Tutorial 2

Problem Statement

$P[1..m]$ is an input list of n points on xy-plane. Assume that all n points have distinct x-coordinates and distinct y-coordinates. Let p_{L} and p_{R} denote the leftmost and points of P, respectively. The task is to find the polygon Q with P as its vertex set such that the following conditions are satisfied.

1. The upper vertex chain of Q is x-monotone (increasing) from p_{L} to p_{R}.
2. The lower vertex chain of Q is x-monotone (decreasing) from p_{R} to p_{L}.
3. Perimeter of Q is minimum.

Algorithm

Say the n points are $x_1, x_2, ..., x_{\mathrm{n}}$. Let's assume them to be ordered by their x-coordinates i.e. x_1 is the leftmost and x_{n} is the rightmost.

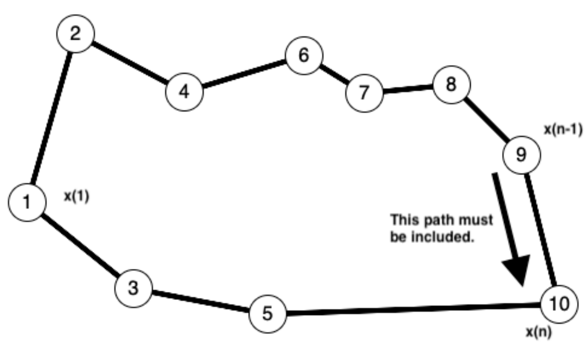


Figure 1: Path $(x_{\text{n-1}}, x_{\mathrm{n}})$

Lemma 1. *The segment $(x_{n\text{-}1}, x_n)$ will be contained in the polygon Q.*

Proof. Suppose the segment $(x_{\text{n-1}}, x_{\mathrm{n}})$ is not contained in Q. This means that the polygon must have a portion like this - $x_{\mathrm{i}}..x_{\text{n-1}}..x_{\mathrm{j}}..x_{\mathrm{n}}$. However we know that both x_{i} and x_{j} have smaller coordinates than $x_{\text{n-1}}$ which means that the portion is not x-monotone, which is a contradiction. $\square$

Now let's frame this problem as finding a path from x_{n} back to itself such that the initially we strictly travel left to x_1 and then we strictly travel right to x_{n}. One possible question that might arise is - **Why should we expect the path to be non-crisscrossing?**. We will answer this later. For now assume that the result is a normal polygon.

From lemma 1, it suffices to find the length of the minimal path going from x_{n} strictly to the left upto x_1 - leaving out $x_{\text{n-1}}$ - and then from x_1 strictly to the right upto $x_{\text{n-1}}$ and add it to the distance between x_{n} and $x_{\text{n-1}}$.

We now make the following observations :

1. Any acceptable path from x_{n} to $x_{\text{n-1}}$ must start with a first edge $(x_{\text{n}}, x_{\text{k}})$ for some $k < n-1$.
2. Since we must visit all points, and since from x_{k} we can only continue to the left, all the points $x_{\text{k+1}}, x_{\text{k+2}}, ..., x_{\text{n-1}}$ must necessarily be visited on the way from left to right (and in this order). So, necessarily our path ends with $x_{\text{k+1}} \longrightarrow x_{\text{k+2}} \longrightarrow ... \longrightarrow x_{\text{n-1}}$.

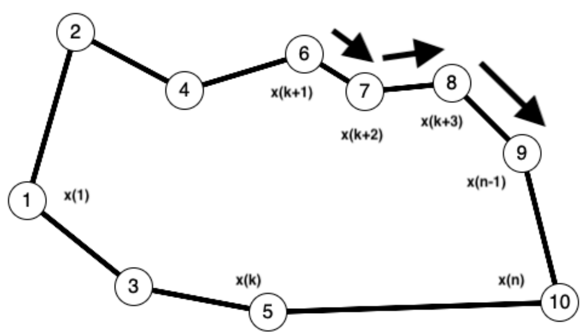


Figure 2: These paths must be visited on the return journey.

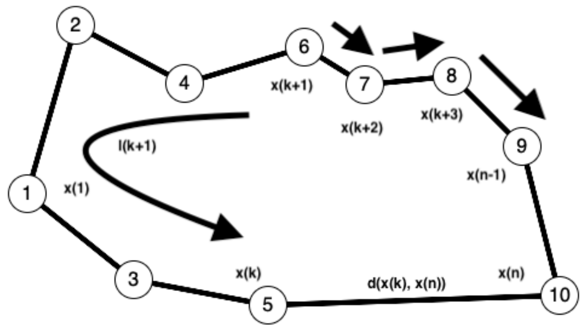
So far we have figured out that an acceptable path from x_{n} to $x_{\text{n-1}}$ has the form

$$x_{\text{n}} \longrightarrow x_{\text{k}} \longrightarrow \;??? \longrightarrow x_{\text{k+1}} \longrightarrow x_{\text{k+2}} \longrightarrow ... \longrightarrow x_{\text{n-1}}.$$

where $\longrightarrow x_{\text{k}} \longrightarrow \;??? \longrightarrow x_{\text{k+1}}$ is in itself an acceptable path (satisfying all the constraints) from x_{k} to $x_{\text{k+1}}$ with $k < n-1$.

