Task: Developing Algorithms to Maximize the Sum of Point Weights while Maintaining Maximum Chain Weight

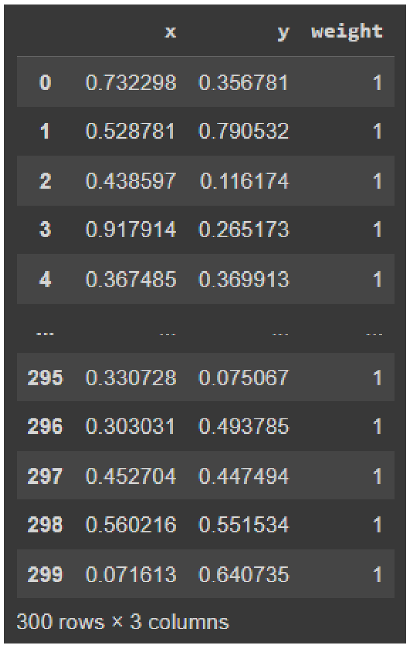
Team Members: Rotem Kimhi, Amit Madmoni, Amit Lustiger, David Bart

Introduction

The problem statement focuses on a collection of points on a plane, where each point has an associated weight. The primary objective is to increase the sum of the weights of all points as much as possible without altering the maximum weight of the longest chain of points, initially considered with each point having a weight of one.
This task involved algorithmic challenges, including sorting, dynamic programming, and the utilization of different data structures.

Data Generation

In order to allow easy testing, we generated every random dataset using pythonic libraries. Each point is represented by an (x, y) coordinate, and initially, each point was assigned a weight of one.
For easier demonstration, our examples in this report will use a dataset of 300 points. As can be seen on the right:

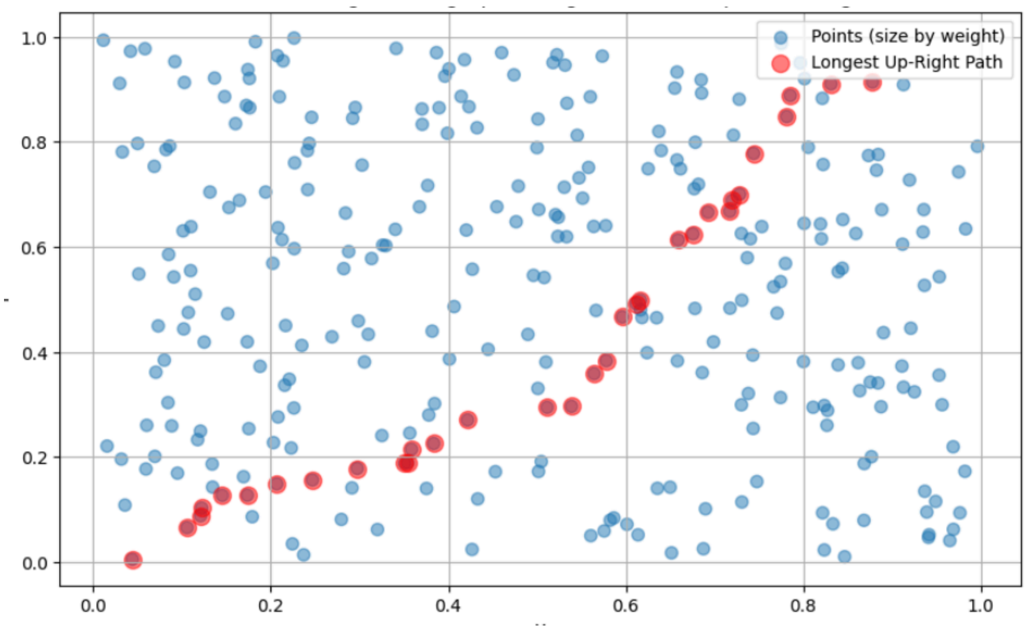


	x	y	weight
0	0.732298	0.356781	1
1	0.528781	0.790532	1
2	0.438597	0.116174	1
3	0.917914	0.265173	1
4	0.367485	0.369913	1
...	...	...	...
295	0.330728	0.075067	1
296	0.303031	0.493785	1
297	0.452704	0.447494	1
298	0.560216	0.551534	1
299	0.071613	0.640735	1

