Second coursework of Advanced Topics in Algorithms:

Generate a simple polygon gluing triangles

By: Marta Gómez (202003750) and Yosbi Saenz (201801653)

Faculdade de Ciencias
Universidade do Porto
Portugal
June 06, 2021

Problem

Develop and implement an algorithm for generating a simple polygon with n-vertices, based on gluing triangles.

- The coordinates of the vertices of the polygon will be non-negative integers, smaller than a given value M.
- The output is the sequence of vertices of the polygon in CCW order and, if required, a triangulation of the polygon represented by a DCEL. You should write a function that creates the polygon and the DCEL representing the triangulation underlying the generation procedure.

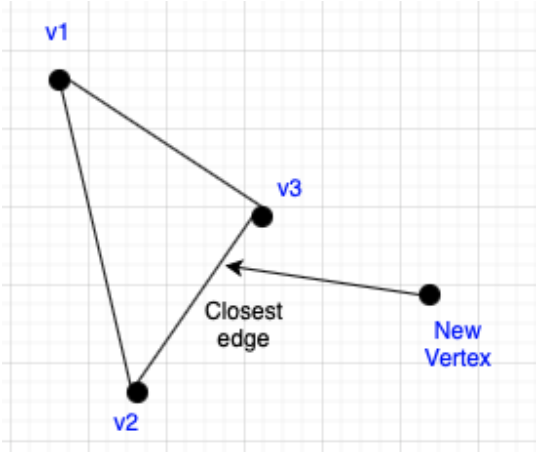
Solution

The proposed solution is based on a DCEL representation. With it we will represent the n-vertex polygon by a set of $HalfEdges$, $Vertices$ and $Faces$. Each HalfEdge stores the information of the origin vertex, the twin, next and previous HalfEdges, and the face in which it is contained, as well as a tag for it.

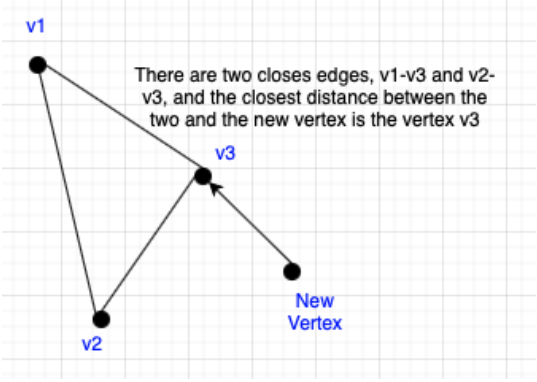
With all this information we generate the algorithm, which is based on finding the outer edge closest to the point to build based on it the new triangle that will "enlarge" the polygon.

A usefull "constant" we defined is that the Face 0 is always the outer face of the polygon.

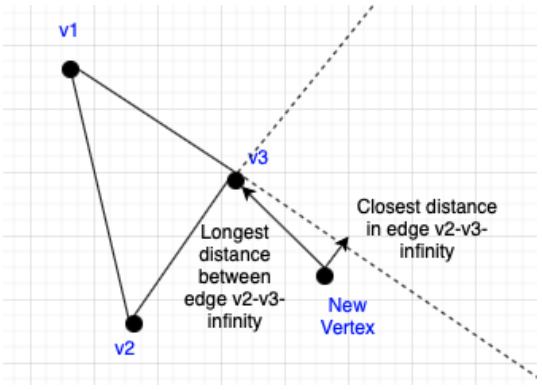
To calculate that nearest edge we have used the function $GetDistanceFromLineSegment$, which has time complexity $O(n-1)$, with n the number of vertices, since it goes through all the outer edges to find the nearest one to the new vertex, and thus build the new triangle.



Sometimes the function $GetDistanceFromLineSegment$ gives a draw between two edges, this case happens when the closest distance to the new vertex from these two edges is the same final vertex of the edge:



To solve this conflict, we use the function *GetDistanceFromLine*, that extend the edges to infinity, and calculate the closest distance between the point and the lines, and, in this case, we use the edge with the longest distance to the new vertex because in this case, the longest distance edge means that it is the most perpendicular edge.



So, in this case we stay with the edge v2-v3 to add the new triangle.

When adding a new vertex then, since we know where to connect it, we have to create and update the values of the HalfEdges, Vertex and Faces associated to this new triangle, with time complexity $O(1)$.

