1 Convex Hull

Problem 1. Prove: For any convex polygon $\mathcal{H}$ with vertices $P_1, P_2, \ldots, P_n$ given in counter-clockwise order, and let P be any point in the region bounded by $\mathcal{H}$, then

$$\overrightarrow{OP} = \sum_{i=1}^{n} w_i \overrightarrow{OP_i}, \text{where } w_1, w_2, \ldots, w_n \geq 0, \text{ and } \sum_{i=1}^{n} w_i = 1.$$

Conversely, given any

$$\overrightarrow{OP} = \sum_{i=1}^{n} w_i \overrightarrow{OP_i}, \text{where } w_1, w_2, \ldots, w_n \geq 0, \text{ and } \sum_{i=1}^{n} w_i = 1,$$

show that $P \in \partial H \cup \text{int}(\mathcal{H})$.

Proof.

By definition, the case where $n = 2$ is always true. The region degenerates into a segment $\overline{P_1P_2}$. $\overrightarrow{OP}$ can thus be expressed as

$$\overrightarrow{OP} = \overrightarrow{OP_1} + \lambda\overrightarrow{P_1P_2} = \overrightarrow{OP_1} + \lambda(\overrightarrow{OP_2} - \overrightarrow{OP_1}) = (1-\lambda)\overrightarrow{OP_1} + \lambda\overrightarrow{OP_2},$$

where $\lambda \in [0, 1]$.

For $n = 3$, the points form a triangle. Consider the case that P is on the edge of the triangle (WLOG, denote it as $A \in \overline{P_2P_3}$), then

$$\overrightarrow{OA} = \overrightarrow{OP_2} + \overrightarrow{P_2A} = \overrightarrow{OP_2} + \lambda\overrightarrow{P_2P_3} = \lambda\overrightarrow{OP_2} + (1-\lambda)\overrightarrow{OP_3}, \lambda \in [0, 1]$$

For the case that P is lying inside the interior of the convex polygon (WLOG, let $B \in \overline{DA}$), then $\overrightarrow{DE} = \xi\overrightarrow{DA}, \xi \in [0, 1]$, and

$$\begin{aligned}\overrightarrow{OE} = \overrightarrow{OD} + \overrightarrow{DE} &= \overrightarrow{OB} + \lambda(\overrightarrow{OC} - \overrightarrow{OB}) + \xi\overrightarrow{DA} \\ &= \overrightarrow{OB} + \lambda(\overrightarrow{OC} - \overrightarrow{OB}) + \xi(\overrightarrow{OA} - \overrightarrow{OD}) \\ &= \xi\overrightarrow{OA} + (1 - \lambda - \xi + \lambda\xi)\overrightarrow{OB} + (\lambda - \lambda\xi)\overrightarrow{OC}.\end{aligned}$$

This shows that we can always represent any point on or within the triangle.

For $n > 3$, let P be a point situated in a triangle $P_1P_iP_{i+1}$ (this is always possible by convexity), where $2 \leq i \leq n-1$.

We can then simplify the problem to the case of $n = 3$ (setting all irrelevant weights 0; or $n = 2$ if degenerated).

By Mathematical Induction. Denote the predicate $P(k)$ as

$$\text{“}\overrightarrow{OP} = \sum_{i=1}^{k} w_i \overrightarrow{OP_i}, \text{where } w_1, w_2, \ldots, w_k \geq 0, \text{ and } \sum_{i=1}^{k} w_i = 1.\text{”}$$

(a) (Base Case) $P(2)$ is true.

(b) (Inductive Step) Suppose $P(k)$ is true, i.e.

$$\text{“}\overrightarrow{OP} = \sum_{i=1}^{k} w_i \overrightarrow{OP_i}, \text{where } w_1, w_2, \ldots, w_k \geq 0, \text{ and } \sum_{i=1}^{k} w_i = 1.\text{”}$$

Then, for $P(k+1)$:

$$\text{“}\overrightarrow{OP} = (1 - w_{k+1})(\sum_{i=1}^{k} \frac{w_i}{1 - w_{k+1}} \overrightarrow{OP_i}) + w_{k+1}\overrightarrow{OP_{k+1}}.\text{”}$$

By the Induction Hypothesis, $\sum_{i=1}^{k} \frac{w_i}{1-w_{k+1}} \overrightarrow{OP_i} \in \partial\mathcal{H} \cup \text{int}(\mathcal{H})$.

Since $\overrightarrow{OP_{k+1}} \in \partial H$, by convexity, $\overrightarrow{OP} \in \partial\mathcal{H} \cup \text{int}(\mathcal{H})$.

□

Problem 2. Let P_1 and P_2 be two sets of points such that any point of P_1 has a smaller x-coordinate than all the points in P_2. You are also given the convex hulls of P_1 and P_2, denoted as $\text{CH}(P_1)$ and $\text{CH}(P_2)$, respectively. The vertices on each convex hull are sorted clockwise. Describe an algorithm to compute $\text{CH}(P_1 \cup P_2)$ in $O(n)$ time, where $n = |P_1| + |P_2|$.

Solution. We initialize p' to be the rightmost point of P_1 and p'' to be the leftmost point of P_2. We will walk p' backwards along P_1 and p'' forward along P_2 until hitting the vertices defining the tangent line.

Let q' be the point immediately preceding p' on P_1, q'' be the point immediately following p'' on P_2. Observe that if $\text{orient}(p', p'', q'') \geq 0$, then we can move p'' to the next point along P_2. Similarly, if $\text{orient}(p'', p', q') \leq 0$, then we can advance p' to its predecessor along P_1. When neither of these conditions applies, we have reached the points of mutual tangency.

```
1:  procedure UpperTangent(P1, P2)
2:      Let p' be the rightmost point of P1, and let q' be its predecessor.
3:      Let p'' be the leftmost point of P2, and let q'' be its successor.
4:      while (orient(p', p'', q'')≥ 0 and orient(p'', p', q')≤ 0) do
5:          while (orient(p', p'', q'')≥ 0) do
6:              advance p'' and q'' to their successors on P2.
7:          end while
8:          while (orient(p'', p', q')≤ 0) do
9:              advance p' and q' to their predecessors on P1.
10:         end while
11:     end while
12:     return (p', p'')
13: end procedure
```

Problem 3. The convex hull of a set S is defined to be the intersection of all convex sets that contains S. For the convex hull of a set of points it was indicated that the convex hull is the convex set with smallest perimeter. We can show that these are equivalent definitions by proving the followings:

(a) The intersection of two convex sets is convex. (Thus, the intersection of a finite family of convex sets is convex as well.)

(b) The smallest perimeter polygon $\mathcal{P}$ containing a set of points P is convex.

(c) Any convex set containing the set of points P contains the smallest perimeter polygon $\mathcal{P}$. Hence every segment in $\mathcal{P}$ must be a line between two points in P. Any convex set containing P must contain all these line segments. Hence, any convex set containing P must contain $\mathcal{P}$.

Proof.