300 rows × 3 columns

Longest Chain Identification

A dynamic programming approach was employed to identify the longest chain of points. This involved sorting the points based on their coordinates and sequentially finding the longest subsequence where each point in the sequence is greater than the preceding one in both coordinates.
The graph below portrays the 300 points and the longest path:



Weight Optimization

Our algorithm maximizes the total "importance" (weight) of a collection of points spread across a plane, ensuring the cumulative importance of a specific sequence - the longest increasing path - remains unaltered. It employs a sophisticated tool, the segment tree, akin to a dynamic scoreboard, to efficiently manage and update weights. By carefully evaluating each point not on the longest increasing path, it determines the maximum possible increase in importance, leveraging the segment tree to track and update the highest achievable weights based on the positioning of points.

The Role of Segment Trees

Segment trees are a powerful data structure used for storing information about array segments and enabling efficient querying of aggregate data over a range, such as sums, maximums, or minimums, within the array.
In the context of our problem, segment trees are utilized to manage and update the weights of points efficiently.

The main advantage of using a segment tree lies in its ability to handle multiple updates and queries in logarithmic time, making it highly suited for problems where frequent modifications and queries are expected.
Given that our task involves dynamically updating the weights of points to maximize the total weight without affecting the maximum weight of the longest chain, segment trees provide an efficient mechanism for these updates.

Matching the Problem Requirements

Our problem necessitates finding the optimal way to increase the weights of all the points that are not present in the longest chain, under the constraint that the weight of the longest chain remains unchanged. The segment tree fits this requirement perfectly because:

Dynamic Updates: It allows for the dynamic updating of point weights based on their position in the y-coordinate sorted list. This is crucial for efficiently recalculating weights as we iterate through points not in the longest chain.

Efficient Queries: The tree facilitates rapid queries to find the maximum weight in any given range of y-coordinates.
This capability is essential for determining how much we can increase a point's weight without violating the constraint on the longest chain's weight.

Implementation and Time Complexity

Implementing the segment tree involves constructing a binary tree where each leaf represents a weight of a single point, and each internal node stores the maximum weight of the segment (or range of points) it covers.

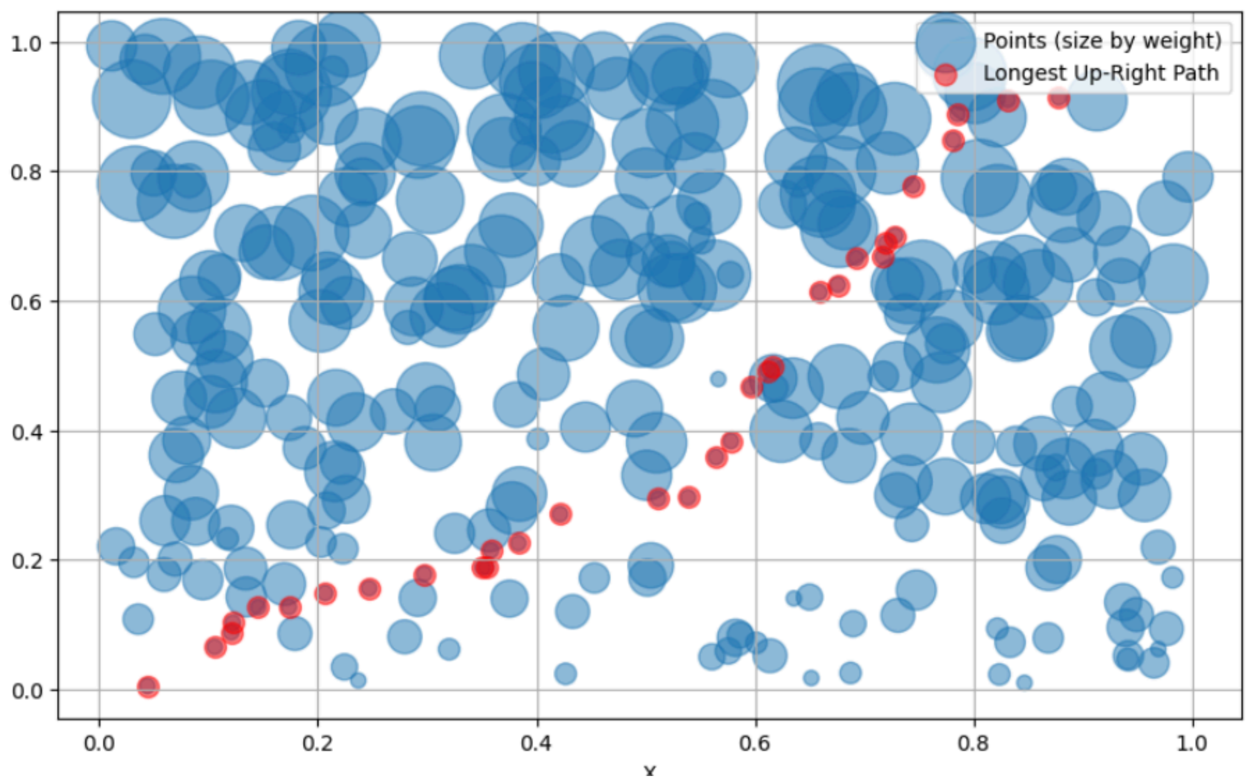
This structure supports two primary operations:

