Student ID: 1155189917. (Jan 19, 2024)

Problem 1. Let P be a set of n points in $\mathbb{R}^2$. Let (x_p, y_p) denote the x-coordinate and y-coordinate, respectively, for p. Define a *preference function* to be a function $f : \mathbb{R}^2 \to \mathbb{R}$ of the form: $f(p) = c_1 \cdot x_p + c_2 \cdot y_p$, where c_1 and c_2 are real-valued constants. Given a preference function f, a *top-1 query* returns a point $p \in P$ that maximizes $f(p)$ among all points in P. Design a structure of $O(n)$ space that answers a query in $O(\log n)$ time. Describe how to construct the structure in $O(n \log n)$ time.

Solution. For any point set S in the convex region which maximizes the preference function, it must be the case that at least one point $p \in P$ satisfies $p \in S$. That is to say, by convexity, the point $p \in P$ which maximizes the preference function must lie on the convex hull of P, $\mathcal{H}(P)$.

Three cases can be identified to prove this fact:

1. p is in the interior of $\mathcal{H}(P)$: shift p with a infinitestimally small distance along the direction $\hat{f}$, reaching p', and we have made $f(p') > f(p)$, thus a contradiction;
2. p lies on the edge of $\mathcal{H}(P)$: if $\hat{f}$ is not parallel to the edge, we can shift p as described above, otherwise shifting p along the edge till the endpoint does not affect the value of the preference function, and then would eventually lead to $p' \in P$;
3. $p \in P$.

We could build the query structure by computing the convex hull of P. Assume that we have the hull vertices in counter-clockwise order (takes up $O(n)$ space).

Imagine scanning a line normal to $c_1 \cdot x_p + c_2 \cdot y_p = k$ (k can take any real values), when we increment k, it would eventually reach the border of $\mathcal{H}(P)$, maximizing the preference function f.

This ensures the unimodality property for applying ternary search on the structure, which is done in $O(\log n)$ time.

```
1: procedure QUERY(vertices on the convex hull of P, H[1], H[2], ..., H[n])
2:     l ← 1
3:     r ← n
4:     while l < r do
5:         mid1 ← l + (r − l)/3
6:         mid2 ← r − (r − l)/3
7:         if f(H[mid1]) > f(H[mid2]) then
8:             r ← mid2
9:         else
10:            l ← mid1
11:        end if
12:    end while
13: end procedure
```

Problem 2. Let P_1, P_2 be two sets of points. Given the convex hulls of P_1 and P_2, describe an algorithm to compute the convex hull of $P_1 \cup P_2$ in $O(n)$ time, where $n = |P_1| + |P_2|$.

Solution. Rotating Calipers can be used to contruct the convex hull.

Theorem. (Bridge) Two vertices $p_i \in P$ and $q_j \in Q$ are *bridge* points if, and only if, they form a copodal pair and the vertices $p_{i-1}, p_{i+1}, q_{j-1}, q_{j+1}$ all lie on the same side of the line $p_i q_j$.

1. Find the vertex with the highest y-coordinate for each of the two polygons.
2. Place a horizontal line (on both polygons, pointing to the positive x-axis) passing through the points found, this line should intersect only with the point for the associated polygon.
3. Rotate these two lines by the smallest angle between the caliper and the segment following the vertex if it passes through (in clockwise order). The rotation is done about the vertex through which the line passes on the associated polygon. If the line passes through more than one vertex of the assciated polygon, the farthest (in clockwise order) is taken.
4. A bridge has been found if the neighboring vertices located by the corresponding line at P and the neighboring vertices located by the corresponding line at Q lies on the same side.
5. Repeat Step 3 and 4 until the two lines have all wrapped around their associated polygons, all bridges should have been reported.
6. Merge these bridges with the edge set of P and Q.

The running time of this algorithm is constrained by Step 1, 5, 6, with time complexity of $O(n)$ in total.

Problem 3. Prove: every polygon (which may be concave) with $n \geq 4$ vertices has at least one diagonal.

Proof. Let v be the leftmost vertex of the polygon, and u, w are vertices adjacent to v.

Then, if $\overline{uw}$ does not cross the edge of the polygon, this is a diagonal for the polygon.

Otherwise, there must be at least one vertex p lying inside $\triangle uvw$. Let v' be the farthest point from line $\overline{uw}$. The segment $\overline{vv'}$ cannot intersect the edge of the polygon since such an edge would have an endpoint inside the triangle that is farther from the line through u and w, contradicting the definition of v'. Hence, $\overline{vv'}$ is a diagonal. □

Problem 4. Consider the following algorithm for triangulating a polygon G:

1: add diagonals to break G into non-overlapping polygons $G_1, G_2, \ldots, G_t$ without split vertices
2: **for** $i \leftarrow 1$ `to` t **do**
3: add diagonals to break G_i into non-overlapping polygons without merge vertices
4: **end for**
5: **for** every polygon G' obtained at Line 3 **do**
6: triangulate G' using an x-monotone algorithm
7: **end for**

Prove: the above algorithm runs in $O(n \log n)$ time where n is the number of vertices in G.

Proof. **Key Observation:** every diagonal is at most shared by two triangles.

To add diagonals for split vertices, there are few steps to notice:

1. Event points are stored in an event queue Q, the next event to be handled is reported in $O(\log n)$ time;
2. A sweepline status implemented by BBST supports modification and queries on segments interseting with the sweepline ℓ in $O(\log n)$ time;
3. Whenever we encounter a split vertex, resolve it by adding diagonals. We can maintain a *helper* for each edge e in constant time (defined as the rightmost visible vertex between e and the edge immediately below e in the sweepline status), so that the split vertex can be connected to the helper of the edge immediately above this vertex;

For the t polygons generated, by symmetry we can easily handle the case for merge vertices.