Finally, the output is the set of vertices of the polygon in CCW order, which will take a complexity time of $O(n)$, with n the number of vertices. We generate this output by just iterating between the n vertices that are in contact with the Face 0 (outer face)

Therefore, our solution for each insertion the complexity is $O(n+1)$ being n the number of vertices already inserted and the "+1" the new vertex to be inserted. And finally for a series of insertions of n vertices, we have a complexity equivalent to a summantion: $n(n+1)/2$, giving a total complexity of $O(n^2)$

Program

We have created a program in which to construct, and at the same time visualise, the polygon with this algorithm.

To do so, we used a free and open-source web framework called Blazor that is being developed by Microsoft, in where we can construct web apps. In our case we used the Blazor WebAssembly using the programming language C# to construct the algorithm, and, HTML5, CSS and javascript to build the UI. The special area in where we make the drawing of the polygon is a HTML5 canvas element.

We have generated an interface using the HTML canvas element in which we can vary the size of the plane (M), we can add vertices, either by clicking on the plane or by adding their coordinates. In addition to visualising the polygon, we can visualise the data that the DCEL is going to s tore, as well as the output solution requested, the vertices ordered according to CCW.

It is considered that the inserted vertices will always be inserted correctly, i.e. outside the polygon generated so far, or on the outer edges. In one of the cases, a cut in the polygon is generated by adding a new vertex, but this case is not covered by our algorithm (see example2). In the rest of the cases, by inserting correct vertices, we obtain the expected solution.

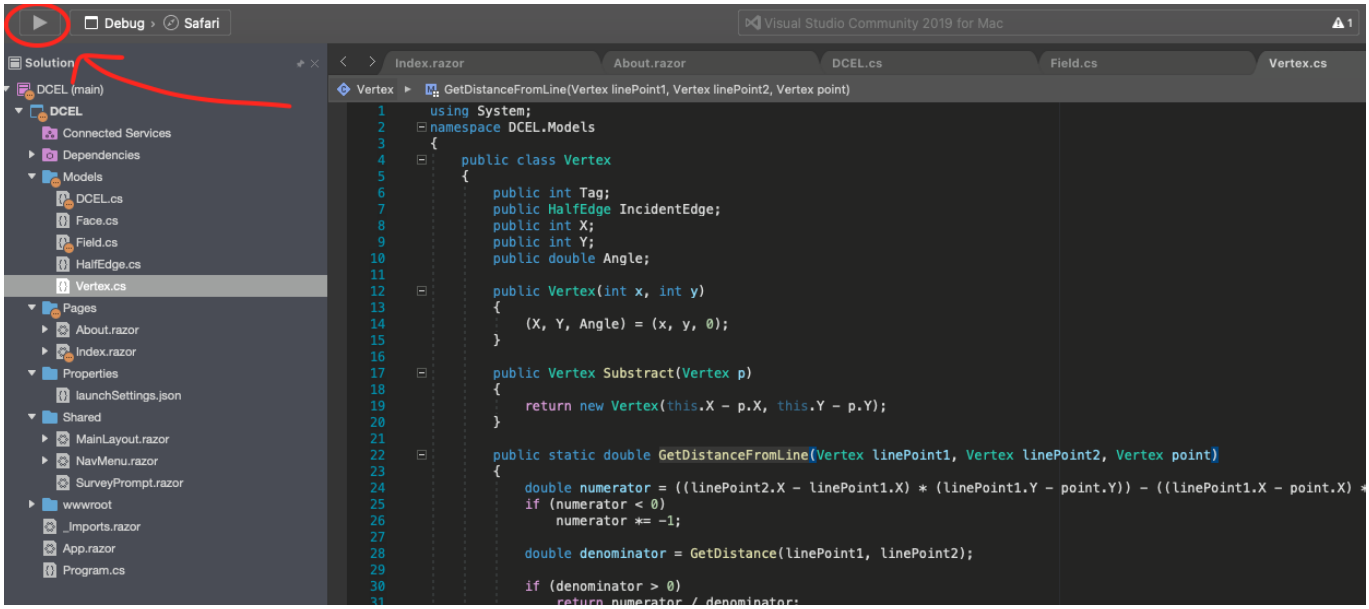
Where is the code?