(a) Let the two convex sets be A, B, respectively. Then, by definition,

$$\forall \overrightarrow{x}, \overrightarrow{y} \in A, \lambda \in [0,1], \lambda \overrightarrow{x} + (1-\lambda)\overrightarrow{y} \in A.$$

$$\forall \overrightarrow{x}, \overrightarrow{y} \in B, \xi \in [0,1], \xi \overrightarrow{x} + (1-\xi)\overrightarrow{y} \in B.$$

Therefore, $\forall \overrightarrow{x}, \overrightarrow{y} \in A \cap B, \gamma \in [0,1]$, $\gamma \overrightarrow{x} + (1-\gamma)\overrightarrow{y} \in A$, since $\overrightarrow{x}, \overrightarrow{y} \in A$; $\gamma \overrightarrow{x} + (1-\gamma)\overrightarrow{y} \in B$, since $\overrightarrow{x}, \overrightarrow{y} \in B$.
Therefore, it must be that $\gamma \overrightarrow{x} + (1-\gamma)\overrightarrow{y} \in A \cap B$.

(b) Suppose the otherwise that $\mathcal{P}$ is not convex, then

$$\exists \overrightarrow{x}, \overrightarrow{y}, \gamma \in [0,1], \text{s.t.} \gamma \overrightarrow{x} + (1-\gamma)\overrightarrow{y} \notin \mathcal{P}.$$

Let the part of segment not included in $\mathcal{P}$ be ℓ, then ℓ must form at least one triangle with the boundary of $\mathcal{P}$, lying on the exterior of $\mathcal{P}$. The sum of length of the other two sides of the triangle must be larger than that of ℓ, this is given by the triangle inequality. This contradicts with the definition of $\mathcal{P}$.

(c) Suppose there is a point $p \in \mathcal{P}$ and $p \notin P$. Then, we can decrease the perimeter by chopping off that corner with a new line segment. By contradiction, all vertex in $\mathcal{P}$ must be a point in P.

To show that two definitions for a convex hull are equivalent, we let $\mathcal{P}_1$ be the intersection of every convex set containing P, and let $\mathcal{P}_2$ be the smallest perimeter polygon containing P. We need to show that $\mathcal{P}_1 = \mathcal{P}_2$.

First, by part (b), the smallest perimeter polygon containing a set of points in convex. Hence, $\mathcal{P}_2$ is one of the sets in the intersection that makes up $\mathcal{P}_1$. Hence, $\mathcal{P}_1 \subseteq \mathcal{P}_2$.

Second, $\mathcal{P}_1$ is convex by extending part (a) to infinite intersections of convex sets. By part (c), any convex set containing the set of points P also contains the smallest perimeter polygon. Hence, since $\mathcal{P}_1$ is convex, it contains $\mathcal{P}_2$. In other words, $\mathcal{P}_2 \subseteq \mathcal{P}_1$. Hence, $\mathcal{P}_1 = \mathcal{P}_2$. □

Problem 4. Let P be a set of points in the plane. Let $\mathcal{P}$ be the convex polygon whose vertices are points from P and that contains all points in P. Prove that this polygon $\mathcal{P}$ is uniquely defined, and that it is the intersection of all convex sets containing P.

Proof. Let $\mathcal{Q}$ be another convex polygon whose vertices are all from P and contains all points in P. Since $\mathcal{Q}$ is distinct from $\mathcal{P}$, we can assume without loss of generality, that there is a point in $\mathcal{Q}$ which does not lie in $\mathcal{P}$; however, this implies that there is a vertex of $\mathcal{Q}$ which is not contained in $\mathcal{P}$ and contradicts the hypothesis. The case where there is a point in $\mathcal{P}$, which is not contained in $\mathcal{Q}$ is handled symmetrically. Thus, $\mathcal{P}$ is uniquely defined.

Let C denote the set of intersection of all convex sets containing the point set P. Let $x \in C$. Since $\mathcal{P}$ is a convex polygon, we must have $C \subseteq \mathcal{P}$ and hence $x \in \mathcal{P}$. Likewise, let $x \in \mathcal{P}$. If $x \notin C$, then a vertex of $\mathcal{P} \notin C$; however, this contradicts the hypothesis that C was the intersection of sets containing P. □

Problem 5. Let S be a set of n (possibly intersecting) unit circles in the plane. We want to compute the convex hull of S.

(a) Show that the boundary of the convex hull of S consists of straight line segments and pieces of circles in S.

(b) Show that each circle can occur at most once on the boundary of the convex hull.

(c) Let S' be the set of points that are the centers of the circles in S. Show that a circle in S appears on the boundary of the convex hull if and only if the center of the circle lies on the convex hull of S'.

(d) Give an $O(n \log n)$ algorithm for computing the convex hull of S.

(e) Give an $O(n \log n)$ algorithm for the case in which the circles in S have different radii.

Proof.

(a) By Mathematical Induction.
(i) (Base Case) The convex hull is S itself when $n = 1$.
(ii) (Inductive Step) Suppose the statement is true for k circles. Now, we insert the $k+1$-th circle in S. There are two cases to consider. First, the inserted circle may be included in the convex hull and thus remains the same. Second, if it lies outside of the hull, the convex hull must contain a part of this circle, tangent from this circle to some others already in the convex hull, and part of the previous convex hull. Each of them is either a straight line segment or a pieces of circles in S. Therefore, the statement is true.

(b) Suppose the otherwise that a circle occurs more than once on the the boundary of the convex hull.

Let this circle be u, and the neighbouring circles in the hull be v and w. Then, the bitangent of u and v, the bitangent of u and w must be on the convex hull and thus completely including u, which is a contradiction.

(c) Suppose that $s, t \in S$ are on the boundary of $CH(S)$. By the above, there is a line tangent to s and t supporting a half-plane containing all of S. This isn't possible if the line through the centers doesn't support a half-space containing all of the centers (since the circles are all the same radius). The converse follows from the same geometric argument.

(d) First, calculate the convex hull of S', which is done in $O(n \log n)$ time. Then, by (b), we know the structure and order of these circular arcs and segments forming S. By (a), we can calculate the outer-tangent (which lies on one consistent side of the convex hull of S') and outer-circular-arc (we know that the arc has angle less than 180°) between circles corresponding to the edge of S', which is constant time per edge.

(e) Apply Divide and Conquer. The merge algorithm rotates a parallel pair of supporting lines L_p and L_q, one for the set P and one for the set Q. At each iteration a decision is made regarding whether the current arc is extreme to both hulls. Then, we merge bridges with the two hulls, which is done in $O(n)$ time.

□

2 Line Segment Intersection

Problem 1. Let S be a set of n triangles in the plane. The boundary of the triangles are disjoint, but it is possible that a triangle lies completely inside another triangle. Let P be a set of n points in the plane. Give an $O(n \log n)$ algorithm that reports each point in P lying outside all triangles.

Solution. Apply the planesweep algorithm: every triangle's segments defines two sets of intervals containing a region in $\mathbb{R}^2$, we scan a vertical line from $x = -\infty$ to $x = +\infty$.

First we sort the x-coordinates of the triangle endpoints and points in P.

Note that the triangles lying fully in the interior of some other triangles are ignored. Then, the segments representing each intervals formed by the triangles must be disjoint.

We require a data structure supporting the following operations in $O(\log n)$ time each:

- Add a pair of segments, defining the range of an interval, formed by triangles;
- Delete the pair of segments;
- Report whether a point lies within any interval in the data structure.

This can be supported by a BBST, saving the linear functions instead of coordinates.

Upon meeting a new triangle, first check if it lies within some other triangles by query defined above. If not, then insert the two leftmost segments into the data structure. Once we meet the second point of the triangle, we remove these interval and add the right part of the triangle region defined by the rightmost two segments. The last point of the triangle will indicate a removal of the second added interval.

Upon meeting a point, check if this point lies within an interval. Report the point if it is not included in any of the intervals.

