Department of Computer Science
Ashoka University

Design and Analysis of Algorithms

Programming Assignment 1

Collaborators: None **Name: Rushil Gupta**

Introduction

For this assignment, we need to use the line sweep method to calculate the maximal points in a set of 2D points. But, we want to make all layers of the set of points in one line sweep.

First, we will describe the data structure we are using to represent multiple staircases and to determine which staircase does a new point belong to in $O(\log n)$ time. For this, we use a vector of vectors, where each vector represents a staircase. Specifically, we creat 2 vectors, `tops` and `layers`. `tops` is a vector of integers, where `tops`$[i]$ is the y-coordinate of the last point in the i-th layer. `layers` is a vector of vectors, where `layers`$[i]$ is the i-th layer.

This allows us to find the layer of a point in $O(\log n)$ time by performing a binary search on `tops` to find the smallest index j such that `tops`$[j] < y$. If no such index exists, then the point does not fit in any existing layer, and a new layer is created. Otherwise, the point is inserted into layer j.

Also, in CPP, vectors take $O(1)$ time to insert an element at the end and take $O(1)$ time to access an element by index. So, it is an efficient data structure to use for this problem, and we do not need to delve into balanced binary trees or other complex data structures.

Proof of Correctness for the Maximal Layers Algorithm

We consider the following algorithm to compute all maximal layers of a set of 2D points in one line sweep. The algorithm works as follows:

1. **Sorting:** Given a set S of n points, the points are first sorted in descending order of their x-coordinates. If two points have the same x-coordinate, they are sorted by descending y-coordinate. This sorting ensures that when processing a point $p = (x, y)$, every point with a larger x-coordinate (and, in case of a tie, a larger y-coordinate) has already been processed.
2. **Layer Assignment via Binary Search:**
   - A vector `tops` is maintained, where `tops`$[i]$ is the y-coordinate of the last point in the i-th layer.
   - For each point $p = (x, y)$ in the sorted order, a binary search is performed on `tops` to find the smallest index j such that
   $$\texttt{tops}[j] < y.$$
   - If no such index exists, then p does not fit in any existing layer, and a new layer is created with p as its first element. Otherwise, p is inserted into layer j, and `tops`$[j]$ is updated to p's y-value.

Invariant

During the algorithm, the following invariant is maintained:

Invariant: The vector `tops` is kept in non-increasing order:

$$\texttt{tops}[0] \geq \texttt{tops}[1] \geq \cdots \geq \texttt{tops}[k-1],$$

where k is the number of layers created so far.

Proof of Correctness

- **Processing Order:** Since the points are sorted in descending order by x and y, when processing any point p, all points q that dominate p are already processed.
- **Layering:** The binary search on `tops` finds the lowest-index layer where the current point p can be inserted. Since p can never be maximal on the x-axis (due to the sorting), it can only be added to a layer where the last point has a y-coordinate less than $p.y$. Finding the lowest index j such that $\texttt{tops}[j] < y$ ensures that p is placed into the most maximal layer possible.

So, since we maintain our invariant and process points in the correct order, we can correctly know the maximal layer a point belongs to. Applying this method to all the points using our line sweep algorithm, we can correctly partition the points into their maximal layers.

Time Complexity Analysis

The running time of the algorithm is determined by the following steps:

1. **Sorting:** Sorting the n points in descending order (by x, and by y for ties) takes $O(n \log n)$ time (CPP uses an efficient sorting algorithm).
2. **Layer Construction:**
   - For each of the n points, a binary search is performed on the `tops` vector. In the worst case, each point is in a different layer, so the number of maximal layers is bounded by n. So, this binary search is done in $O(\log n)$ time.
   - Each point is processed exactly once. Thus, the total time for this step is

$$O(n \log n).$$

Therefore, the overall time complexity of the algorithm is:

$$O(n \log n) + O(n \log n) = O(n \log n).$$