we have two sub-cases:
   (a) **$p \in R_i^+$ after first pass:** In this case, if q is scanned before p during second pass, then it will remove p from R_i^+, and p would not be included in R_i^+ when it is scanned later. Also, if q is scanned after p during second pass, the it again prune p from R_i^+. Hence, we reach a contradiction that $p \in R_i^+$.
   (b) **$p \notin R_i^+$ after first pass, but $p \in R_i^+$ after second pass** In can happen when p happens to dominate some point r. However, such a point would also be dominated by q. If q is scanned before p during second pass, then it will remove r from R_i^+, and hence p would not be included in R_i^+ when it is scanned later, as it has nothing to dominate. Also, if q is scanned after p during second pass, the it would prune both p from r. Hence, we reach a contradiction that $p \in R_i^+$.

4 Problem 4

- In the first pass of the algorithm, we calculate the smallest number in the input stream and store it.
- Similarly, in the second pass, we calculate the element which is just greater than the smallest number calculated in first pass. We store this element and discard the result of first pass.
- In this manner, in the i^{th} pass, we calculate the element which is just greater than the element calculated in the $(i-1)^{th}$ pass. We store this number to be used in $(i+1)^{th}$ pass and discard the result of $(i-1)^{th}$ pass.
- When n is odd, we need $\left(\frac{n+1}{2}\right)$ passes to find the median element. However, when n is even, we find the $\left(\frac{n}{2}\right)^{th}$ and $\left(\frac{n}{2} + 1\right)^{th}$ element and take their average.
- In both cases, we need $O(n)$ passes, where each pass itself takes $O(n)$ time. So, the overall time complexity of the algorithm is $O(n^2)$.
- Also, in every pass, we only need the result of previous pass for calculating the result. Hence, the space complexity is $O(1)$, thus fulfilling our criteria of $o(n)$ space.
