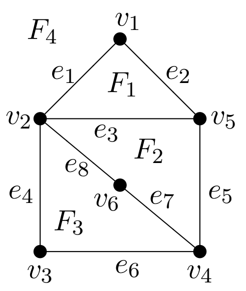
CSCE-620/VIZA-670 Computational Geometry

Fall 2025

Instructor: Dr. Jianer Chen
Office: PETR 428
Phone: 845-4259
Email: `chen@cse.tamu.edu`
Office Hours: MWF 2:40 pm–3:40 pm

Solution to Assignment # 1

1. (Geometric Data Structures I) Show the Doubly-Connected-Edge-List structure for the following PSLG.



Solution.

	V_1	V_2	F_1	F_2	N_1	N_2
e_1	v_1	v_2	F_1	F_4	e_2	e_4
e_2	v_1	v_5	F_4	F_1	e_1	e_3
e_3	v_2	v_5	F_1	F_2	e_1	e_5
e_4	v_2	v_3	F_3	F_4	e_8	e_6
e_5	v_4	v_5	F_2	F_4	e_7	e_2
e_6	v_3	v_4	F_3	F_4	e_4	e_5
e_7	v_4	v_6	F_3	F_2	e_6	e_8
e_8	v_2	v_6	F_2	F_3	e_3	e_7

2. (Polygon Intersection) Develop an $O(n \log n)$-time algorithm for the following problem: given two polygons P_1 and P_2, determine if P_1 and P_2 intersect at their edges.

Solution. This problem is obviously related to the segment intersection problem where we are asked to report all intersections of n line segments. As given in the class, the segment intersection problem can be solved in time $O((n+m)\log n)$, where m is the number of segment intersections. The algorithm has this time complexity because at each event point, which can be either an end of a segment or an intersection of two segments, we need to do a constant number of insertion and deletion operations on the sweeping line. Each such an operation takes time $O(\log n)$, which gives the time complexity $O((n+m)\log n)$ for the algorithm.

For the polygon intersection problem, as given in this question, we can modify the segment intersection algorithm such that once we encounter the first intersection, we simply stop and report. Thus, we have only $2n$ event points that are the ends of the n edges of the two given polygons, and the time complexity of the algorithm is reduce to $O(n \log n)$.

Let e_1 and e_2 be two edges of the polygons P_1 or P_2. We say that e_1 and e_2 (P_1, P_2)-*intersect* if they belong to different polygons and intersect.

The algorithm is given as follows. Compared with the segment intersection algorithm, here the algorithm removes the case where the event point is an intersection point, and modifies the cases where the event point is an end of a polygon edge: once we find an intersection point during examining these cases, we immediately report the intersection and stop the algorithm, instead of inserting the intersection point into the set EV. Moreover, we also simplify the handling of the event points: since now only edge ends are event points, we do not need a complicated dynamic data structure to store them. We can simply pre-sort them and just process them in the sorted order.

```
Algorithm  Polygon-Intersection
Given:  two polygons P1 and P2
Output: report if P1 and P2 intersect
/* We use a vertical line L to sweep the plane. At any moment, the segments
   intersecting L are stored in ST, sorted by the y-coordinates of their
   intersection points with the line L. The event points stored in EV are
   sorted by their x-coordinates. */

1.  sort the edge ends in P1 and P2 by x-coordinates: q_1, q_2, ..., q_m;
2.  ST = {};
3.  For (h = 1; h <= m; h++)
       If (q_h is a right-end of an edge s)
          let si and sj be the two edges adjacent to s in ST;
          If (s (P1,P2)-intersects si or sj at a point p) Report(p); Stop;
          Delete(ST, s);
          If ((si and sj (P1,P2)-intersect at a point p') Report(p'); Stop;
       Else /* q_h is a left-end of an edge s */
          Insert(ST, s);
          let si and sj be the adjacent edges of s in ST;
          If (s (P1, P2)-intersects si or sj at p) Report(p); Stop.
4.  Report('no intersection').
```

Suppose that there are totally n edges in the two polygons P_1 and P_2. In step 3, the algorithm `Polygon-Intersection` examines every event point, which is an end of an edge in the polygons P_1 and P_2. For each event point, the algorithm takes time $O(\log n)$ to do a constant number of Insert, Delete, and Search (search neighbors of an edge in the set ST) operations on the set ST, plus some other operations that take time $O(1)$. Since there are $2n$ event points, we conclude that step 3 of the algorithm takes time $O(n \log n)$. This, plus step 1 that sorts the edge ends in $O(n \log n)$ time, gives the time complexity $O(n \log n)$ for the algorithm `Polygon-Intersection`.

3. (Convex Hull I) Given n points in the plane such that the x-coordinates of these points are the integers $1, 2, \ldots, n$ (However, the points are not necessarily sorted by their x-coordinates). Show that the convex hull of this set can be found in linear time.

