Water points in the school

2021-2022

Name, surname, level of students : Lohan Larisa and Mateș Andra, 10th grade
Lorena Gabrian, Cristian Gavrila, Teofil Voicu, Cezara Iancu, Ilinca Moisa, Mihai Pruneanu, 11th grade
Tyliann Nozais, Ismaël Bejaoui, Philomène Minima, Mathilde Barouch-Lemarchand, Grégoire Lenay, Flavien Sansonetti, Bilel Lotfi, Chloé Blanc, 11th grade

Schools : Colegiul Național "Mihai Eminescu", Colegiul Național "Emil Racoviță", School "Val-De-Durance"

Teachers : Daly Marciuc, Ariana Văcărețu, Hubert Proal

Researchers : Yves Papegay-Inria, Sophia Antipolis institute, George-Cătălin Țurcaș, Faculty of Mathematics and Computer Science, Babeș-Bolyai University

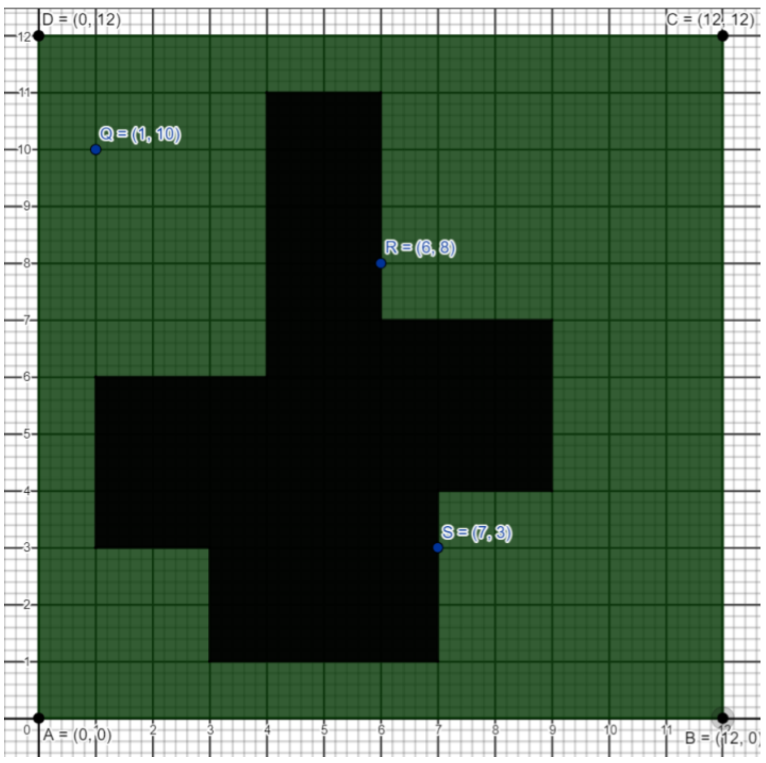
Abstract. This article is based on the MATh.en.JEANS subject no.4 in the 2021-2022 ERASMUS+ MaSuD project. The problem proposes a school's building site and some water points, the task asking for different distances and areas to be found. In the present manuscript we proposed variants of solutions for the main task of the project and describe some generalisations. The MaSuD project, a KA229 Erasmus+ partnership, combines two educational issues: math anxiety and raising awareness on environmental issues. We believe that learning mathematics through open-ended problems allows students to overcome math anxiety and acquire various concepts and thinking skills.

Contents

1 Introduction

The statement of the research topic is as follows: Given a fixed planar model for the shape of the walls of our school, where the fixed points Q, R and S represent water hydrants (fire hydrant or yard hydrant), decide, for every point in the interior of our school, which water hydrant is the closest. [1]

Figure 1 – The plan of the school



2 Approaches and solving

2.1 A mathematical solution

As advertised in the introduction, we are going to work with a planar model of our school. Every point P in the plane can therefore be represented by a pair of real numbers (x_P, y_P).

Formula The distance between two points

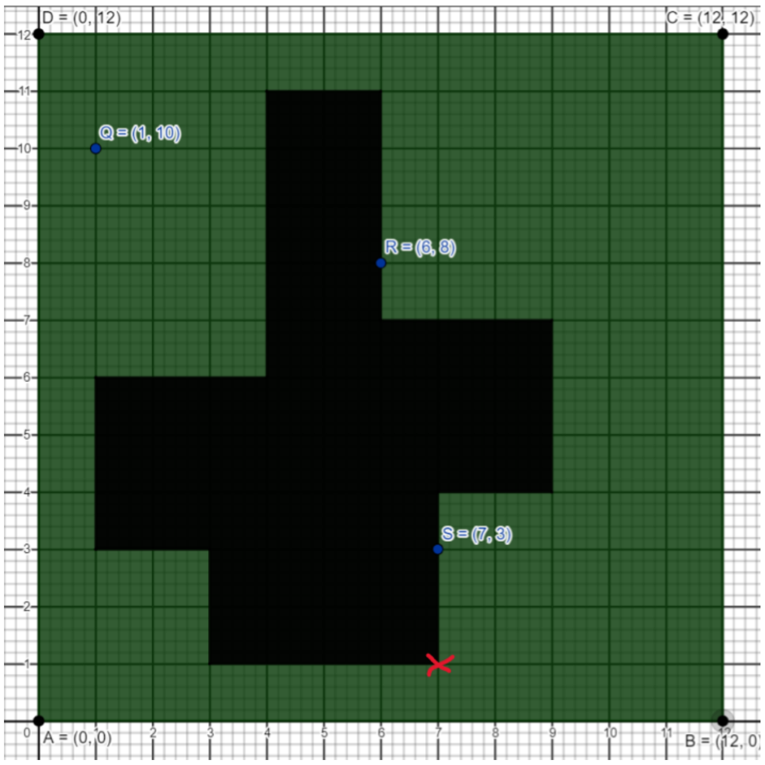
$$A(x_1, y_1), B(x_2, y_2) \tag{1}$$

is given by the formula

$$d = \sqrt{(x_1 - x_2)^2 + \left(y_1 - y_2\right)^2}\,. \tag{2}$$

The first idea was to take some random points and calculate the distance between them and the water points. So, for example we took the point marked with the red cross on *Fig.2* and the distance obtained by applying the formula mentioned before for QX is 11.7, or for QZ is 10.2. But, soon enough we understood that this was not at all an efficient method, an infinity of calculations being needed for a precise result and moreover, there is no deterministic algorithm that can perform this computation.