Problem 2. Let S be a set of n disjoint triangles in the plane. We want to find a set of $n-1$ segments with the following properties:

- Each segment connects a point on the boundary of one triangle to a point on the boundary of another triangle.
- The interiors of the segments are pairwise disjoint and they are disjoint from the triangles.
- Together they connect all triangles to each other, that is, by walking along these segments and the triangle boundaries it must be possible to walk from a triangle to any other triangle.

Develop a plane sweep algorithm for this problem that runs in $O(n \log n)$ time. State the events and data structures that you use explicitly, and describe the cases that arise and the actions required for each of them. Also state the sweep invariant.

Solution. We scan a vertical line from $x = -\infty$ to $x = +\infty$. Maintain a BBST for all edges that intersect with the sweepline. Maintain $helper(e)$ for all edges defined as the rightmost vertically visible vertex below the edge e.

To apply the variant of polygon triangulation algorithm, we need to add two auxilary lines $y = -\infty$ and $y = +\infty$.

Events

- Upon encountering the first point of a triangle, let e be the edge immediately above the point, connect this point to $helper(e)$. Insert the two neighboring edges to the BBST. Update the helpers correspondingly.
- Upon encountering the endpoint of a segment, delete the segment from BBST, update the helper for the edge immediately above.

Invariant: all triangles being swept are in one connected component.

Problem 3. Let P_1 and P_2 be two convex polygons. The vertices of each polygon are given to you in clockwise order in an array. Let n be the total number of vertices of P_1 and P_2. Suppose that each edge of P_1 shares at most one common point with an edge of P_2. Describe an algorithm to compute the intersection points of the edges of P_1 and P_2 in $O(n)$ time.

Solution. A convex polygon can be split into two x-monotone chains. We scan a vertical line from $x = -\infty$ to $x = +\infty$. Note that the number of edges intersecting with the sweepline is always constant. We only need to store the upper edge and lower edge intersected by the sweepline of the two polygons. Now, we just need to check if there is any intersection point. Since there are only constant number of possibilities, each event can be handled in $O(1)$ time. The total running time is $O(n)$ as required.

Problem 4. Let P_1 and P_2 be two convex polygons. The vertices of each polygon are given to you in counterclockwise order in an array. Let n be the total number of vertices of P_1 and P_2. Describe an algorithm to compute the vertices of P in counterclockwise order in $O(n)$ time.

Solution. We conclude by Problem 3 that all points on the intersection $P_1 \cap P_2$ can be reported in $O(n)$ time. The points are listed in lexicographical order by the sweepline algorithm, thus constructing the convex hull will only cost $O(n)$ time to give the required order of vertices in P.

Problem 5. Describe an $O(n \log n)$ time method for determining if two sets A and B of n points in the plane can be separated by a line.

Solution. First, we use $O(n \log n)$ time to compute the convex hull of A and B. Second, to check whether the two hulls intersect with each other, it suffices to verify that no two segments on different convex hulls intersect at a point.

Simply apply the sweep line algorithm. There are at most n line segments on the convex hull of A, and at most n line segments on the convex hull of B. Noting that the time complexity for the line segment intersection reporting algorithm is $O((n + I) \log n)$. Since we only care about the case where $I \neq 0$ and not the points of intersection, the algorithm can be done in $O(n \log n)$ time.

3 Polygon Triangulation

Problem 1. Prove that any polygon admits a triangulation, even if it has holes. Give a formula about the number of triangles in the triangulation.

Proof. (i) Let P have h holes and n vertices in total. The proof is by induction on h primarily, and n secondarily.

It is proven in the exercise that any simple polygon has a ear, and thus it can be proven by induction the basis of the induction for the case of $h = 0$.

For the general case, let d be a completely internal diagonal, whose existence can be guaranteed by the following argument which is used as the forementioned.

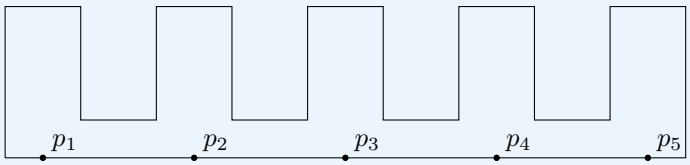
Choose an arbitrary convex vertex v_2, with neighbors v_1 and $v3$, on the outer boundary of P, and let $d = v_1v_3$ if d is internal, and otherwise let $d = v_2x$, where x is the closest vertex to v_2 measured perpendicular to v_1v_3. If d has one endpoint on a hole, then it increases n by 2, but decreases h by 1. If d has both endpoints on the outer boundary of P, then it partitions P into two polygons P_i with $n_i < n$ vertices and $h_i < h$ holes, $i = 1, 2$. In either case, the induction hypothesis applies and establishes the triangulation part of the theorem. □

(ii) Let the outer boundary of P have n_0 vertices, and let the ith hole have n_i vertices; thus $n = n_0 + n_1 + \ldots + n_h$. The sum of the interior angles of the outer boundary is $(n_0 - 2) \cdot 180$ degrees; the sum of the exterior angles of the ith hole is $(n_i + 2) \cdot 180$. Thus $180[(n_0 - 2) + (n_1 + 2) + \ldots + (n_h + 2)] = 180t$ or $t = n + 2h - 2$.

The same result may be obtained with Euler's Theorem. There are $V = n$ vertices, $F = t + h + 1$ faces, one for each triangle and hole, plus the exterior face, and $E = (3t + n)/2$ edges, where three per triangle plus the boundary counts each edge twice. Then $V - E + F = 2$ yields $t = n + 2h - 2$ as above. □

Problem 2. A rectilinear polygon is a simple polygon of which all edges are horizontal or vertical. Let $\mathcal{P}$ be a rectilinear polygon with n vertices. Give an example to show that $\lceil n/4 \rceil$ cameras are sometimes necessary to guard it.

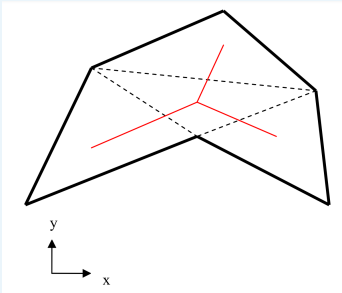
Proof.



Pictorially shown in the figure above, we can extend the construction to contain $n = 4k$ points, giving the $\lceil n/4 \rceil$ bound as required. □

Problem 3. Prove or disprove: The dual graph of the triangulation of a monotone polygon is always a chain, that is, any node in this graph has degree at most 2.

Solution. Here is a counterexample.



This polygon (shown in black bold lines) is monotone with respect to x-axis. Its triangulation is shown by black dash lines, and the dual graph shown by read lines. It's easy to see the dual graph has degree of 3 at the center node.

Problem 4. Give an algorithm that computes in $O(n \log n)$ time a diagonal that splits a simple polygon with n vertices into two simple polygons each with at most $\lfloor 2n/3 \rfloor + 2$ vertices. (Hint: Use the dual graph of a triangulation)

Solution. Computing the triangulation and its dual graph is done in $O(n \log n + n) = O(n \log n)$ time. Further, let $m = n - 2$ be the number of edges in the dual graph. We use dynamic programming to compute the size of every subtree. Now, we may decide how many nodes in the dual graph for split. To have at most $\lfloor 2n/3 \rfloor + 2$ vertices, we need to split the in the dual graph (which is a tree) such that the two components have at most $\lfloor 2(m+2)/3 \rfloor + 2 - 2 = \lfloor 2(m+2)/3 \rfloor$ nodes each.