You can find the code in github in here: `https://github.com/YosbiAlves/TAA`

How to run

In order to run the program you need to have installed Visual Studio 2019. Open the solution (DCEL.sln) with the Visual Studio and push the play button to run the web-app.

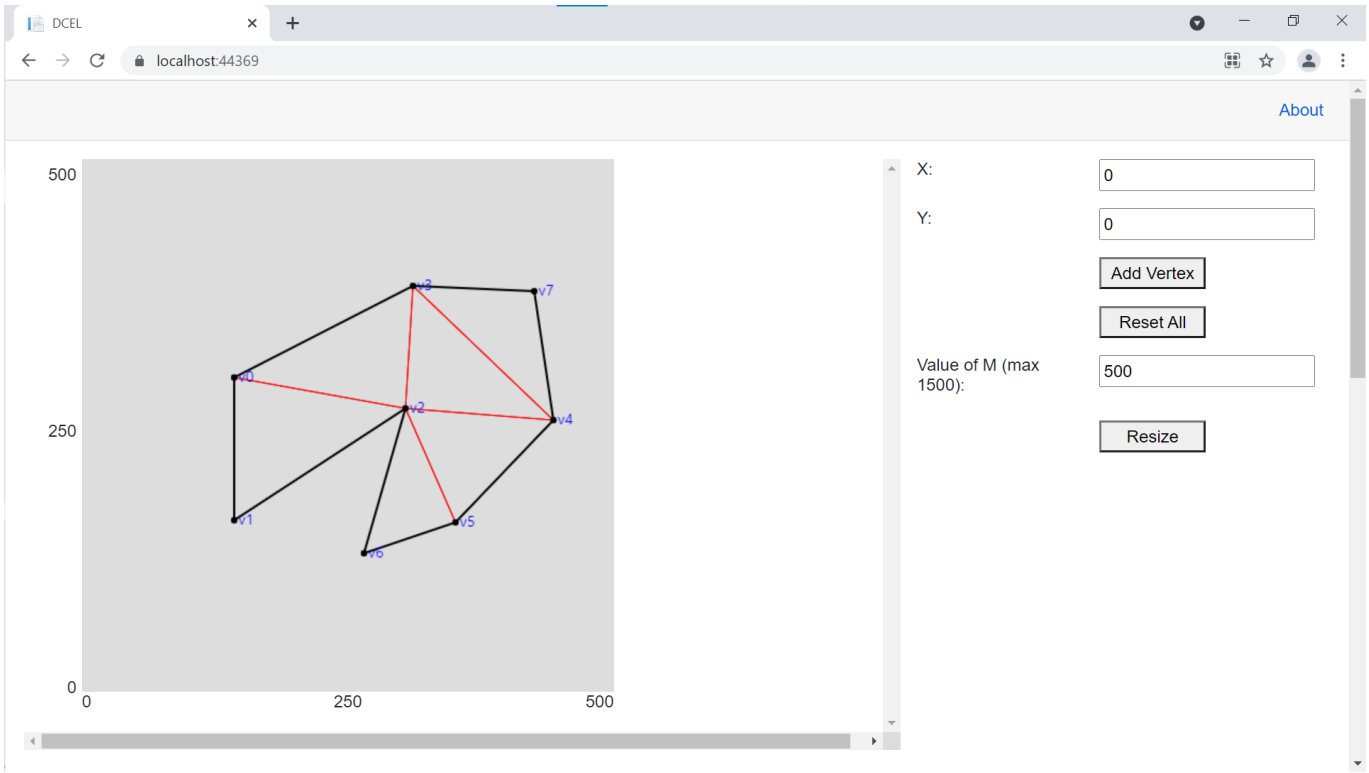
We tested and develop the app using Windows 10 and MacOS BigSur.