Figure 2 – The point we chose to use to exemplify the distance formula



2.2 An efficient method using Geogebra

As the problem is one about distances between points, we thought about circles and their radiuses. It is well known that circles can be used to determine equal distances, therefore they can be used to compare distances, as well. For a clean, accurate and rigorous graphic representation, we used the computer program Geogebra Geometry.

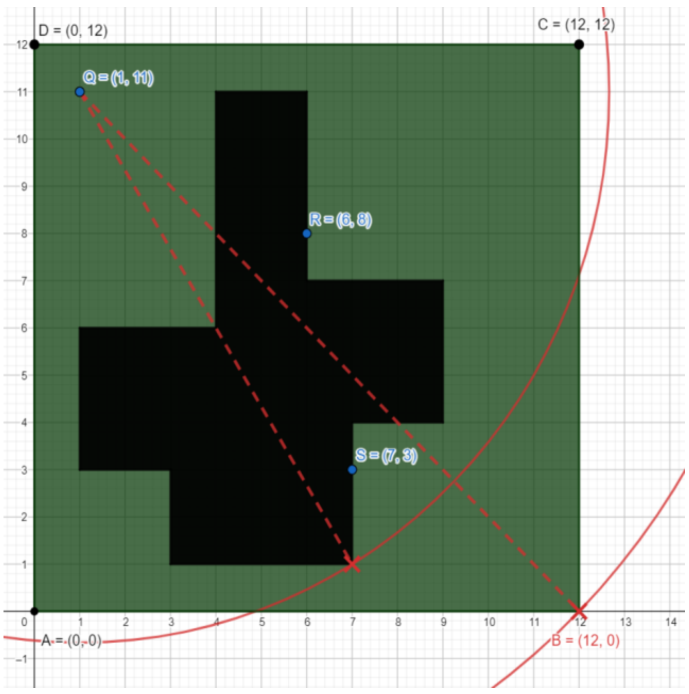
2.2.1 The greatest distance between a water point and a point in the school

In this section, we aim to learn which points inside and outside of the school are situated furthest away from each water point.

Extensive scheme. [2] If we create a circle with the center in the water point, the points furthest away from it, that are included in the surface of the circle, will be located its boundary. Any other point situated inside the created shape will not be as far from the water point as them. So, if the circle is enlarged until it intersects the area of the school in only one point (and the rest of the school is inside the circle), that will be the greatest distance between the used water point and a point in the school.

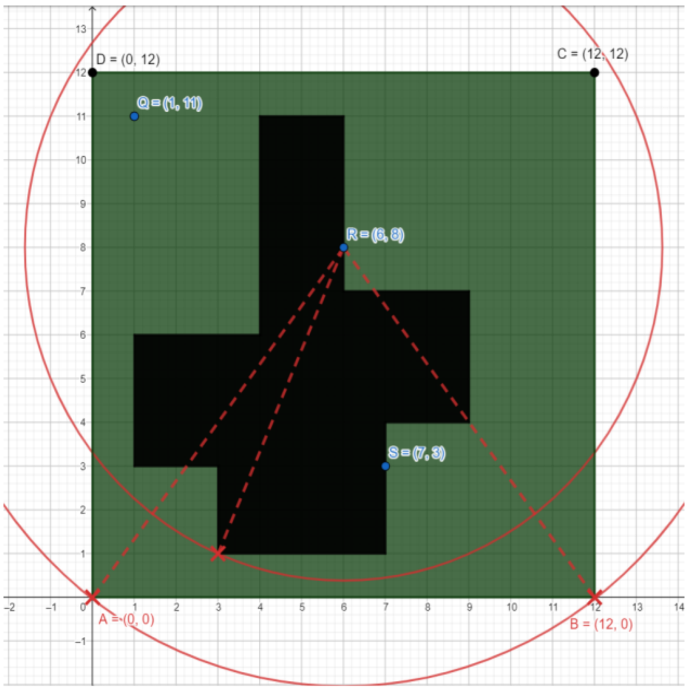
The first analyzed case was for the point Q(1,11). As described before, the circle with the center in Q was created and its radius enlarged until it cut through the area of the school just once. Then the same was done for the yard. Therefore the points marked with the red crosses (*Fig.3*) were discovered as creating the biggest distance.