Essentially, we need to argue that any tree admits a separation as the above required (and thus how the simple polygon is triangulated does no matter). The dual graph of a polygon triangulation has all of its nodes with degree at most 3. We can orient the tree rooted at the node corresponding to one of the polygon's ear such that the tree is a binary tree.

Now, note the piece of fact: in every binary tree of size $n_t > 1$, there is a node u with its subtree size ranging between $\lfloor (n_t+1)/3 \rfloor$ and $\lfloor (2n_t-1)/3 \rfloor$, which is what we wanted, now we let $n_t = m$.

Report any subtree with the feasible size in one depth-first search, which can be done in $O(m)$ time.

The final complexity is $O(n \log n + n + m + m) = O(n \log n)$.

Problem 5. Triangulate a set of n points, in $O(n \log n)$ time.

Solution. Apply Graham Scan. Place the point with the smallest y-coordinate into $p[1]$ and sort all other points counterclockwise around $p[1]$: $p[i]$ is larger than $p[j]$ if $p[i]$ is left of the directed line from $p[1]$ to $p[j]$.

Now, add edge $(p[1], p[2])$ and push $p[1], p[2]$ to the stack.

The remaining points is processed as below:

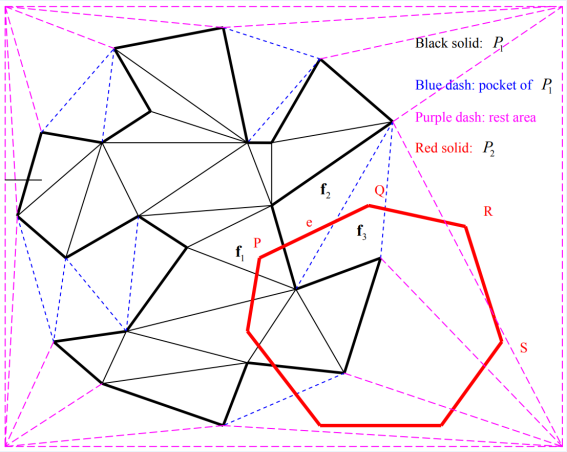
- add $(p[1], p[i])$ and $(p[i-1], p[i])$ to t
- while from the last edge of the convex hull there is no left turn to $p[i]$:
  - remove the point on the top of the stack
  - add $(p[1], top(q))$ to t
- add $p[i]$ to q

Let h be the number of points on the convex hull of P. There can be $e = 3n - 3 - h$ edges and $f = 2n - 2 - h$ triangles and thus the triangulation process ends in $O(n)$ time as required.

Sorting costs at most $O(n \log n)$ time.

Problem 6. The pockets of a simple polygon are the areas outside the polygon, but inside its convex hull. Let $\mathcal{P}_1$ be a simple polygon with m vertices, and assume that a triangulation of $\mathcal{P}_1$ as well as its pockets is given. Let $\mathcal{P}_2$ be a convex polygon with n vertices. Show that the intersection $\mathcal{P}_1 \cap \mathcal{P}_2$ can be computed in $O(m+n)$ time.

Solution.



Suppose we have a sufficiently large rectangle covering $\mathcal{P}_1$, the pocket of $\mathcal{P}_1$ and the convex polygon $\mathcal{P}_2$. It is categorized into 3 regions: triangulation inside $\mathcal{P}_1$ (black solid lines), triangulations outside $\mathcal{P}_1$ but inside the convex hull (or equivalently, inside the pockets, in blue dash lines), and triangulations of the rest of the area (pink dash lines), all of which can be stored by DCELs. The number of triangles inside the rectangle is bounded by $(n-2)+(n+4+2-2) = O(n)$.

Step 1: Arbitrarily pick an edge on $\mathcal{P}_2$ and pick one of its endpoint as P. As there are only n triangles, we can locate P using at most $O(n)$ point in triangle operations. Let the face containing P be $\mathbf{f}_1$.

Step 2: March from P along the edge e, decide whether e intersects with $\mathbf{f}_1$. Note that we can make decision in $O(1)$ time since $\mathbf{f}_1$ is a triangle. There are two conditions:

- If e does not intersect with $\mathbf{f}_1$, then advance and take the other endpoint Q on the edge e and check the next edge on $\mathcal{P}_2$.
- Otherwise, e crosses into a new face $\mathbf{f}_2$, we can easily detect this face by calling $\mathbf{f}_1$`.Twin()`. Now we can check whether e intersects with $\mathbf{f}_2$.

We do this until meeting the point Q, and so we have detected all faces intersecting with e. Suppose the intersection happens k times, the time complexity in this step is $O(1+k)$.

Continue this process of checking if an edge intersects with a face, marching along the polygon $\mathcal{P}_2$ from Q to R, R to S, and so on, until it returns to P. The process costs in total $O(m + \sum_{i=1}^{m} k_i)$ time, $\sum_{i=1}^{m} k_i$ is the total number intersection number $\mathcal{P}_2$ has to the triangulation of the whole triangulation areas.

Step 3: With all intersection points, and their face belonging information to $\mathcal{P}_1$, its pocket, or the rest, we can perform overlay actions, to get the subdivisions of $\mathcal{P}_1 \cap \mathcal{P}_2$ in $O(k)$ time.

Any triangle can intersect with a convex polygon with at most 6 times. The intersection of triangulation of $\mathcal{P}_1$ with its pocket can intersect for at most $6 \cdot 2(n-2)$ time, which means k is still in $O(n)$. Therefore, the algorithm is in $O(n+m)$ time.

Problem 7. The stabbing number of a triangulated simple polygon $\mathcal{P}$ is the maximum number of diagonals intersected by any line segment interior to P. Give an algorithm that computes a triangulation of a convex polygon that has stabbing number $O(\log n)$.

Solution. For simplicity assume that $n = 2^k$.

Construct a recursive triangulation as follows:

- Connect point i and point $i + 2$;
- Recursively triangulate the remaining untriangulated region with $n/2$ vertices until $n = 3$.

For each layer of recursion, a line can at most stab two diagonals. This gives the stabbing number $O(\log n)$ as required.

Proof: Let $\ell^\perp$ be a line orthogonal to the segment. The monotonicity of convex polygons ensures that the diagonals added are also monotone w.r.t. $\ell^\perp$. Split the untriangulated polygon into two chains w.r.t to $\ell^\perp$. Once an edge on a chain intersects with the segment, we know immediately that no other edges on the chain can intersect with the segment since it is either on the left or on the right of the segment. This gives $O(1)$ number of intersecting segments as required.

Problem 8. Given a simple polygon $\mathcal{P}$ with n vertices and a point p inside it, show how to compute the region inside $\mathcal{P}$ that is visible from p.

Solution. Use the sweepline algorithm in Exercise 1 Problem 6. When we mark the visible segments, record the current angle and the segment visible. Now we know the entire region bounded by these segments described by the record.

4 Closest Pair

Problem 1. Let P be a set of points in $\mathbb{R}^d$. Give an $O(n \log n)$ expected time algorithm to find the 2nd closest pair of P. Formally, define $T = \{\{p, q\} | p, q \in P \wedge p \neq q\}$. The 2nd closest pair is the $\{p, q\} \in T$ that has the second smallest $dist(p, q)$ (i.e., Euclidean distance between p, q).