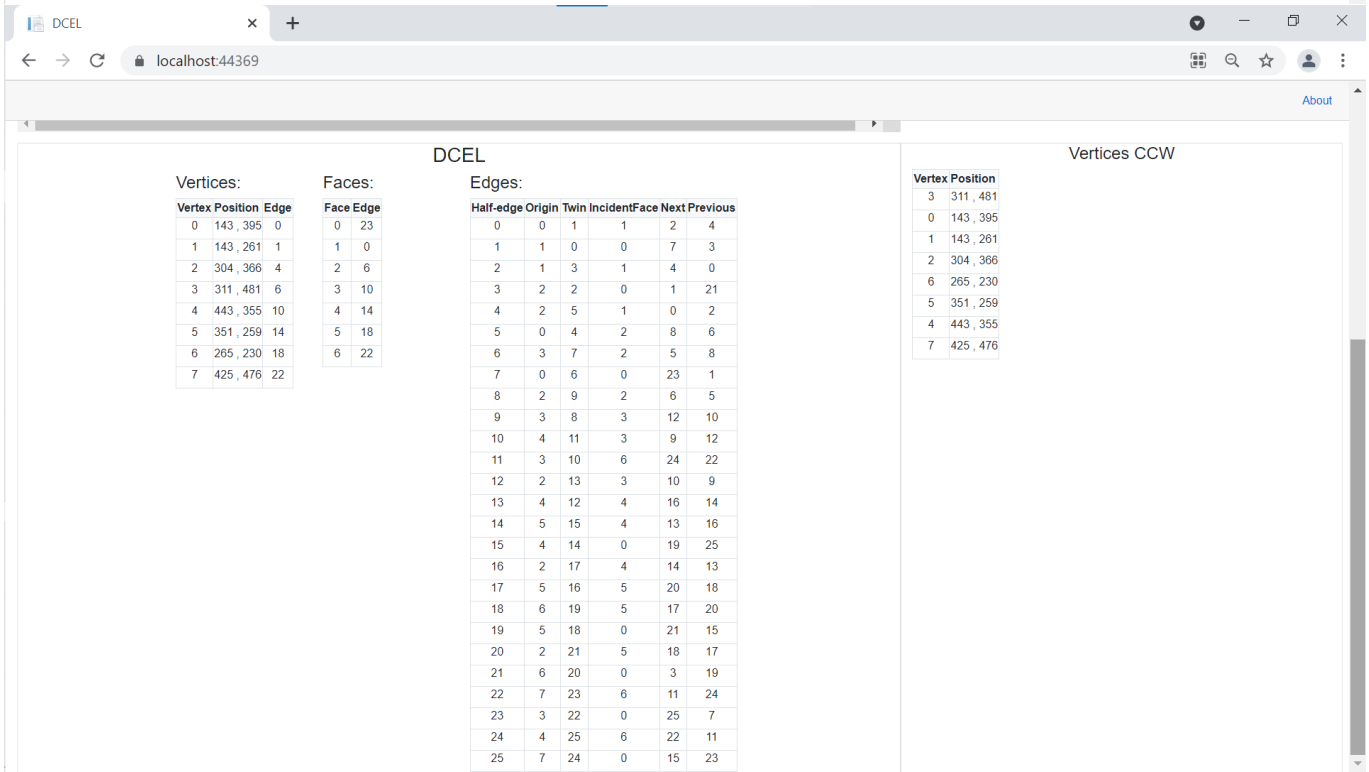


You can run it in linux too but it requires a more elaborated configuration since you have to use Visual Studio Code, you can find how to configure the environment in linux by following the steps on this website:
`https://dev.to/rineshpk/blazor-server-crud-app-using-visual-studio-code-2b2g#:~:text=Blazor%20is%20a%20new%20Microsoft,C%23%2C%20HTML%2C%20and%20CSS.`