Figure 3 – The points furthest away from Q



The second case was for the point R(6,8). We proceeded just like in the first case: a circle with the center in R was created and extended until it reached the final point. However, in this particular case, when creating the second circle to search for the point furthest away in the yard, it can be observed that actually two points fulfill the requirements (*Fig.4*)

Figure 4 – The points furthest away from R

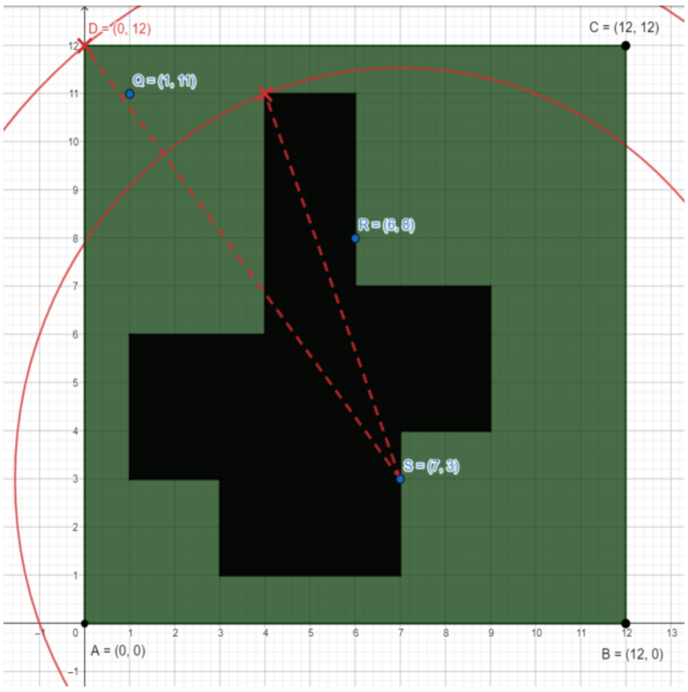


The third case for the point S(7,3). This case comes with no surprises. The procedure stays the same: the circle is created and enlarged until the entire school is in the circle, except for one point; and the same goes for the yard. See (*Fig.5*) for the solutions.

2.2.2 Which area is closest to each water point

The same idea can be adjusted in order to find which water point is closest to any point or area. A randomly chosen point T is created and then three circles, the distances between the water points and T being their radiuses. So, basically we combine all the three previous cases. The point is movable hence this solution answers the question for any point.

Figure 5 – The point furthest away from S



All the particular cases put together get to the core of the answer. If all the points closest to each water point are given different colors, the graphic below is achieved. Red represents all the points closest to R, blue- the ones closest to Q and orange for the ones within the closest range to S. In consequence, for any random point in the school the closest water point is given by the color of the point in *Fig.7* [3]

2.2.3 Using the perpendicular bisector

Definition. A perpendicular bisector of a line segment is a line that bisects the line segment at a right angle, through the intersection point. Thus, we can say, a perpendicular bisector always divides a line segment through its midpoint. The term bisect itself means dividing equally or uniformly.

The basic property of a perpendicular bisector is that it contains all the points equally distanced from the ends of the bisected segment. Sometimes the aforementioned property is taken as the definition of the perpendicular bisector, i.e. the perpendicular bisector of a segment is the geometric locus of all points in the plane that are equidistant from the end-points of the given segment. That implies the property of dividing the points from a plane

Figure 6 – The way we can find the closest water point

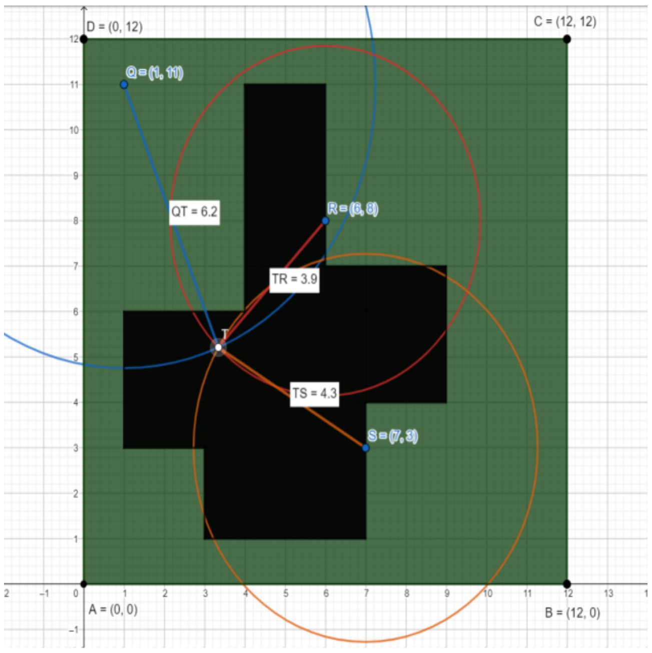
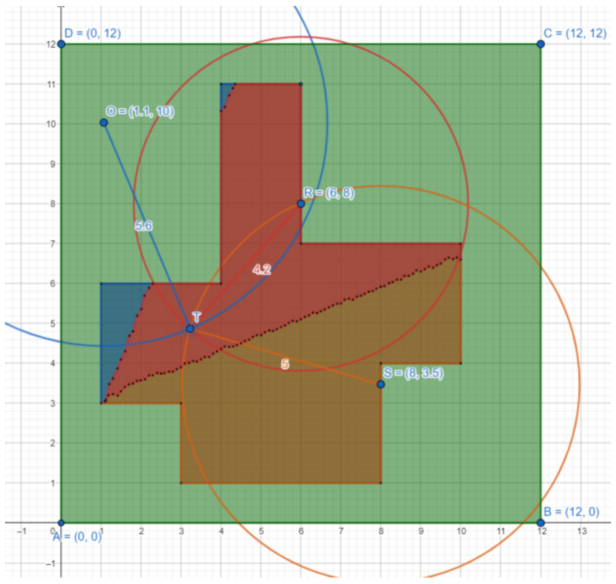
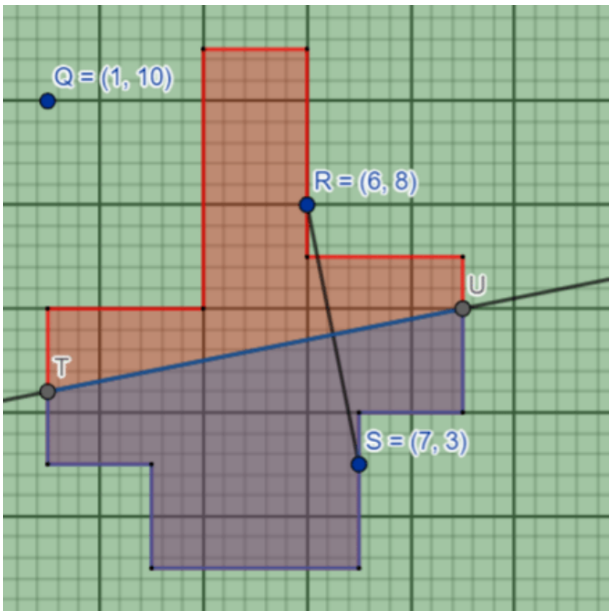


Figure 7 – The separation of points



into the ones closer to each end. For example, in the following image we consider the segment RS. On the same diagram, UT is its perpendicular bisector. That means all the points on the right of UT (colored in blue) are closer to S rather than R and the ones in the red area are closer to R.