Solution. Let $(p_{l1}, p_{l2}), (p_{r1}, p_{r2})$ be the closest and second closest pair resp. returned by left and right subproblem in the algorithm. Let d be the distance of the pair with the second smallest distance among $p_{l1}, p_{l2}, p_{r1}, p_{r2}$, and impose the grid with side length $\ell = d/\sqrt{2}$. Let c_1 be any cell on the left of the spliting hyperplane. By packing lemma, at most $O(1)$ cells c_2 will be covered and the number of points in a grid cell will be no larger than 2, except of that containing the closest pairs p_{l1}, p_{r1}. Therefore, the number of points in a grid cell is still bounded by $O(1)$.

There can be in total $O(4n) = O(n)$ non-empty cells, incurring $O(n)$ cross pairs to be considered for the merge algorithm, giving $O(n \log n)$ time as required.

Problem 2. Given a set of points P in $\mathbb{R}^d$ and a fixed distance $r > 0$. Report all pairs of distinct points $p, q \in P$ such that $|pq| \leq r$, in $O(n + k)$ expected time, where k is the number of pairs.

Solution. Impose an axis-parallel grid, where all the cells are square-shaped, with the side length $r/\sqrt{d}$. Now, we can create a link list to access all points in a specified cell, in $O(n)$ time expected by hashing. Now, for every cell, we enumerate its r-neighbors (including itself) and report the all cross-point-pairs(referring to distinct points) that satisfies the requirement.

Now, we argue that this algorithm gives the desired time complexity bound $O(n + k)$. Hashing obviously takes $O(n)$ time expected. Surprisingly, the cross-pair cells point pairs are also computed in $O(n + k)$ time. Let $c_1, c_2, \ldots, c_m$ be the non-empty cells, for some $m \geq 1$. Define $n_i = |c_i(P)|$, which is the number of points covered by c_i, for each $i \in [1, m]$. The cost of is then

$$\begin{aligned}
&\sum_c |c(P)|^2 + \sum_{\substack{c_1 \neq c_2, \\ c_2 \text{ is the } r\text{-neighbour of } c_1}} |c_1(P)||c_2(P)| \\
&\geq \sum_c \left(\frac{|c(P)|(|c(P)| - 1)}{2} + \frac{|c(P)|}{2} \right) + \sum_{\substack{c_1 \neq c_2, \\ c_2 \text{ is the } r\text{-neighbour of } c_1}} \frac{|c_1(P)|^2 + |c_2(P)|^2}{2} \\
&\leq O(k + n + k + n) \leq O(n + k)
\end{aligned}$$

by noticing that $\sum_{i=1}^{m} n_i \geq n$ and the number of r neighbours is $O\left(\left(1 + \lceil \frac{2(r+r)}{r/\sqrt{d}} \rceil\right)^d\right) = O(1)$.

5 Well-Separated Point Decomposition

Problem 1. Let $s > 0$ and let $x = 2/s + 1$. Further, let $S = \{x^i | 0 \leq i \leq n - 1, i \in \mathbb{N}\}$ and let $\{A_j, B_j\}(1 \leq j \leq m)$ be an arbitrary s-WSPD for S. Show that

$$\sum_{j=1}^{m} (|A_j| + |B_j|) = \binom{n}{2} + m.$$

Hint: For each j at least one of both sets A_j and B_j is a singleton.

Proof. We want to prove that for any $i < j$, $s \cdot (x^i - x^j) = s \cdot (x^j(x^{i-j} - 1)) \geq 2x^j$. Then, $\{x^i, x^j\}$ and any set with elements smaller than x^j and size larger than 2 is not well separated.

Note that $sx^d - s \geq sx - s \geq 2$. Therefore, $s \cdot (x^i - x^j)/2 \geq x^j$. For each j, either A_j or B_j is a singleton. Since

$$\bigcup_{k=1}^{m} (A_k \otimes B_k) = P \otimes P,$$

we have

$$\left| \bigcup_{k=1}^{m} (A_k \otimes B_k) \right| = \sum_{k=1}^{m} |A_k \otimes B_k| = \sum_{k=1}^{m} |A_k|\,|B_k| = |P \otimes P| = \binom{n}{2},$$

and thus

$$\sum_{k=1}^{m} (|A_k| + |B_k|) = \sum_{k=1}^{m} (|A_k||B_k| + 1) = \binom{n}{2} + m$$

□

Problem 2. Let P be a set of n points in $\mathbb{R}^d$. Let $p \in P$ and let $q \in P$ be the next neighbor of p in P, i.e. $|pq| = \min\{|pr| : r \in P, r \neq p\}$. Consider an arbitrary s-WSPD for P with $s > 2$.

1. Let $\{A, B\}$ be a pair in this decomposition and assume that p lies in A and q lies in B. Show that A only contains p.
2. Show that the size of an arbitrary s-WSPD with $s > 2$ is at least $n/2$.

Proof.

1. Suppose the otherwise that p' also lies in A. Then, let r be the size of the ball containing A, $|pp'| \leq 2r < sr \leq |pq|$, which is a contradiction.
2. Let q_p be the next neighbor of any point $p \in P$. There cannot be $A_i \otimes B_i$ with $\{p, q_p\}, \{p', q_{p'}\}, \{p'', q_{p''}\}$ where p, p', p'' are distinct. WLOG, let $p \in A_i, q_p \in B_i$, then A_i must be a singleton. Then, $q_{p'} = p$ and B_i should be a singleton as well, this implies $\{p, q_p\} = \{p', q_{p'}\}$ and p'' is either p or p', which is a contradiction. No three pairs containing a point and its next neighbor can be contained in a set decomposition, the size of an arbitrary s-WSPD with $s > 2$ is at least $n/2$.

□

6 Duality and Arrangements

Problem 1. Let S be a set of n segments in the plane. We want to preprocess S into a data structure that can answer the following query: Given a query line ℓ, how many segments in S does it intersect?

(a) Formulate the problem in the dual plane.

(b) Describe a data structure for this problem that uses $O(n^2)$ expected storage and has $O(\log n)$ expected query time.

(c) Describe how the data structure can be built in $O(n^2 \log n)$ expected time.

Solution.

(a) S^* becomes a set of n left-right wedges in the plane. For every query line ℓ, we count the number of wedges containing the point ℓ^*.

(b) Each wedge is defined by two lines, which constitutes two disjoint left and right regions. These lines form an arrangement $\mathcal{A}(L)$, where $|L| \leq 2n$, and hence, it has complexity $O(n^2)$. Thus, each region represents a set of point queries which are covered by the same set of left-right wedges. A point location structure under this situation can be built with $O(n^2)$ expected storage, and has $O(\log n^2) = O(\log n)$ expected query time.

(c) Sweep a vertical line ℓ from left to right.
Color the two lines of each wedge with different colors, namely black if it is bounding the left-wedge from the lower side and white otherwise, bounding the left-wedge from the upper side.
We can sort the lines by their slope in ascending order so that the line sweeping from left is intersecting with the leftmost unbounded regions (there are at most $2n$ such regions). Now, we insert these lines to the sweepline status $\sum$. The count of each region is always answered by issuing a query for the number of white lines subtracted by the number of black lines

above the current query point $y = y_0$, which is solvable using a BBST.
There are at most $k = O(n^2)$ intersection points. The normal sweepline algorithm will spend $O(n^2 \log n)$ time handling all regions. Assuming general position that the set S has no two segments sharing the same point, only one region will be created after the intersection event. If the two lines are from the same wedge, alternate their colors. For the count of this new region, issue a query call for this new region with the query point on the lower line intersecting with ℓ, in $O(\log n)$ time. This does not change the time complexity of the sweepline algorithm.
We have calculated the counts for all regions within $O(n^2 \log n)$ time as desired.

