Quiz 2 25/02/21

Juan E. Murcia N.

1)

- HandleStartVertex(v_i):

  Insert e_i in $\mathcal{T}$ and set helper(e_i) to v_i

Assuming $\mathcal{T}$ is a balanced binary search tree, inserting e_i in $\mathcal{T}$ should take $O(\log n)$ where n is the number of vertices, we could think that start vertices are only found at the beginning, so it should be $O(1)$, but there are cases of polygons with a mayority of start vertices.

- HandleEndVertex(v_i):

  If helper(e_{i-1}) is a merge vertex:

    Insert the diagonal connecting v_i to helper(e_{i-1}) in $\mathcal{D}$

  Delete e_{i-1} from $\mathcal{T}$

Inserting a diagonal in $\mathcal{D}$ takes constant time with respect to the total of vertices, therefore this function takes $O(\log n)$, due to the delition in $\mathcal{T}$

- HandleSplitVertex(v_i):

  Search in $\mathcal{T}$ to find the edge e_j directly left of v_i

  Insert the diagonal connecting v_i to helper(e_j) in $\mathcal{D}$

  helper(e_i) = v_i

  Insert e_i in $\mathcal{T}$ and set helper(e_i) = v_i

Searching and inserting both take $O(\log n)$, meanwhile inserting in $\mathcal{D}$ and assigning take $O(1)$ so this function into $\mathcal{T}$ takes $O(\log n)$

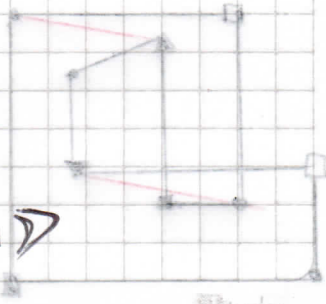
- HandleMergeVertex(v_i):

  if helper(e_{i-1}) is a merge vertex:

    Insert the diagonal connecting v_i to helper(e_{i-1}) in $\mathcal{D}$

  Delete e_{i-1} from $\mathcal{T}$

  Search in $\mathcal{T}$ to find the edge e_j directly left of v_i



If helper(e_j) is a merge vertex:

Insert the diagonal connecting v_i to helper(e_j) in $\mathcal{D}$

helper(e_i) = v_i

Once again this function takes $O(\log n)$

Handle Regular Vertex(v_i):

If interior of $\mathcal{P}$ lies on the right of v_i:

If helper(e_{i-1}) is a merge vertex:

Insert the diagonal connecting v_i to helper(e_{i-1}) in $\mathcal{D}$

Delete e_{i-1} from $\mathcal{T}$

Insert e_i in $\mathcal{T}$ and set helper(e_i) to v_i

Search in $\mathcal{T}$ to find the edge e_j directly left of v_i

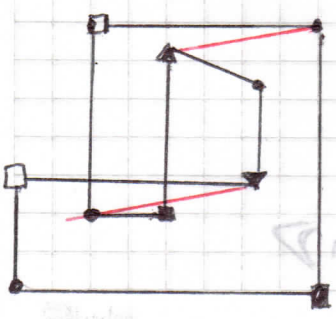
—If helper(e_j) is a merge vertex:

Insert the diagonal connecting v_i to helper(e_j) in $\mathcal{D}$

helper(e_j) = v_i

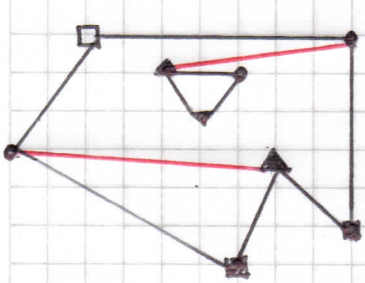
This function also takes $O(\log n)$. Now, as we are expecting a simple polygon $\mathcal{P}$, the complex. of the subdivision and n are around the same, and at most we're inserting a new edge per vertex, so at most there are around 2 edges per vertex once is divided into monotone poligons, then it is also logaritmic regarding the complex. of the subdivision.

2) If we follow the pseudocode of MakeMonotone(P), then the answer would be no, because the non-simple polygon P will get the following output



In red are the new edges added, we clearly see that the algorithm couldn't output monotone polygons. However, if we modify the Handle Regular Vertex(V) function to check if a vertex is in the interior of the polygon before adding new edges, we could obtain our answer.

we're also expecting the algorithm to fail if the polygon has holes,



This can be solved by spliting the polygon into filled polygons first, or dealing with hole vertices with new events perhaps.

However if we assume a proper monotone division, the triangulation algorithm holds for both degenerations.

3) The most crucial ingredient for the polygon triangulation algorithm is the fact that all polygons can be triangulated, there exists a triangulation. Other theorems like the necessity to get rid of split or merge vertices to obtain y-monotone polygons are not so important, because we could do triangulations with x-monotone polygons. And the time complexity theorems are a consequence of the algorithm, but not a crucial ingredient for it.