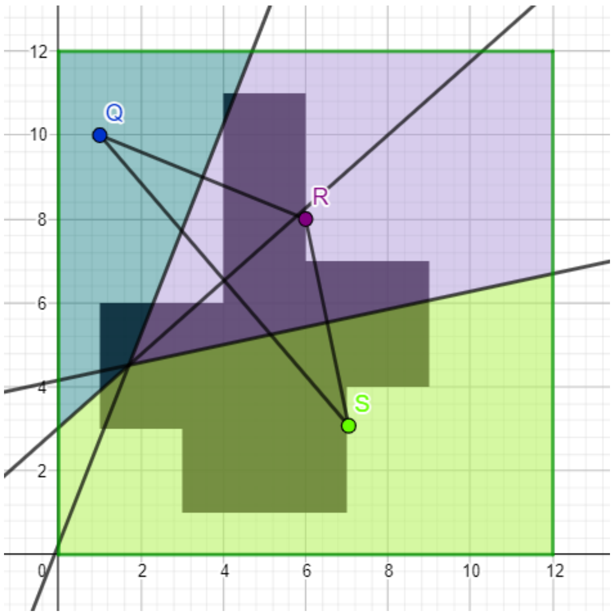
Figure 8 – The perpendicular bisector



This means that by using this method for any number of points [4] we will be able to divide the area in smaller sectors closer to each point, or in our case water points. These regions are called Voronoi cells. [1] [2] [3] Of course a problem appears when some points are in the area situated outside the walls. We considered that any points from the boundary of it are for watering plants. (See details in *Fig. 8*)

Now, let us generalise. We applied the previous idea for the given points and obtained this sketch as seen in *Fig. 9*. Naturally we can use more random points. As we mentioned this would indicate some wrong cases. These is essentialy due to limitations in the functionalities of Geogebra, where we did not know how to take walls into consideration when generating the separating regions. However, the authors managed to overcome this difficulty in the C++ implementation.

Figure 9 – The generalisation of the method



2.3 A solution using C++ programming

In this subsection, we present a C++ implementation of the solution described above. For doing so, the model of the school is discretized, namely we will consider a finite grid of points in the plane. The code will be separated in subprograms and specific parts which are explained. The C++ language was used because it is the one we were most familiar with.

2.3.1 Creating the matrix

Below, the first subprogram has been separated. As shown by its type and name it is a void subprogram that creates the shape of the school. It's a void because it does not have to return any value. Inside the function there are multiple for-loops, with i and j as variables that browse and create the exact needed shape. In our subprogram the coordinates given in the problem were used, but they can be easily changed for any random shape. In the end we will have a matrix where the points in the school are marked with 1 or 2 (for the water points) and the others with 0.

```
void school()
{
    for (int j = 11; j >= 6; j--)
    {
        for (int i = 4; i <= 6; i++) a[i][j] = 1;
    }
    for (int j = 7; j >= 6; j--)
    {
        for (int i = 6; i <= 9; i++) a[i][j] = 1;
    }
    for (int j = 6; j >= 4; j--)
    {
        for (int i = 1; i <= 9; i++) a[i][j] = 1;
    }
    for (int j = 4; j >= 3; j--)
    {
        for (int i = 1; i <= 7; i++) a[i][j] = 1;
    }
    for (int j = 3; j >= 1; j--)
    {
        for (int i = 3; i <= 7; i++) a[i][j] = 1;
    }
}
```

A slight issue appears here, regarding how many points are there. In the 2 dimensional space there are an infinity, but in C++ the “1”s will appear at every unit, so not an infinity, however, as mentioned above, in order to build a computer model we must restrict to a finite number of points. Instead of a continuous 2 dimensional model, we will work with a discrete finite grid consisting of points. Increasing the number of points in our discrete model, would increase the accuracy of the approximation of the continuous model. The next part is also a void function, and as shown by the name, it displays the matrix on the screen when called.

2.3.2 Finding the distance

The next function finds the distance between two points with the coordinates i1, j1 and i2, j2 in the matrix. It applies the basic math formula, which we showed previously. Unlike the ones explained before it is not a void because it returns d (the calculated distance). calculated.

```
double distanceBetweenTwoPoints(int i1, int j1, int i2, int j2)
{
    double d = (i1 - i2) * (i1 - i2) + (j1 - j2) * (j1 - j2);
    d = sqrt(d);
    return d;
}
```

2.3.3 Finding the closest area

This function consists in traversing the entire matrix, and for each point the distance from it to each water point is checked and the nearest water point is displayed. It is not the most efficient solution, but it will always give a good result. [5]

```
void theClosestArea()
{
    double l[1001];
    for (int j = 12; j >= 0; j--)
    {
        for (int i = 0; i <= 12; i++)
        {
            for (int k = 1; k <= n; k++)
            {
                l[k]=distanceBetweenTwoPoints(p[k].first,p[k].second,i,j);
            }
            int closestWaterPoint = minim(l, n);
            cout << closestWaterPoint << ' ';
        }
        cout << "\n";
    }
}
```