Problem 2. Let S be a set of n segments in the plane. A line ℓ that intersects all segments of S is called a *transversal* or *stabber* for S.

(a) Give an $O(n^2)$ algorithm to decide if a stabber exists in S.

(b) Now assume that all segments are vertical. Give a randomized algorithm with $O(n)$ expected running time that decides if a stabber exists for S.

Solution.

(a) Assume the general position that no line segment in S is vertical.
Let the lower point of the n segments form set A, the upper point of the n segments form set B.
Consider A, B and the stabber ℓ the dual space. By the order property, ℓ^* is lower than all lines in A^*, and higher than all lines in B^*. It is easy to compute if the described region exists in $O(n^2)$ time by halfplane intersection and convex polygon intersection.

(b) We can formally describe S as the set of vertical segments $\ell_i = \{(x_i, y) : y_{i,1} \leq y \leq y_{i,2}\}$. Then, we can establish a linear program in 2D with $2n$ constraints as follows:

$$\begin{aligned} &\textbf{minimize} && a \\ &\textbf{subject to} && y_{i,1} \leqslant ax_i + b \leqslant y_{i,2},\ i = 1,\ 2,\ ...,\ n. \end{aligned}$$

If the linear program returns feasible, a stabber exists. Otherwise, a stabber is impossible. This is a randomized algorithm with $O(n)$ expected running time.

7 Point Location

Problem 1. Show that, given a planar subdivision $\mathcal{S}$ with n vertices and edges and a query point q, the face of $\mathcal{S}$ containing q can be computed in $O(n)$ time. Assume that $\mathcal{S}$ is given in doubly-connected edge list.

Solution. Shoot vertical lines through every vertex and split the subdivision into vertical slabs. In $O(\log n)$ time by binary search with x-coordinate of q, we can identify the slab where q lies within. Record all the segments intersecting with the slab. Now within the slab we can judge for every trapezoid to see whether q is included. The basic operation in this process is to judge whether q lies above, on, or below a line.

Problem 2. A polygon $\mathcal{P}$ is called *star-shaped* if a point p in the interior of $\mathcal{P}$ exists such that, for any other point q in $\mathcal{P}$, the line segment $\overline{pq}$ lies in $\mathcal{P}$. Assume that such a point p is given with the star-shaped polygon $\mathcal{P}$. Given the polygon $\mathcal{P}$ with vertices sorted in order along the boundary in an array. Show that, given a query point q, it can be tested in $O(\log n)$ time whether q lies inside $\mathcal{P}$. What if $\mathcal{P}$ is star-shaped, but the point p is not given?

Solution. The point p visible to all other vertices is given. As the polygon is star-shaped, the ray pp_i partitions the plane into n polar angle slabs in order. Thus, we can apply a binary search in $O(\log n)$ time to locate the slab in between $\overline{pp_i}, \overline{pp_{i+1}}$ containing q. Now, it should be clear that we can verify if q is in $\triangle pp_ip_{i+1}$ in $O(1)$ time.

For the case that p is not given, we can form an 2D LP instance for the feasible region of p. This can be computed in $O(n)$ expected time.

8 Linear Programming

Problem 1. Suppose we want to find *all* optimal solutions to a 3-dimensional linear program with n constriants. Argue that $\Omega(n \log n)$ is a lower bound for the worst-case complexity of any algorithm to solving this problem.

Proof. It is well known that finding a convex hull in $\mathbb{R}^2$ requires $\Omega(n \log n)$ time.

Suppose we have an algorithm for finding all optimal solution for 3D LP better than $O(n \log n)$. We can solve the 2D convex hull problem with input $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ in time better than $O(n \log n)$, which is a contradiction. Specifically, we construct two 3D LPs with the following constraints:

$$\begin{aligned} &\textbf{minimize} && z \\ &\textbf{subject to} && y \leqslant x_i x - y_i,\ i = 1,\ 2,\ ...,\ n, \\ &&& z \geq 1. \end{aligned}$$

$$\begin{aligned} &\textbf{minimize} && z \\ &\textbf{subject to} && y \geqslant x_i x - y_i,\ i = 1,\ 2,\ ...,\ n, \\ &&& z \geq 1. \end{aligned}$$

This returns two envelopes on $z = 1$. By the duality transformation, we know that these two envelopes corresponds to the upper and lower hull of the n input points. □

Problem 2. Let H be a set of at least three halfplanes with a non-empty intersection such that not all bounding lines are parallel. We call a halfplane $h \in H$ *redundant* if it does not contribute an edge to $\cap H$. Prove that for any redundant halfplane $h \in H$ there are two halfplanes $h', h'' \in H$ such that $h' \cap h'' \subset h$. Give an $O(n \log n)$ time algorithm to compute all redundant halfplanes.

Solution. As $\cap H \cap h = \cap H$ and $\cap H \neq \emptyset$, we can sweep h toward the intersection. The first touched point of $\cap H$ is the intersection point of some edge h' and h''.

We first sort all halfplanes by their polar angle, and maintain a deque. For every halfplane, if the last intersection point formed by the front(rear) elements in the deque is on the opposite side of the current processing halfplane, we see the front(rear) halfplane as redundant in the deque.

Problem 3. Prove that RandomPermutation(A) is correct, that is prove that every possible permutation of A is equally likely to be the output. Also show that the algorithm is no longer correct if we change k in line 3 to n.

```
1: procedure RandomPermutation(A)
2:     for k ← n downto 2 do
3:         rndIndex ←Random(k).
4:         Exchange A[k] and A[rndIndex].
5:     end for
6: end procedure
```

Proof. WLOG, suppose that the original sequence $A = (1, 2, \ldots, n)$. Now we prove that any permutation of A, resulting in $(b_1, b_2, \ldots, b_n)$, can be generated with probability $1/n!$.

Denote the event Π_i as having $B_j = b_j$ for every $n \geq j \geq i$.

$$\mathbf{Pr}[\Pi_1] = \mathbf{Pr}[B_n = b_n] \cdot \mathbf{Pr}[B_{n-1} = b_{n-1}|\Pi_n] \cdot \mathbf{Pr}[B_{n-2} = b_{n-2}|\Pi_{n-1}] \ldots \mathbf{Pr}[B_1 = b_1|\Pi_2].$$

First, $\mathbf{Pr}[B_n = b_n] = 1/n$. Having $B[n]$ fixed, $B[n-1]$ can only come from the $n-1$ remaining elements, thus $\mathbf{Pr}[B_{n-1} = b_{n-1}|\Pi_n] = 1/(n-1)$. By the same reasoning, $\mathbf{Pr}[B_{n-j} = b_{n-j}|\Pi_{n-j+1}] = 1/(n-j)$. Thus, $\mathbf{Pr}[\Pi_1] = 1/n!$.

If we change k in line 3 to n, we would end up with having n^n possible results. When $n = 3$, there are 27 possible results with equal probability but the number of permutations is 6, which is impossible, as 27 is not divisible by 6. □

Problem 4. Below is a paranoid algorithm to compute the maximum of a set A of n real numbers:

```
1: procedure ParanoidMaximum(A)
2:     if card(A) = 1 then
3:         return the unique element x ∈ A
4:     else
5:         Pick a random element x from A.
6:         x′ ←ParanoidMaximum(A\{x}).
7:         if x ≤ x′ then
8:             return x′
9:         else
10:            Now we suspect that x is the maximum, but to be absolutely sure, we compare x
    with all card(A) − 1 other elements of A.
11:            return x
12:        end if
13:    end if
14: end procedure
```

What is the worst-case running time of this algorithm? What is the expected running time (with respect to the random choice in line 3?)