Solution. Consider the algorithm `Modified-Graham-Scan` that constructs the convex hull for a given set S of points in the plane:

```
Algorithm Modified-Graham-Scan
Input: a set S of n points in the plane
Output: the convex hull CH(S) of S;

1. L = the points in S sorted in decreasing x-coordinates: p_1, p_2, ..., p_n;
2. assume p_1 = (x_1, y_1);  let p_0 = (x_1, y_1 - 1);
3. Push(K,p_0); Push(K,p_1); /* K is a stack */
4. For (i = 2; i < n; i++)
   /* K[1] and K[2] are the 1st and 2nd vertices on the top of K. */
     While (K[2]K[1]p_i is a right-turn) Pop(K);
     Push(K,p_i).
   /* the resulting list minus the point p_0 is the upper hull. */
5. construct the lower hull using the reversed list p_n, ..., p_2, p_1;
6. concatenate the upper and lower hulls to form the convex hull CH(S).
```

Note that besides step 1, steps 2-6 of the `Modified-Graham-Scan` algorithm takes time $O(n)$ since each point in S is inserted into and deleted from the stack K at most once. Thus, on a set S of points that are sorted by x-coordinates, we can construct the convex hull for S in time $O(n)$. As a result, to have a linear-time algorithm for constructing the convex hull, we only need to find a linear-time algorithm that sorts the input points by their x-coordinates.

As given in the question, the given set S of points have their x-coordinates $1, 2, \ldots, n$, respectively. This set of points can be easily sorted by their x-coordinates in linear time: simply place a point (x, y) in the set S in the x-th position in an array, as given as follows:

```
Algorithm Linear-Time Sorting
Input: a set S = (p_1, ..., p_n) of n points in the plane whose x-coordinates
       are 1, 2, ..., n
Output: array A[1..n], which is the set S sorted by x-coordinates;

1. For (h = 1; h <= n; h++)
     let p_h = (x_h, y_h);  A[x_h] = p_h;
```

Thus, if we replace step 1 in the `Modified-Graham-Scan` algorithm with the algorithm `Linear-Time Sorting`, we get a linear-time algorithm that constructs the convex hull of the set S of points that satisfies the assumed condition.

4. (Convex Hull II) Suppose we know that the convex hull of a set of points is a polygon of no more than $\log \log n$ vertices. Design an efficient algorithm to construct the convex hull.

Solution. Note that `Jarvis March` constructs the convex hull for a set S of n points in time $O(kn)$, where k is the number of vertices on the convex hull, while `Graham Scan` constructs the convex hull for S in time $O(n \log n)$ that is independent of the number k of vertices on the convex hull (the sorting step in `Graham Scan` sorts all n given points in the input, regardless of the number of vertices on their convex hull). Thus, if the convex hull has fewer vertices (i.e., $k < \log n$), then `Jarvis March` is faster, while when the convex hull has more vertices (i.e, $k > \log n$), then `Graham Scan` is faster.

The set of points given in the question is known to have a convex hull that has no more than $\log\log n$ vertices, i.e., $k \leq \log\log n$. Since $\log\log n$ is much smaller than $\log n$, we use `Jarvis March` that constructs the convex hull for the set that satisfies the assumed condition in time $O(n \log\log n)$.

5. (Monotone polygons) Design an algorithm for the following problem: given a set of n points in the plane, construct a simple monotone polygon such that the vertices of the polygon are the points in the given set.

Solution. First note that a *monotone chain* is a chain that is monotone with respect to the y-axis, i.e., each horizontal line intersects the chain at at most one point. A *monotone polygon* is given by two monotone chains.

When a set S of points is sorted by y-coordinates, a monotone chain containing all points in S can be easily constructed by just connecting neighboring points in the sorted list.

To construct a monotone polygon that contains all points in the given set S, we can first find the highest point p_h and the lowest point p_l in the set, then construct two monotone chains between p_h and p_l. Note that we also need to make sure that the two constructed monotone chains do not intersect.

Based on this observation, we have the following algorithm:

```
Algorithm Monotone Polygon
Input: a set S of n points in the plane
Output: a monotone polygon that contains all points in S;

1. let p_h and p_l be the highest and lowest points in S, respectively;
2. S_L = the set of points in S that are on the left side of the segment [p_h,p_l];
   S_R = the set of points in S that are on the right side of the segment [p_h,p_l];
3. sort S_L in decreasing y-coordinates;
   sort S_R in increasing y-coordinates;
4. the concatenation p_h-S_L-p_l-S_R gives a monotone polygon with two monotone
   chains p_h-S_L-p_l and p_l-S_R-p_h.
```

Step 1 of the algorithm takes time $O(n)$ that finds the points with the largest and smallest y-coordinates, respectively, in the set S. Step 2 takes another time $O(n)$ that goes through the points in S to divide S into S_L and S_R. Step 3 sorts in time $O(n \log n)$ the two sets S_L and S_R by their y-coordinates (note that (S_L, S_R) can be a very unbalanced partition of S, but each has at most $n-2$ points). Step 4 takes time $O(n)$ to concatenate the point p_h, the chain S_L, and point p_l, and the chain S_R. Note that the two chains S_L and S_R cannot intersect because they are on two different sides of the segment $\overline{p_h p_l}$. In conclusion, this is an $O(n \log n)$-time algorithm that constructs the monotone polygon.