2.3.4 Finding the greatest distance between a water point and any point in the matrix

This function only applies 4.2 for calculating the distance between two points and it displays the maximum distance between each water point and any point in the matrix. As before, this program uses one repetitive structure in another, but this is the only way to search through one matrix.

```
        {
   double theGreatestDistance(int i1, int j1)
  {
        double maxd = 0;
        for (int j = 12; j >= 0; j--)
        {
            for (int i = 0; i <= 12; i++)
            {
                if (distanceBetweenTwoPoints(i1,j1,i,j) > maxd && a[i][j]==1)
                {
                    maxd = distanceBetweenTwoPoints(i1, j1, i, j);
                }
            }
        }
        return maxd;
   }
```

2.3.5 The main function

Finally, there is the main function. Here all the subprograms are put to use and the final result is shown. The algorithm works for any number of water points and any coordinates and that is the greatest advantage of all. Naturally there are the declarations and readings of the different variables. As explained in the comments alongside the code, water points are marked with 2 in the matrix and points other than water stations are marked with 1. Then, using the functions explained at 4.2, 4.3 and 4.4 it will display the nearest points and then the ones situated furthest away. This algorithms and functions can be used and altered for various requirements, for example if a fire starts in a point it can be quickly determined which water source is closest; or if the plants in the school yard need to be watered: the same way of applying the functions. Our next goal is to be able to color the points in the matrix with 3 or more colors given by the number of water points, in order to differentiate the areas closest to each water source.

```
int main()
{
    school(); ///points in the school, other than water points,
    are marked with 1
    cout << "Enter the number of water points and their coordinates:
    " << "\n";
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        cin >> p[i].first >> p[i].second;
        a[p[i].first][p[i].second] = 2; ///water points are marked
        with 2
    }
        display(); ///display of the school

    cout << "The closest area for each water point : " << "\n";
    theClosestArea();
    cout << "The greatest distance is : " << "\n";
    for (int i = 1; i <= n; i++)
    {
        cout << theGreatestDistance(p[i].first, p[i].second) << "\n";
    }
    return 0;
}
```

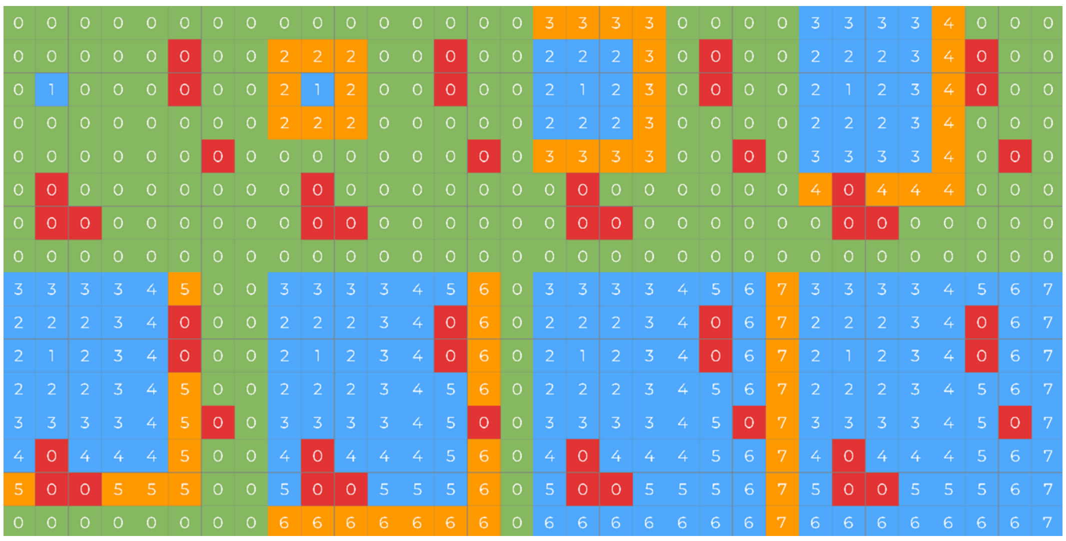
2.3.6 Using and adapting Lee's algorithm

Lee's algorithm [4] is one possible solution for maze routing problems. It always gives an optimal solution, if one exists, but is slow and requires large memory for dense layout.

Understanding how it works. One key concept to understand is that breadth-first searches go wide, while depth-first searches go deep. The algorithm is a breadth-first based algorithm that uses queues to store the steps. It usually uses the following steps:
1. Choose a starting point and add it to the queue.
2. Add the valid neighboring cells to the queue.
3. Remove the position you are on from the queue and continue to the next element.
4. Repeat steps 2 and 3 until the queue is empty. Using the example of a maze solving algorithm, a depth-first approach will try every possible path one by one until it reaches a dead end or the finish and returns the result. However the path it returns might not be the most efficient, but simply the first complete path to the finish that the algorithm was able to