Solution. The worst-case running time of this algorithm is $O(n^2)$ if the random element x is picked in descending order, falling into the case where $x > x'$.

Every element has equal probability to be picked. Thus, we fix the sequence $A = (a_1, a_2, \ldots, a_n)$ indicating the elements getting picked in the i-th stage of recursion. Denote the event B_i by "a_i is the largest element among all a_j with $j > i$".

By backward analysis, $\mathbf{Pr}[B_i|a_1, a_2, \ldots, a_{i-1}] = 1/(n-i+1)$, and thus

$$\begin{aligned}\mathbf{Pr}[B_i] &= \sum_{a_1,a_2,\ldots,a_{i-1}} \mathbf{Pr}[B_i|a_1, a_2, \ldots, a_{i-1}] \cdot \mathbf{Pr}[a_1, a_2, \ldots, a_{i-1}] \\ &= (i-1)! \cdot \frac{1}{(i-1)!} \cdot \frac{1}{(n-i+1)} \\ &= \frac{1}{(n-i+1)}.\end{aligned}$$

By total probability law, the expectation of the running time of this algorithm is

$$\mathbf{E}[T] = \sum_{i=2}^{n} (\mathbf{Pr}[B_i] \cdot (n-i+1) + (1 - \mathbf{Pr}[B_i]) \cdot 1) = O(n).$$

Problem 5. On n parallel railway tracks n trains are going with constant speeds $v_1, v_2, \ldots, v_n$. At time $t = 0$ the trains are at positions $k_1, k_2, \ldots, k_n$. Give an $O(n \log n)$ algorithm that detects all trains that at some moment in time are leading. To this end, use the algorithm for computing the intersection of halfplanes.

Solution. We can represent the position of the trains by $y_i := k_i + v_i t$. The trains that are leading forms an upper envelope once we put the line equations on a graph. We can compute in $O(n \log n)$ time the upper envelope by intersection of the n halfplanes $y_i \geq 0$ and identify the edges on the envelope corresponds to the n trains.

Problem 6. Let R be a set of n red points in the plane, and let B be a set of n blue points in the plane. We call a line ℓ a *separator* for R and B if ℓ has all points of R to one side and all points of B to the other side. Give a randomized algorithm that can decide in $O(n)$ expected time whether R and B have a separator.

Solution. We can form two 2D instances. If neither of the LP instances give a solution, we report impossible. Otherwise, the LP provides a possible separator ℓ.

$$\begin{aligned}&\textbf{minimize} && a \\ &\textbf{subject to} && y_{B_i} \leqslant a x_{B_i} + b,\ i = 1,\ 2,\ \ldots,\ n, \\ &&& a x_{R_i} + b \leqslant y_{R_i},\ i = 1,\ 2,\ \ldots,\ n.\end{aligned}$$

$$\begin{aligned}&\textbf{minimize} && a \\ &\textbf{subject to} && y_{R_i} \leqslant a x_{R_i} + b,\ i = 1,\ 2,\ \ldots,\ n, \\ &&& a x_{B_i} + b \leqslant y_{B_i},\ i = 1,\ 2,\ \ldots,\ n.\end{aligned}$$

9 Voronoi Diagram

Problem 1. Prove that for any $n > 3$ there is a set of n point sites in the plane such that one of the cells of $\mathrm{Vor}(P)$ has $n - 1$ vertices.

Solution. Let $P = \{p_1, p_2, \ldots, p_n\}$ be the n points, and we construct the sites as follows:

- $p_1 = (0, 0)$;
- $p_i = (\cos \frac{2\pi(i-1)}{n-1}, \sin \frac{2\pi(i-1)}{n-1})$, these points lies on a unit circle.

Consider the circle $\odot O$ having $\overline{p_1 p_i} (i \geq 2)$ as diameter. We can see that no other p_j $(j \geq 2, j \neq i)$ can lie on the interior of $\odot O$ (as $|\overline{p_1 p_j}| = |\overline{p_1 p_i}| = \mathrm{diam}$ and $p_i \neq p_j$), and thus there must be an edge between p_1 and p_i.

Therefore, there are $n - 1$ edges bounding the Voronoi cell of p_1.

Problem 2. Show that $\Omega(n \log n)$ is a lower bound for computing Voronoi diagrams by reducing the sorting problem to the problem of computing Voronoi diagrams. You can assume that the Voronoi diagram algorithm should be able to compute for every vertex of the Voronoi diagram its incident edges in cyclic order around the vertex.

Solution. Let $A = (a_1, a_2, \ldots, a_n)$ be the array to be sorted. Then we can let $P = \{(a_1, 1), (a_2, 1), \ldots, (a_n, 1)\}$ to be the set of n points in $\mathbb{R}^2$. Any algorithm that solves the computation of Voronoi diagrams better than $O(n \log n)$ implies an algorithm that solves the sorting problem better than $O(n \log n)$ time.

Problem 3. Let P be a set of n points in the plane. Give an $O(n \log n)$ time algorithm to find two points in P that are closest together. Show that your algorithm is correct.

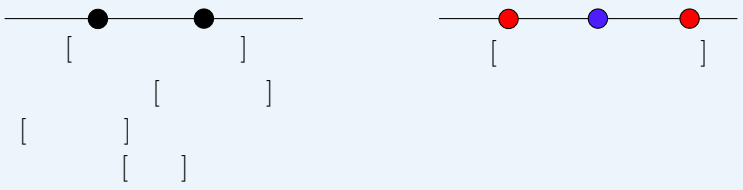
Solution. Compute first the Voronoi diagram of the n points in $O(n \log n)$ time. Now, the closest pair of sites in P are neighbors in the Delaunay triangulation, which corresponds to two neighboring Voronoi cells. Therefore, it suffices to check every edge of the Voronoi diagram and the points corresponding to the two cells separated by the edge. There are only $O(n)$ edges and thus the closest pair problem is solved in $O(n \log n)$ time.

10 Vapnik-Chervonenkis Dimension

Problem 1. Let $\mathcal{H}_{int}$ be the class of intervals in $\mathbb{R}$. Prove: $\dim_{\text{VC}}(\mathcal{H}_{int}) = 2$.

Proof.

- $\dim_{\text{VC}}(\mathcal{H}_{int}) \geq 2$. Illustrated as below.
- $\dim_{\text{VC}}(\mathcal{H}_{int}) < 3$. Consider the labelling of any 3 points excluding the middle one (shown as blue). Any interval including the two red points must also include the blue point.



□

Problem 2. Let $\mathcal{H}^d_{rec}$ be the class of axis-parallel hyperrectangles in $\mathbb{R}^d$. Prove: $\dim_{\text{VC}}(\mathcal{H}^d_{rec}) = 2d$.

Proof.