If we want to minimize the length of path from x_{n} to $x_{\text{n-1}}$, we must also minimize the length of the path from x_{k} to $x_{\text{k+1}}$ (which is the same as the length of the acceptable path from $x_{\text{k+1}}$ to x_{k}).
For any $i > 1$ let $l(i)$ denote the length of the acceptable minimum length path from x_{i} to $x_{\text{i-1}}$. The preceeding arguments imply that :

$$l(n) = d(x_{\text{n}}, x_{\text{k}}) + l(k+1) + \textstyle\sum_{m=k+1}^{n-2} d(x_{\text{m}}, x_{\text{m+1}})$$



For some $k < n$ we have :

$$l(n) = \min_{1<i<n} \left[\; d(x_{\mathrm{n}}, x_{\mathrm{i\text{-}1}}) + l(i) + \sum_{m=k+1}^{n-2} d(x_{\mathrm{m}}, x_{\mathrm{m+1}}) \; \right]$$

The exact same reasoning also applies on such paths for any $2 < p < n$ i.e we have obtained the recursion :

$$l(p) = \min_{1<i<p} \left[\; d(x_{\mathrm{p}}, x_{\mathrm{i\text{-}1}}) + l(i) + \sum_{m=k+1}^{p-2} d(x_{\mathrm{m}}, x_{\mathrm{m+1}}) \; \right]$$

And $l(2) = d(x_2, x_1)$. We can use this recursion to successively calculate $l(p)$ for $p = 2, ..., n$, then the required lenght for all sets of points would be :

$$l(n) + d(x_{\mathrm{n}}, x_{\mathrm{n\text{-}1}})$$

How to construct the path?

To construct the optimal path satisfying the constraint we only need to store the value of i that optimizes $l(p)$ for all values of p. From this we can find the neighbour of x_{n} and then the neighbour of that neighbour and so on.

Why would the resulting path be a polygon?

Now coming back to the assumption. It is actually very easy to see that for any path that consists of criss crossing edges we can construct a shorter path without having crossing edges. This is a direct consequence of the triangle inequality.

Consider an example where the paths AB and CD in the above polygon are replaced by AD and BC. Also, note that Q is the intersection AD and BC in the new polygon.

By triangle inequlity $BQ + AQ$ in the second polygon must be greater than AB in the first polygon.
Also, $CQ + DQ$ in the second polygon must be grater than CD in the first polygon.

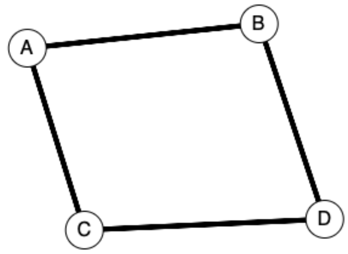


Figure 3: Polygon 1

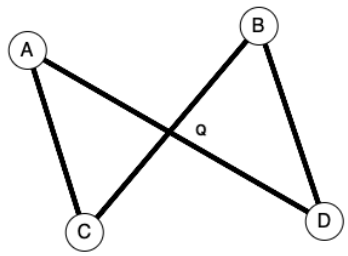


Figure 4: Polygon 2

The first polygon must have smaller total path length than the second polygon.

Therefore, we can say that the result of the algorithm will always be a polygon.

Time and Space Complexities

1. The time complexity is $\mathcal{O}(n^2)$.

   Justification - It can be seen that for each value of $p > 2$ we have to take the minimum of $p - 2$ terms. Also, note that the value of $\sum_{m=k+1}^{p-2} d(x_\mathrm{m}, x_\mathrm{m+1})$ can be calculated in $\mathcal{O}(1)$ time after a pre-processing that takes $\mathcal{O}(n^2)$ time. Therefore the required time complexity calculate will take the form $\sum_i (i-2)$ which will lead to a an overall time complexity of $\mathcal{O}(n^2)$.

2. The space complexity is also $\mathcal{O}(n)$.

   Justification - As explained in the previous section, we need to store the value of i that optimizes $l(p)$ for all values of p, where $2 < p < n$. Therefore, we require $\mathcal{O}(n)$ space.