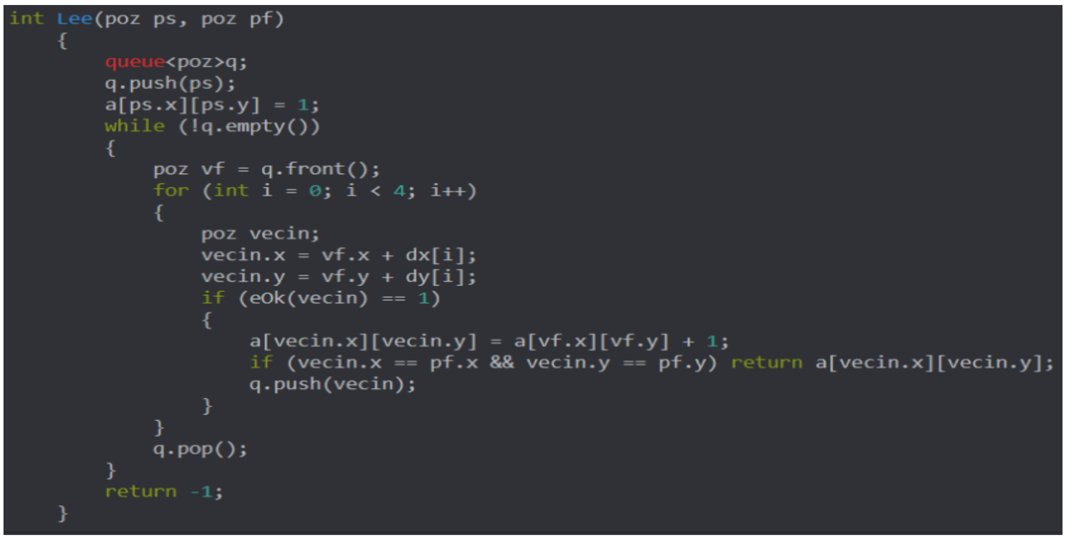
find. This basic use of the algorithm can be developed for solving more complex and realistic problems. For example, we can add as many starting points as we need. This is exactly the use of the algorithm we will use to solve our problem.

Figure 10 – How Lee's algorithm runs



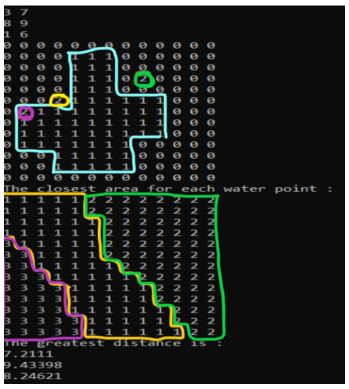
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	3	3	0	0	0	0	3	3	3	3	4	0	0	0
0	0	0	0	0	0	0	0	2	2	2	0	0	0	0	0	2	2	2	3	0	0	0	0	2	2	2	3	4	0	0	0
0	1	0	0	0	0	0	0	2	1	2	0	0	0	0	0	2	1	2	3	0	0	0	0	2	1	2	3	4	0	0	0
0	0	0	0	0	0	0	0	2	2	2	0	0	0	0	0	2	2	2	3	0	0	0	0	2	2	2	3	4	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	3	3	0	0	0	0	3	3	3	3	4	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	4	4	4	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	3	3	3	4	5	0	0	3	3	3	3	4	5	6	0	3	3	3	3	4	5	6	7	3	3	3	3	4	5	6	7
2	2	2	3	4	0	0	0	2	2	2	3	4	0	6	0	2	2	2	3	4	0	6	7	2	2	2	3	4	0	6	7
2	1	2	3	4	0	0	0	2	1	2	3	4	0	6	0	2	1	2	3	4	0	6	7	2	1	2	3	4	0	6	7
2	2	2	3	4	5	0	0	2	2	2	3	4	5	6	0	2	2	2	3	4	5	6	7	2	2	2	3	4	5	6	7
3	3	3	3	4	5	0	0	3	3	3	3	4	5	0	0	3	3	3	3	4	5	0	7	3	3	3	3	4	5	0	7
4	0	4	4	4	5	0	0	4	0	4	4	4	5	6	0	4	0	4	4	4	5	6	7	4	0	4	4	4	5	6	7
5	0	0	5	5	5	0	0	5	0	0	5	5	5	6	0	5	0	0	5	5	5	6	7	5	0	0	5	5	5	6	7
0	0	0	0	0	0	0	0	6	6	6	6	6	6	6	0	6	6	6	6	6	6	6	7	6	6	6	6	6	6	6	7

Figure 11 – How Lee's algorithm is written



```
int Lee(poz ps, poz pf)
    {
        queue<poz>q;
        q.push(ps);
        a[ps.x][ps.y] = 1;
        while (!q.empty())
        {
            poz vf = q.front();
            for (int i = 0; i < 4; i++)
            {
                poz vecin;
                vecin.x = vf.x + dx[i];
                vecin.y = vf.y + dy[i];
                if (eOk(vecin) == 1)
                {
                    a[vecin.x][vecin.y] = a[vf.x][vf.y] + 1;
                    if (vecin.x == pf.x && vecin.y == pf.y) return a[vecin.x][vecin.y];
                    q.push(vecin);
                }
            }
            q.pop();
        }
        return -1;
    }
```

How the code runs. The Geogebra implementation doesn't always output an accurate picture of the true answer, because we did not take into account that the school might have obstacles inside, such as walls. In *Fig.12 left*, we took some random coordinates for the water points. The point marked in yellow is the first water point and we can see that the program displays 1 for the points in the matrix that are closest to this water point. We marked with yellow the closest area for the first water point and did the same thing for the other two water points. In *Fig.12 right*, we took a random point in the matrix (the star) and three other water points. By using Lee's algorithm we took into account obstacles (school walls) and calculated the shortest path.



```
3 7
8 9
1 6
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 1 1 1 0 2 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 2 1 1 1 1 1 1 0 0 0
0 2 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
The closest area for each water point :
1 1 1 1 1 2 2 2 2 2 2 2 2
1 1 1 1 1 2 2 2 2 2 2 2 2
1 1 1 1 1 1 2 2 2 2 2 2 2
1 1 1 1 1 1 2 2 2 2 2 2 2
3 1 1 1 1 1 2 2 2 2 2 2 2
3 3 1 1 1 1 2 2 2 2 2 2 2
3 3 1 1 1 1 1 2 2 2 2 2 2
3 3 3 1 1 1 1 1 1 2 2 2 2
3 3 3 3 1 1 1 1 1 2 2 2 2
3 3 3 3 1 1 1 1 1 2 2 2 2
3 3 3 3 1 1 1 1 1 1 2 2 2
3 3 3 3 3 1 1 1 1 1 2 2 2
3 3 3 3 3 1 1 1 1 1 1 2 2
The greatest distance is :
7.2111
9.43398
8.24621
```