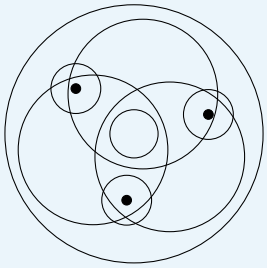
- $\dim_{\text{VC}}(\mathcal{H}^d_{rec}) \geq 2d$. Consider a set S of $2d$ points where each point only has one of the d dimensions set to either 1 or -1 and 0 for all other dimensions. It is easy to see that any subset of these points can be shattered by an axis-parallel hyperrectangle. Initially, set R to include no point. Let $T \subseteq S$ and for any point $p \in T$, we enlarge the hyperrectangle in the dimension which has coordinate of p being non-zero on that dimension, to include p. This operation does not affect inclusion and exclusion of any other points.
- $\dim_{\text{VC}}(\mathcal{H}^d_{rec}) < 2d + 1$. Let R be the minimum bounding hyperrectangle of the point set S. Now, at least one point is not the extreme points of the hyperrectangle. If we label this point negative, then by the minimality of the bounding rectangle, no other rectangles can exclude these points.

□

Problem 3. Let $\mathcal{H}_{disk}$ be the class of disks in $\mathbb{R}^2$. Prove: $\dim_{\text{VC}}(\mathcal{H}_{disk}) = 3$.

Proof.

- $\dim_{\text{VC}}(\mathcal{H}^d_{disk}) \geq 3$. See the illustration below.



- $\dim_{\text{VC}}(\mathcal{H}^d_{disk}) < 4$. Let C be the minimum enclosing ball of the point set S. Now, either there are four points that are cocircular or there is at least one point in the interior of the circle. For the former case, we label one of the points on the boundary of the circle as negative and no other circle can exclude this point without losing other points by minimality. For the latter case, label the interior points as negative. Since we at least have to bound the points on the boundary, there is no way we can exclude the interior points.

□

Problem 4. Let $\mathcal{H}_{half}$ be the class of halfspaces in $\mathbb{R}^d$. Prove: $\dim_{\text{VC}}(\mathcal{H}^d_{half}) = d + 1$.

Proof.

- $\dim_{\text{VC}}(\mathcal{H}^d_{half}) \geq d + 1$. Pick a good set of $d + 1$ points: the original and all points with a 1 in one coordinate and zeros in the rest (that is all neighbors of the origin on the Boolean cube). Let p_i be the point with a 1 in the ith coordinate. Suppose we wish to partition this set into two pieces S_1 and S_2 with a hyperplane (say the origin is in S_1), then choose the hyperplane
$$\sum_{\{i:p_i \in S_2\}} x_i = 1/2.$$

- $\dim_{\text{VC}}(\mathcal{H}^d_{half}) < d + 2$.
Let Q be any set of $d + 2$ points. Radon's theorem implies that if a set Q of $d + 2$ points is being shattered by $\mathcal{H}^d_{half}$ then we can partition this set Q into two disjoint sets Y and Z such that $\text{CH}(Y) \cap \text{CH}(Z) \neq \emptyset$. In particular, let s be a point in $\text{CH}(Y) \cap \text{CH}(Z)$. If a halfspace h contains all points of Y, then $\text{CH}(Y) \subseteq h$, since a halfspace is a convex set. Thus any halfspace h containing all points of Y will contain the point $s \in \text{CH}(Y)$. But $s \in \text{CH}(Z) \cap h$, and this implies that a point of Z must lie in h, by the lemma below. Namely, the subset $Y \subseteq Q$ cannot be realized by a halfspace, which implies that Q cannot be shattered.

Lemma. Let $P \subseteq \mathbb{R}^d$ be a finite set, let s be any point in $\text{CH}(P)$, and let h be a halfspace of $\mathbb{R}^d$ containing s. Then there exists a pont of P contained inside h.

Proof. The halfspace h can be written as $h = \{t \in \mathbb{R}^d | \langle t, v \rangle \leq c\}$. Now $s \in \text{CH}(P) \cap h$, and as such there are $\alpha_1, \alpha_2, \ldots, \alpha_m \geq 0$ and points $p_1, p_2, \ldots, p_m \in P$, such that $\sum_i \alpha_i = 1$ and $\sum_i \alpha_i p_i = s$.

By the linearity of the dot product, we have that

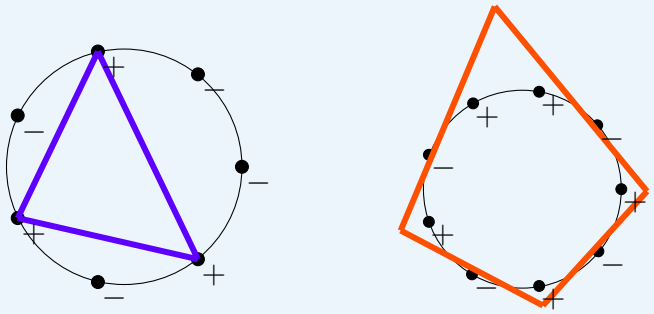
$$s \in h \Rightarrow \langle s, v\rangle \leq c \Rightarrow \left\langle \sum_{i=1}^{m} \alpha_i p_i, v \right\rangle \leq c \Rightarrow \beta = \sum_{i=1}^{m} \alpha_i \langle p_i, v\rangle \leq c.$$

Setting $\beta_i = \langle p_i, v\rangle$, for $i = 1, 2, \ldots, m$, the above implies that β is a weighted average of $\beta_1, \beta_2, \ldots, \beta_m$. In particular, there must be a β_i that is no larger than the average. That is $\beta_i \leq c$. This implies that $\langle p_i, v\rangle \leq c$. Namely, $p_i \in h$ as claimed. □

Problem 5. Let $\mathcal{H}_{poly}$ be the class of convex d-polygons in $\mathbb{R}^2$. Prove: $\dim_{\text{VC}}(\mathcal{H}^d_{poly}) = 2d+1$.

Proof.

- $\dim_{\text{VC}}(\mathcal{H}_{poly}) \geq 2d+1$.We can induce any labeling of any $2d+1$ points on a circle using a d-gon as follows: if there are more negative labels than positive labels, use the positive points as the vertices as shown in the figure. Otherwise, use tangents to the negative points as edges.



- $\dim_{\text{VC}}(\mathcal{H}_{poly}) < 2d+2$. Given any point set with size $2d+2$. If one of the points is inside the convex hull of the rest, then it is not possible to label that point negative and the rest positive. Otherwise, it is not possible to label them in alternating $+, -, +, -, +, -, \ldots, +, -$ order.

□

11 ε-net

Problem 1. Design an probabilistic algorithm to compute a small point set N such that the minimum enclosing ball bounds all but at most $\varepsilon|A|$ points from an n-point set $A \subseteq \mathbb{R}^d$ with probability at least $1 - e^{-\lambda}$.

Solution. Let $\mathcal{B}_d$ be the family of d dimensional balls. The VC dimension of the set system $(\mathbb{R}^d, \mathcal{B}_d)$ has the same dimension, $d+1$, as the set system $(\mathbb{R}^d, \mathcal{B}_d^{\text{comp}})$, where $\mathcal{B}_d^{\text{comp}}$ is the complement of all the closed d dimensional balls. We can obtain an ε-net with probability $1 - e^{\lambda}$ and size $O\left(\left\lceil \frac{(d+1)\ln n + \lambda}{\varepsilon} \right\rceil\right)$ with respect to $\mathcal{B}_d^{\text{comp}}$.

Let $B \in \mathcal{B}_d$ be the minimum bounding ball of the random sampling S from A. Since $\mathcal{B}_d^{comp} \cap S = \emptyset$, this implies that less than $\varepsilon|A|$ points of A are out of B. Otherwise, by definition, $\mathcal{B}_d^{comp} \cap A \neq \emptyset$.