Update: The weight of a point is updated, and the change is propagated up the tree to maintain accurate maximum values for each segment.
This operation is crucial for adjusting weights while preserving the constraint on the longest chain's weight.

Query: The maximum weight within a specific range of y-coordinates is queried to determine the allowable increase for a point's weight. This operation ensures that any weight adjustment does not impact the longest chain's predefined weight.

Both update and query operations in a segment tree have a time complexity of **O(log n)** where n is the number of points (The unique y-coordinates number). This efficiency is derived from the tree's binary nature, which allows it to divide the problem space in half at each level of the tree. Consequently, the segment tree's logarithmic time complexity makes it an ideal choice for our problem, where numerous updates and queries are performed across a large set of points.

The graph below portrays the longest path and the weighted points:



Results

The algorithm successfully increased the sum of the weights of the points without altering the weight of the longest chain. The visualization provides clear evidence of the weight distribution before and after the optimization, showing an increase in the overall sum of weights while maintaining the longest chain's weight constant.

Part B - weight optimization with squared weights

In addressing the second task, which evolves from the first by focusing on maximizing the sum of squared weights of all points in a dataset, while preserving the weight of the longest increasing subsequence (LIS), we chose to adapt our initial algorithm for several reasons. Our decision was informed by a comprehensive evaluation of various algorithms, through which our initial algorithm consistently demonstrated superior performance in most scenarios compared to the alternatives.

Rationale for Algorithm Selection

Optimization towards the Top-Right Corner: Our algorithm inherently prioritizes increasing weights of points closer to the top-right corner of the coordinate plane. This strategic weighting not only aligns with the goal of maximizing the sum of squared weights - since squaring larger weights has a more pronounced effect - but also navigates the constraint of not surpassing the LIS's weight sum. By focusing on these strategically positioned points, the algorithm effectively increases the overall sum of squared weights without risking the creation of a new LIS with a higher weight sum.

Selective Weight Enhancement: The algorithm's methodology of selectively enhancing weights, especially of points not within the LIS, inherently ensures that the sum of weights of any potential new increasing subsequences remains below that of the original LIS. This selectivity is crucial for adhering to the task's constraints, where the primary objective is to increase the dataset's overall weight metric without altering the preeminence of the LIS.

Efficient Utilization of the Segment Tree: The use of a segment tree for weight updates offers an efficient means of managing the dynamic nature of weight adjustments across the dataset. This efficiency is vital for processing large datasets, where brute force or less sophisticated algorithms might falter in terms of performance and accuracy.

Potential Disadvantages

Despite its advantages, the algorithm is not without potential disadvantages. One such concern is the possibility of under-optimizing paths not leading towards the top-right corner. These paths, while not surpassing the LIS in their sum of weights, may also warrant weight improvements to enhance the overall sum of squared weights. The algorithm's focused approach on points nearer to the top-right corner might inadvertently neglect these paths, thereby not fully optimizing the dataset's weight distribution as per the task's objective.

Conclusion

In conclusion, our algorithm stands out for its strategic focus on increasing the weights of points positioned closer to the top-right corner, leveraging the mathematical advantage of squaring higher weights, and its careful compliance with the task's constraints through selective weight enhancement. While acknowledging the potential oversight of certain paths, the algorithm's overall efficacy in achieving the desired outcome - maximizing the sum of squared weights without altering the LIS - justifies its selection for this complex optimization challenge.