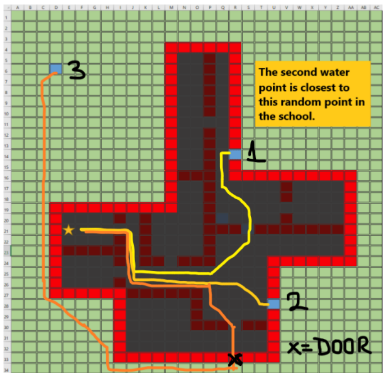
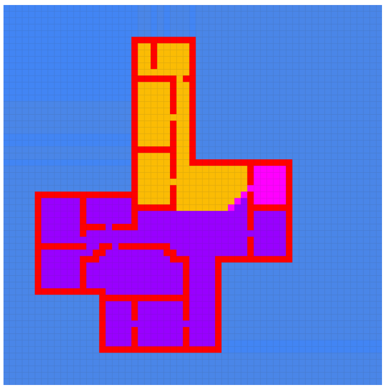

3
The second water point is closest to this random point in the school.
1
2
x=DOOR

Figure 12 – How the code runs: in the compiler(left) and in a random matrix(right)

We can use colors for a better understanding and visualization: the blue area is closest to the hydrant outside the school, the yellow area is closest to the top hydrant, the purple area is closest to the bottom hydrant and the pink area is placed at the same distance from both hydrants inside the building as seen in *Fig. 13*.

Figure 13 – The conclusion for some randomly chosen walls



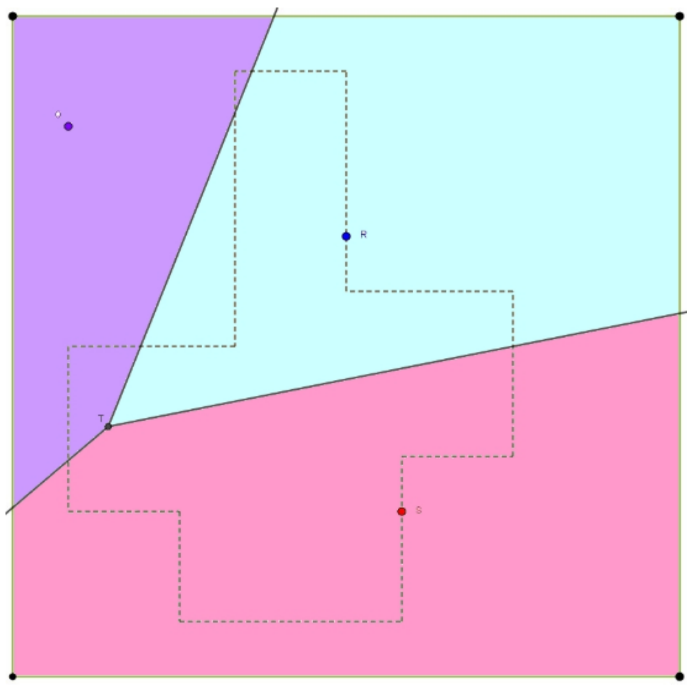
2.4 Limitations of study and future research

A main concern regarding our actual method of solving the problem geometrically regards the existence of walls or any other such obstacles. With coding the problem turns into a classic one, but we are still working on a solution using geometry to be able to also consider the walls and not ignore them. One idea is to use more analytic geometry concepts together with the idea of Voronoi cells. As future plans, besides the one described above we consider to make an application that simulates Lee's algorithm and also take into account the possibility of the school having several floors.

2.5 Another approach

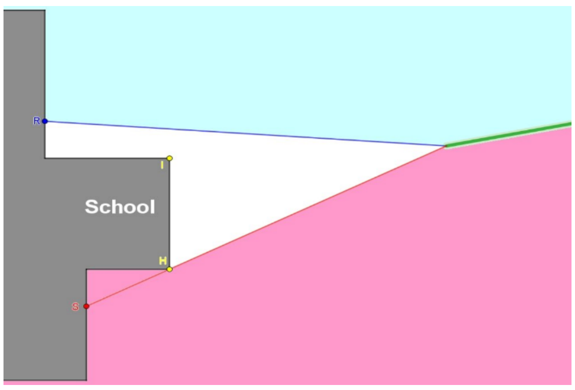
[6] To solve our problem without the issue of the school, we just need to trace three segments [QR], [RS] and [SQ]. Then we need to trace the perpendicular bisector of each segment. After having defined the three segments, we get the point T, i.e. the circumcenter of QRS, which is the intersection of the three segments. In this way, we have defined three areas. But the problem is harder to solve because of the school's walls. Our research was mostly based on the shape of the borders between the zone of the faucet R and the one of the faucet S. Our results give three parts depending on whether the pipe touches the wall or not.

Figure 14 – The medians(perpendicular bisectors)



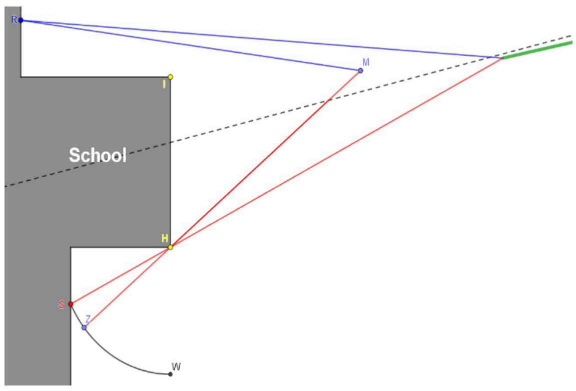
In this case (if the pipes don't touch the corners), the frontier between the two zones is a part of the median. In this picture we can see the blue area which represents the area from the watering point R because every point is closer to R than S. And in pink the area which belongs to the watering point S. But there still are the corners I and H which give us a more complicated problem to define the white area which can't be defined because of the median.

