CS 478 Computational Geometry
Homework 3

Barkın Saday
Bilkent University
Department of Computer Science

21902967

Question 1

Problem:
Given two sets A and B of points in the plane, each arranged as a staircase (that is, each set coincides with its set of maxima in the dominance relation), find the pair (p_i, p_j), $p_i \in A$ and $p_j \in B$, closest in the L_1-metric. Is linear time achievable?

Solution:
Since both sets A and B are staircases, we can efficiently find the closest pair by using a two-pointer approach:

- Assume that the points in A and B are sorted in increasing x-coordinate and decreasing y-coordinate.
- Initialize two pointers: one at the beginning of A, and one at the beginning of B.
- At each step, compute the L_1 distance between the current points.
- Move only one pointer at a time:
  - Advance the pointer (either in A or in B) whose movement is more likely to reduce the distance, based on comparing the coordinates.
- If moving neither pointer reduces the distance, then the current pair is globally optimal and the search can be terminated.

Correctness:
The correctness follows from the properties of staircases and the monotonic behavior of the L_1 metric. As we move forward along the staircase, future points will have larger x-coordinates and smaller y-coordinates, which cannot yield a smaller Manhattan distance than the current closest pair. Therefore, if no move improves the distance, the current pair must be the global optimum.

Complexity:
Each pointer advances at most once per step, and each point is visited at most once. Thus, the algorithm runs in $O(n + m)$ time, where $n = |A|$ and $m = |B|$.

Conclusion:
Yes, it is possible to find the closest pair between two staircases in linear time.

Question 2

Problem:
You are given two arbitrary convex polygons P and Q with N and M vertices respectively. The polygons can be disjoint, one inside the other, or the boundaries might intersect a number of times. Find the smallest and largest number of vertices on the boundary of ConvexHull($P \cup Q$) in terms of N and M. Draw an example for both the smallest and largest cases.

Solution:

Smallest number of vertices:
The convex hull of $P \cup Q$ can have as few as 3 vertices, independent of N and M, if P and Q are nested or close enough such that their union forms a simple triangle. Thus, the minimum number of vertices is:

$$\boxed{3}$$

Largest number of vertices:
To maximize the number of vertices on the convex hull, P and Q should be placed so that all their vertices contribute to the boundary. This can be achieved by placing P and Q adjacent to each other with outward-facing curved shapes, ensuring that no vertex is hidden.

Thus, the maximum number of vertices on the convex hull is:

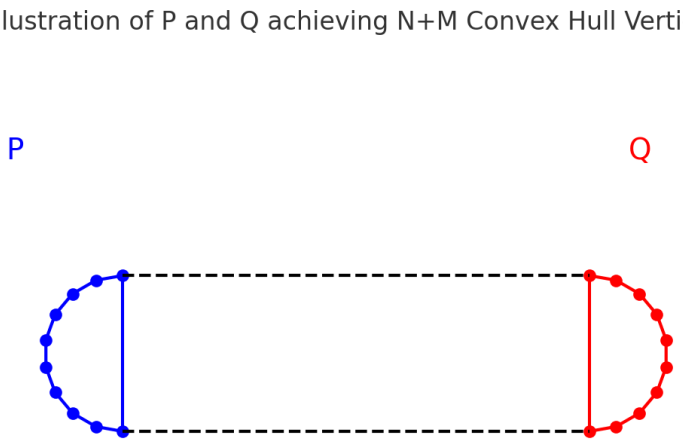
$$\boxed{N + M}$$

If, for some reason, the two vertices at the connecting flat edges of P and Q are not considered to contribute to the convex hull (for instance, if they become interior points in a degenerate configuration), then the maximum number would instead be:

$$\boxed{N + M - 2}$$

However, under normal conditions, $N + M$ vertices are achievable.