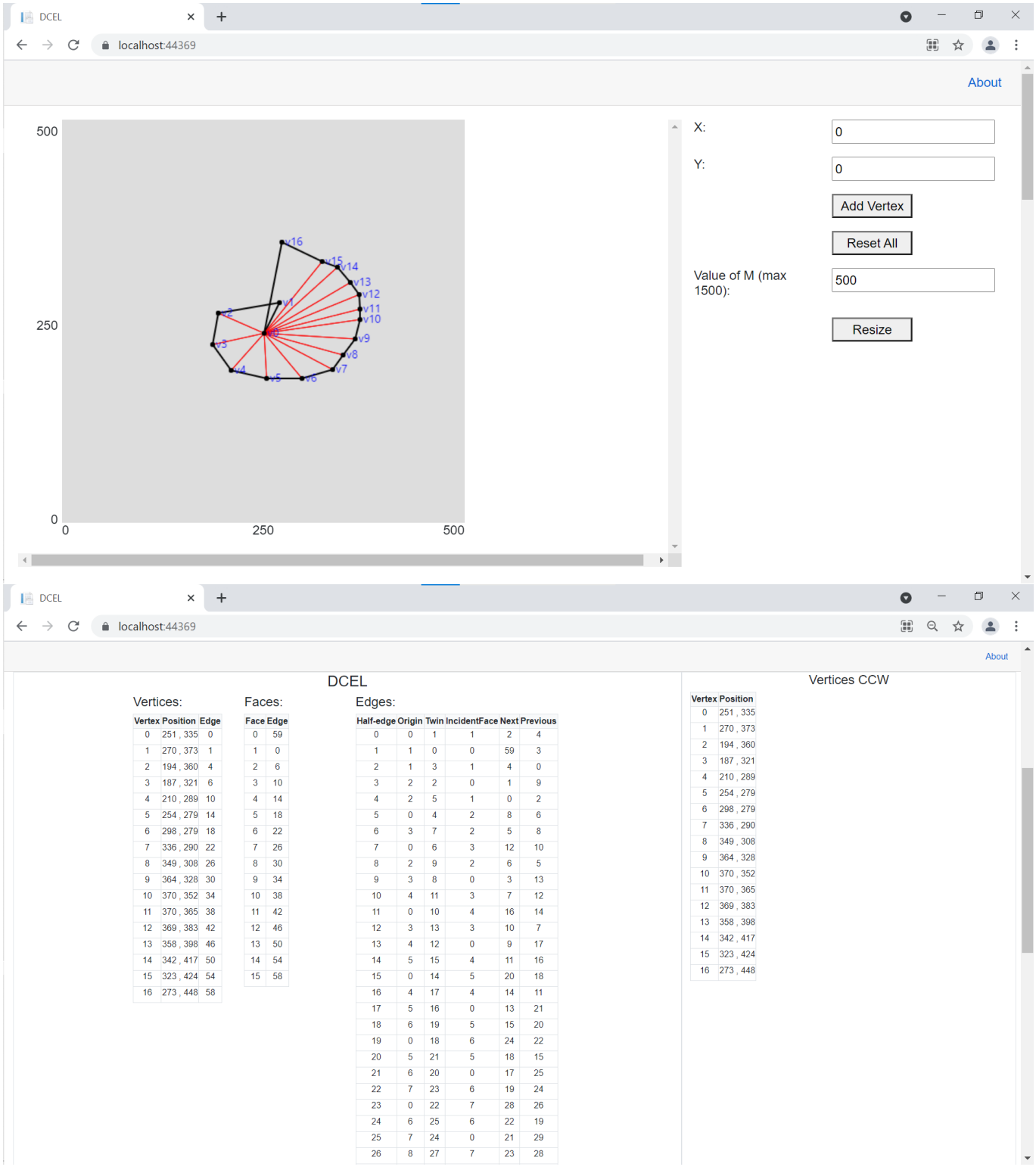
Examples

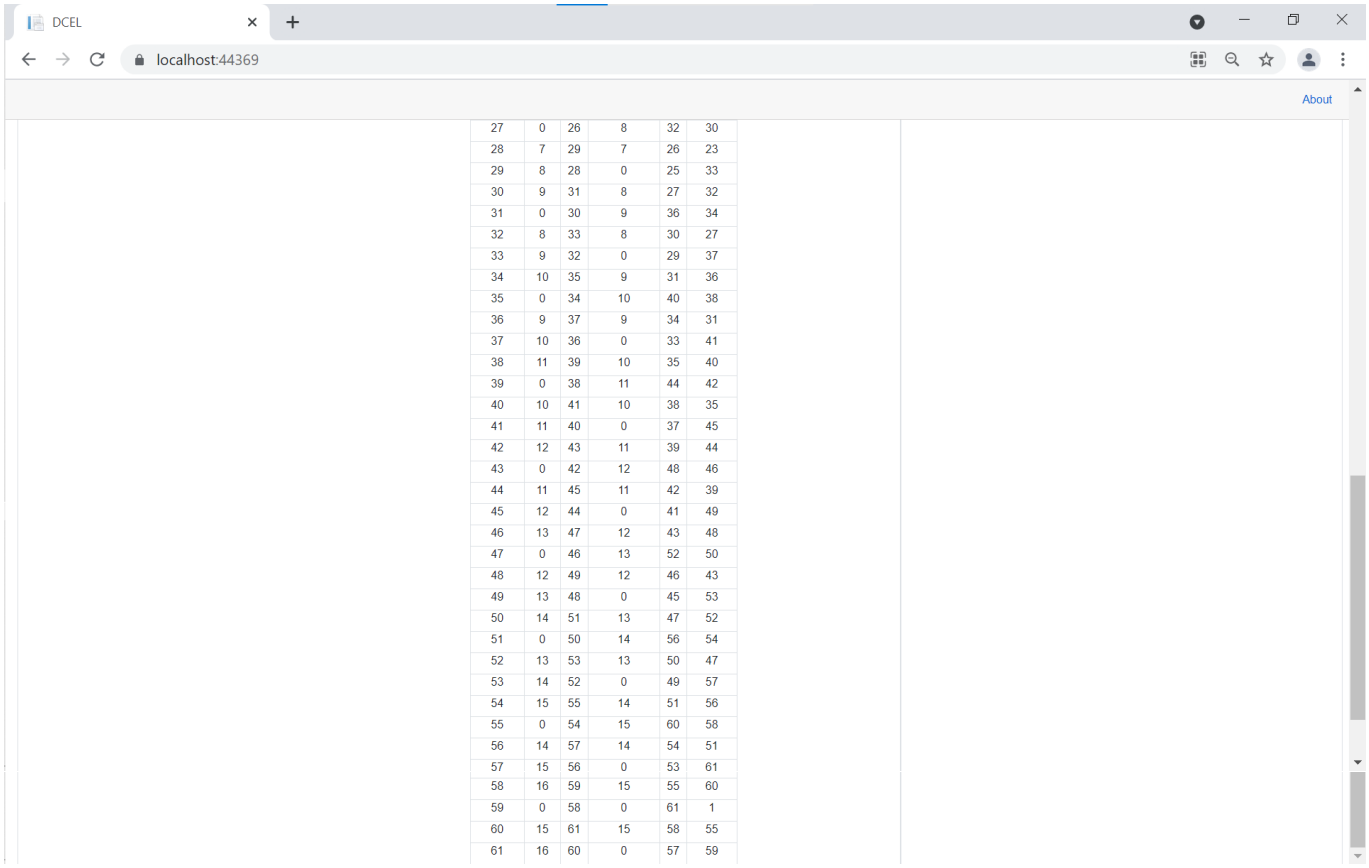
Example 1: Here we have an example of how does the program work.





Example 2: Here we have the exception mentioned above.





27	0	26	8	32	30
28	7	29	7	26	23
29	8	28	0	25	33
30	9	31	8	27	32
31	0	30	9	36	34
32	8	33	8	30	27
33	9	32	0	29	37
34	10	35	9	31	36
35	0	34	10	40	38
36	9	37	9	34	31
37	10	36	0	33	41
38	11	39	10	35	40
39	0	38	11	44	42
40	10	41	10	38	35
41	11	40	0	37	45
42	12	43	11	39	44
43	0	42	12	48	46
44	11	45	11	42	39
45	12	44	0	41	49
46	13	47	12	43	48
47	0	46	13	52	50
48	12	49	12	46	43
49	13	48	0	45	53
50	14	51	13	47	52
51	0	50	14	56	54
52	13	53	13	50	47
53	14	52	0	49	57
54	15	55	14	51	56
55	0	54	15	60	58
56	14	57	14	54	51
57	15	56	0	53	61
58	16	59	15	55	60
59	0	58	0	61	1
60	15	61	15	58	55
61	16	60	0	57	59

References

- Minimum distance from a point to the line segment using Vectors, in https://www.geeksforgeeks.org/minimum-distance-from-a-point-to-the-line-segment-using-vectors/
- Distance from a point to a line, in https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line#Line_defined_by_two_points
- Doubly Connect Edge List (DCEL), in https://www2.cs.sfu.ca/~binay/813.2011/DCEL.pdf
- Plane Graphs and the DCEL, in https://www.ti.inf.ethz.ch/ew/courses/CG12/lecture/Chapter%205.pdf