The running time will be

$$O\left(\sum_{k=1}^{t} v_k \log v_k\right) = O(n \log n),$$

by noting that $t \leq n - 2$ and thus

$$\sum_{k=1}^{t} v_k \leq n + 2t \leq 3n,$$

where v_k is the number of vertices of the k-th polygon generated.

Finally, the monotonic partitioning algorithm will generate polygons with in total at most $3n$ vertices, and thus triangulating these x-monotone polygons will take at most $O(n)$ time in total. □

Problem 5. Let G be a convex polygon of n vertices, which are given to you in clockwise order. Given an arbitrary point $q \in \mathbb{R}^2$, describe an algorithm to decide whether q is inside or outside G in $O(\log n)$ time.

Solution. Note that all vertices are given in counterclockwise order.

We can first find the point p_0 with the smallest x-coordinate, assuming general position that no two points share the same x-coordinate. By Problem 1, this has an $O(\log n)$ solution. Label the other vertices in counterclockwise order with $p_1, p_2, \ldots, p_n$.

Now verify that q lies within the convex cone bounded by $\overline{p_0p_1}$ and $\overline{p_0p_n}$, which can be done in $O(1)$ time by investigating the cross product.

By calculating $\overrightarrow{p_0p_1} \times \overrightarrow{p_0q}$, $\overrightarrow{p_0p_1} \times \overrightarrow{p_0p_n}$, $\overrightarrow{p_0p_n} \times \overrightarrow{p_0q}$ and $\overrightarrow{p_0p_n} \times \overrightarrow{p_0p_1}$, we can claim the followings:

- The given point q lies on the line $\overline{p_0p_1}$, if $\overrightarrow{p_0p_1} \times \overrightarrow{p_0p} = 0$;
- The given point q lies on the line $\overline{p_0p_n}$, if $\overrightarrow{p_0p_n} \times \overrightarrow{p_0p} = 0$;
- The given point q lies out of the cone formed by $\overline{p_0p_1}$ and $\overline{p_0p_n}$ if either $\overrightarrow{p_0p_1} \times \overrightarrow{p_0q}$ and $\overrightarrow{p_0p_1} \times \overrightarrow{p_0p_n}$ or $\overrightarrow{p_0p_n} \times \overrightarrow{p_0q}$ and $\overrightarrow{p_0p_n} \times \overrightarrow{p_0p_1}$ has different signs.

We then check if q really locates within the segment $\overline{p_0p_1}$.

After dealing with all the special cases, we can consider spliting the convex polygon in triangular regions, and locate the point q by binary search.

```
1:  procedure BinarySearch
2:      l ← 1
3:      r ← n
4:      while l < r do
5:          mid ← (l + r)/2
6:          if (p0pmid × p0q ≥ 0) then
7:              r ← mid
8:          else
9:              l ← mid + 1
10:         end if
11:     end while
12: end procedure
```

At last, verify that q lies within $\triangle p_0p_lp_{l-1}$ by checking three cross products, in $O(1)$ time.

Problem 6. Given a polygon G of n vertices, decide in $O(n)$ time whether G can be made x-monotone by rotating the coordinate system at the origin.

Solution. This is equivalent to determining whether the polygon G is monotone with respect to some line L.

Note that the definition of monotone polygons with respect to a line ℓ ensures that every line ℓ' orthogonal to ℓ should intersect with the polygon in one single continuous interval.

Consider the concave vertices which obstructs the lines from intersecting the polygon no more than twice, these lines form a set of forbidden angles from which the orthogonal line cannot be placed. Note that we can easily calculate the angles of these segments on the polygon, in $O(1)$ time. Thus, determining the forbidden angles for the concave vertices is also easily computed in $O(1)$.

Then, we calculate the union of these forbidden angles, in $O(n)$ time.

Finally, if the union covers the all possible angles, we determine that it is impossible to make G x-monotone by rotation.

Otherwise, of any possible angle α not forbidden, the polygon is monotone with respect to the line with slope $-1/\tan\alpha$.

Problem 7. Let G be a polygon with n vertices. Two points p and q in the polygon are *visible* to each other if the the segment $\overline{pq}$ is fully contained by the polygon. Given a set S of vertices of G, we say that S guards G if every point inside G is visible to at least one vertex in S. Give an $O(n \log n)$ time algorithm to find a set S of size at most $n/3$ to guard G.

Solution. A simple polygon can be triangulated in $O(n \log n)$ time.

The graph generated by the triangulation can be 3-colored. Notice that the dual of the triangulated polygon graph forms a tree. Every diagonal added splits the polygon into two parts, leading to the fact that any removal of edges from the dual graph splits the graph into two components. The degree of the triangles (vertices in the dual) is contrained to have a degree of at most 3, and thus in the original triangulated polygon, we can alternate the colors to avoid clashing.

Then, the triangulation admits a 3-coloring by constructing a mapping from the vertex set V of the polygon to the 3 colors, such that two vertices p and q have the same color if and only if $\overline{pq}$ is not an edge of the polygon and $\overline{pq}$ is not a diagonal. One single depth-first search traversal gives the complexity of $O(n)$ of coloring the vertices. Note that once the first triangle has manually been colored, the other triangles have their vertices colored automatically as well.

Now we know that every vertex $v \in G$ are visible to all points lying on or inside triangles with this vertex. Otherwise, this violates the convexity of a triangle. With the coloring specificly picks colors for each vertex evenly, we can guard G using at most $n/3$ vertices.

The overall running time is $O(n \log n)$ as required.