Illustration of the Maximum Case: N+M



Two convex polygons P and Q are shaped as half-circles:

- P curves towards the left and has $N-2$ vertices along its curved side.
- Q curves towards the right and has $M-2$ vertices along its curved side.
- The flat sides of P and Q are adjacent, and the 2 vertices on each flat side contribute to the convex hull.

Thus, in total, all N and M vertices appear on the convex hull.

Question 3

Problem:
Two points p_i and p_j determine an edge of the Delaunay triangulation if and only if there exists a circle passing by these two points that does not contain any other point in its interior. Show that this definition leads to a triangulation that satisfies the circumcircle property. Also show that this definition gives a unique triangulation, assuming no four points are cocircular.

Solution:

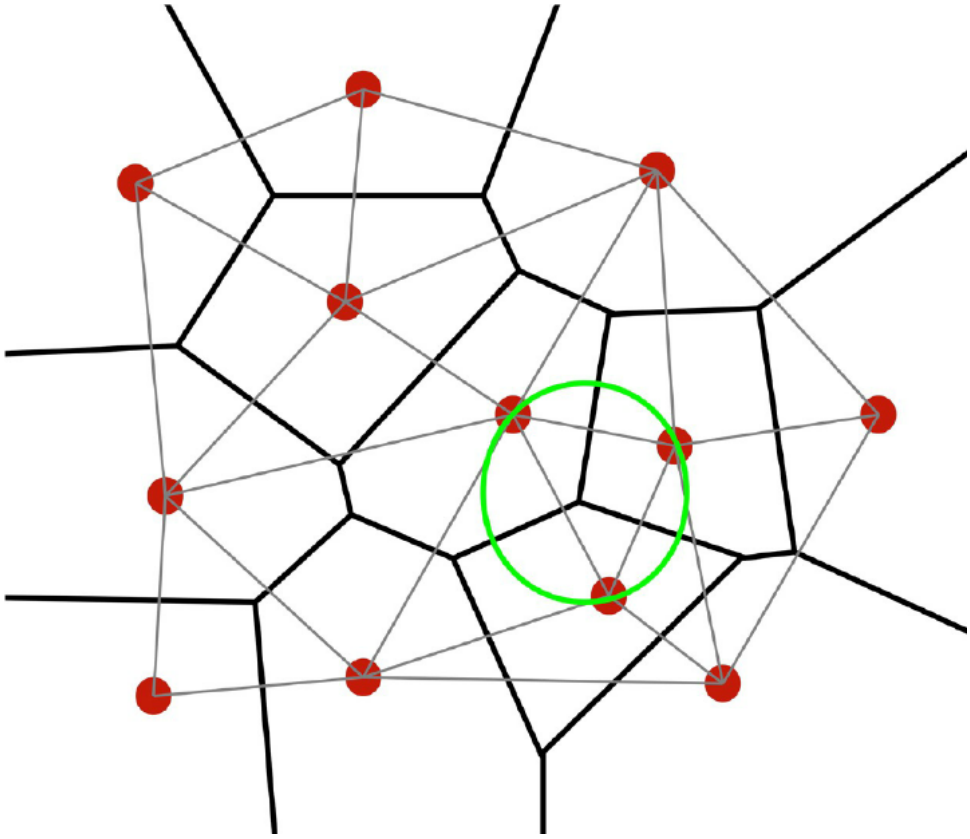
Circumcircle Property:
Given the definition, two points are connected if there exists an empty circle through them. If three points (p_i, p_j, p_k) are mutually connected, then the individual circles for each edge do not contain other points. Thus, the unique circumcircle passing through all three points will also not contain any other point inside, satisfying the empty circumcircle property.

Duality with Voronoi Diagram:
Since two points are connected when their corresponding Voronoi cells are adjacent, and

the empty circle property is satisfied, this triangulation corresponds exactly to the dual graph of the Voronoi diagram.

Uniqueness:
Under the assumption that no four points are cocircular, the circumcircle through any three points is uniquely defined. No ambiguity arises for triangulation choices (no flipping between diagonals), and thus the Delaunay triangulation is unique.