Figure 15 – The frontier



When only the pipe of the red's faucet touches the corner of the school and not the blue one: we make a rotation which has as center the point H of the faucet S in Z with Z, H and M being aligned. In consequence we have SH+HM=ZH+HM=ZM. Then we make the median of RZ, the line HZ and the intersection will be our border (remember that Z is movable on the arc).

Figure 16 – The rotation



The weakness of this method is that it does not consider if the two pipes touch the two corners of the school. We are going to make the distance E1 which corresponds to RI-SH. We turn this distance into an arc with a ray corresponding to the distance E1, taking the point I as the center, the point that won't move. We take the intersection of the median of [HF1] with the half-line [F1I) to find G1 who's at the same distance to the two faucets. Then we make

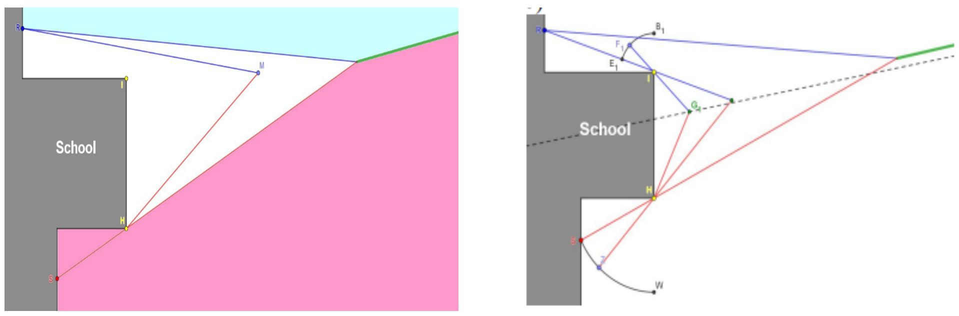
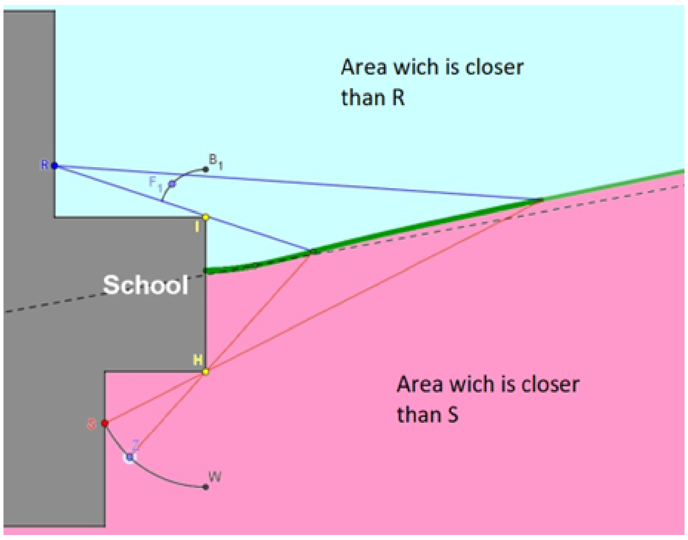


Figure 17 – The arc

the same process with the other watering point to define all the limits.
As you can see in *Fig.18*, the limit between the area from R and S (green sorts) isn't a line but a curved line composed with 3 parts.

Figure 18 – The limit between areas



3 Conclusion

The results obtained with the two entirely different methods can be observed both in *Fig. 13* and *Fig.18*.

References

[1] de Berg, Mark; van Kreveld, Marc; Overmars, Mark; Schwarzkopf, Otfried (2008). Computational Geometry (Third ed.). Springer-Verlag. ISBN 978-3-540-77974-2. 7.4 Farthest-Point Voronoi Diagrams. Includes a description of the algorithm.

[2] Burrough, Peter A.; McDonnell, Rachael; McDonnell, Rachael A.; Lloyd, Christopher D. (2015). "8.11 Nearest neighbours: Thiessen (Dirichlet/Voroni) polygons". Principles of Geographical Information Systems. Oxford University Press. pp. 160–. ISBN 978-0-19-874284-5.

[3] Okabe, Atsuyuki; Boots, Barry; Sugihara, Kokichi; Chiu, Sung Nok (2000). Spatial Tessellations – Concepts and Applications of Voronoi Diagrams (2nd ed.). John Wiley. ISBN 978-0-471-98635-5.

[4] Lee, C. Y. (1961), "An Algorithm for Path Connections and Its Applications", IRE Transactions on Electronic Computers, EC-10 (2): 346–365, doi:10.1109/TEC.1961.5219222

EDITION NOTES

[1] The statement of the problem is not very precise: are we interested in points both in the garden and within the building? Since the real distance inside the school depends also from walls, staircases, etc. the Lee algorithm described below should be mentioned.

[2] This paragraph is rather obscure. It can be replaced by:

“We can create a circle with the center in a water point and enlarge it until the area of the school is intersected in only one point and the rest of the school is inside it. The radius of the circle obtained in this way is the greatest distance between the used water point and a point in the school.”

[3] This solution ignores walls, doors, staircases, etc. And what about the garden?

[4] This point would deserve a deeper explanation. The present situation, with only 3 water points, is simple because the three perpendicular bisectors intersect in one point (the circumcenter). What happens with more than 3 points? With 4 points, for instance, we have 6 bisectors.

[5] In what sense the result is good?

[6] The position of this section in the paper suggests that it was conceived as the beginning of future researches, and hence that an exhaustive treatment should not be expected. More detailed explanations are actually needed. In particular, the role of the arcs centered in H and I should be clarified.