Illustration:



The figure shows a generic Delaunay triangulation. Edges connect points based on the empty circle property, and circumcircles of triangles do not enclose any other points, demonstrating the Delaunay conditions.

Question 4

Problem:
Propose a Delaunay Triangulation algorithm (DELAUNAY_TRIANGULATION) that is optimal ($O(N \log N)$). The algorithm should use the DCEL representation described in the textbook and allow transforming the result into the Voronoi diagram in $O(N)$ time.

Solution:

We use the classical **Divide and Conquer** strategy to compute the Delaunay triangulation in optimal $O(N \log N)$ time. The algorithm proceeds as follows:

- **Input:** A set S of N points in the plane.
- **Sort** the points by their increasing x-coordinate.
- **Recursively:**
  - Divide S into two halves: left set S_L and right set S_R.
  - Recursively compute the Delaunay triangulation for each half, yielding T_L and T_R.
- **Merge:**
  - Find the **lower common tangent** between T_L and T_R.
  - From the lower tangent upward, "zip" the triangulations together while maintaining the empty circle property:
    - Add new edges across S_L and S_R.
    - Each time an edge is added, insert corresponding directed edges into the DCEL.
    - Properly update the six fields of each DCEL entry:
      - **V1, V2:** Origin and terminus vertices.
      - **F1, F2:** Faces to the left and right of the directed edge.
      - **P1, P2:** Pointers to the next edges around the origin and terminus in counterclockwise order.
- **Output:** The complete Delaunay triangulation represented using the DCEL.

Transforming into Voronoi Diagram:
After obtaining the Delaunay triangulation:

- Compute the **circumcenter** of each triangle (face) in the DCEL.
- For each pair of adjacent triangles, create an edge between their circumcenters.
- Assemble the Voronoi cells around each site by connecting the circumcenters of adjacent triangles.
- Since each triangle and each edge are processed once, this transformation requires only $O(N)$ time.

Complexity:

- Delaunay triangulation construction: $O(N \log N)$
- Voronoi diagram construction from DCEL: $O(N)$

Conclusion:
By using the Divide and Conquer method and carefully maintaining the DCEL structure during merging, we achieve an optimal Delaunay triangulation that can be efficiently transformed into a Voronoi diagram.

Question 5

Problem:
M is a 2D matrix that represents a discrete 2D plane. $M[i][j] = k$ if there is a point k located at (i, j); otherwise, $M[i][j] =$ null. We want to mark each empty cell with a marker of the point that is closest to that cell, using Manhattan distance. If multiple points are at the same distance, any one of them can be used. Give an $O(XY)$ algorithm to mark all the cells, where X and Y are the matrix dimensions.

Solution:

We use a **multi-source Breadth-First Search (BFS)** to solve the problem efficiently:

- Initialize a queue containing all cells (i, j) such that $M[i][j]$ is not null (i.e., cells that already have a point).
- Each queued entry carries its corresponding marker k.
- While the queue is not empty:
  - Pop a cell (i, j) from the queue.
  - For each of its four neighbors (up, down, left, right):
    - If the neighbor is empty (i.e., $M[i'][j'] =$ null):
      - Assign $M[i'][j'] = k$, where k is the marker of the current cell.
      - Enqueue (i', j') into the queue.

Correctness:

- BFS guarantees that each empty cell is filled the first time it is reached, which corresponds to the shortest (minimum Manhattan) distance.
- Because all edge movements (up, down, left, right) have the same cost, the first reach is guaranteed to be the closest.
- Ties are handled automatically by the BFS traversal: any of the closest points can be assigned without affecting correctness.

Time Complexity:

- Each cell is enqueued and processed at most once.
- The total number of cells is XY, leading to an overall $O(XY)$ time complexity.

Conclusion:
Using a multi-source BFS allows us to fill all cells correctly according to the Manhattan distance criterion, in linear time relative to the matrix size. The solution is efficient, generic, and robust to all configurations of points.