Assignment 2

Voronoi Diagram: The Fundamentals

In the last assignment you have implemented the DCEL. We used it to store the regions each watchtower was responsible for. However, if you are given a point location, how do you find the nearest watchtower? An obvious solution is to compute the distance to all watchtowers and to select afterward the closest one (pick one randomly if there is more than one). This works fine if you have a single query but the cost is of course $O(mn)$ for m queries if there are n watchtowers.

An alternative approach is to precompute the region of all points that is closer to a watchtower than to all other watchtowers. This region is called the *Voronoi region* or *Voronoi cell* of that watchtower. If we computed the Voronoi cell for each watchtower (note that this region is unique), then we could simply lookup the region that contains our location and know the responsible watchtower. The planar subdivision of all Voronoi cells is called the ***Voronoi diagram***.

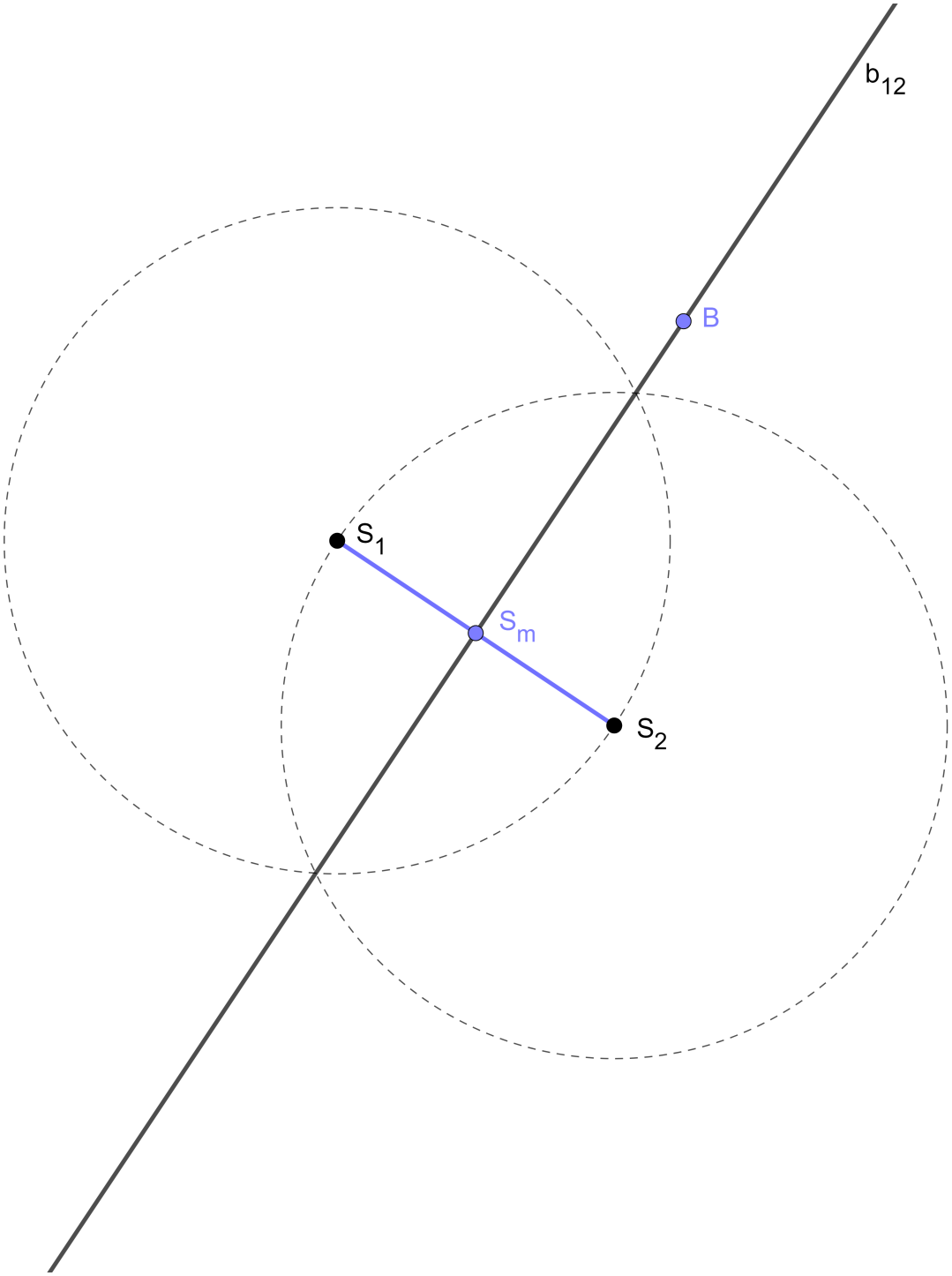
Bisector

An important concept for Voronoi diagrams is the bisector of two points. The bisector is orthogonal to the line segment connecting the two points and is equidistant to both points. If you had a compass, you would simply center the compass on each site, draw a circle (it does not have to have the other site on its circumference but the circles need to overlap to generate at least one intersection). Of course, for an implementation, we need an actual formula. The point S_m is easy to compute as it is just the midpoint of S_1 and S_2, i.e.,

$$S_m = (\frac{S_{1x} + S_{2x}}{2}, \frac{S_{1y} + S_{2y}}{2}).$$

The actual bisector is just a usual straight line and its formula is

$$y = -\frac{S_{2x} - S_{1x}}{S_{2y} - S_{1y}} \times (x - S_{mx}) + S_{my}.$$



Note that every point on the bisector is equidistant to the sites S_1 and S_2. In particular, it divides the line segment $\overline{S_1 S_2}$ into two equal halves at point S_m. Finally, the line segment $\overline{S_1 S_2}$ is orthogonal to the bisector b_{12}.

Voronoi cell

In this assignment, we will use the DCEL to store the Voronoi diagram. More formally, in general we have n sites $S_1, \ldots, S_n$ (our watchtowers), then the ***Voronoi cell*** of a site S_i for a given region R (say Victoria) is defined as the set of points P in the given region R that fulfil the following condition:

$$VC(S_i) = \{P \in R \mid dist(P, S_i) < dist(P, S_j)\ \forall S_j, j \neq i\}.$$

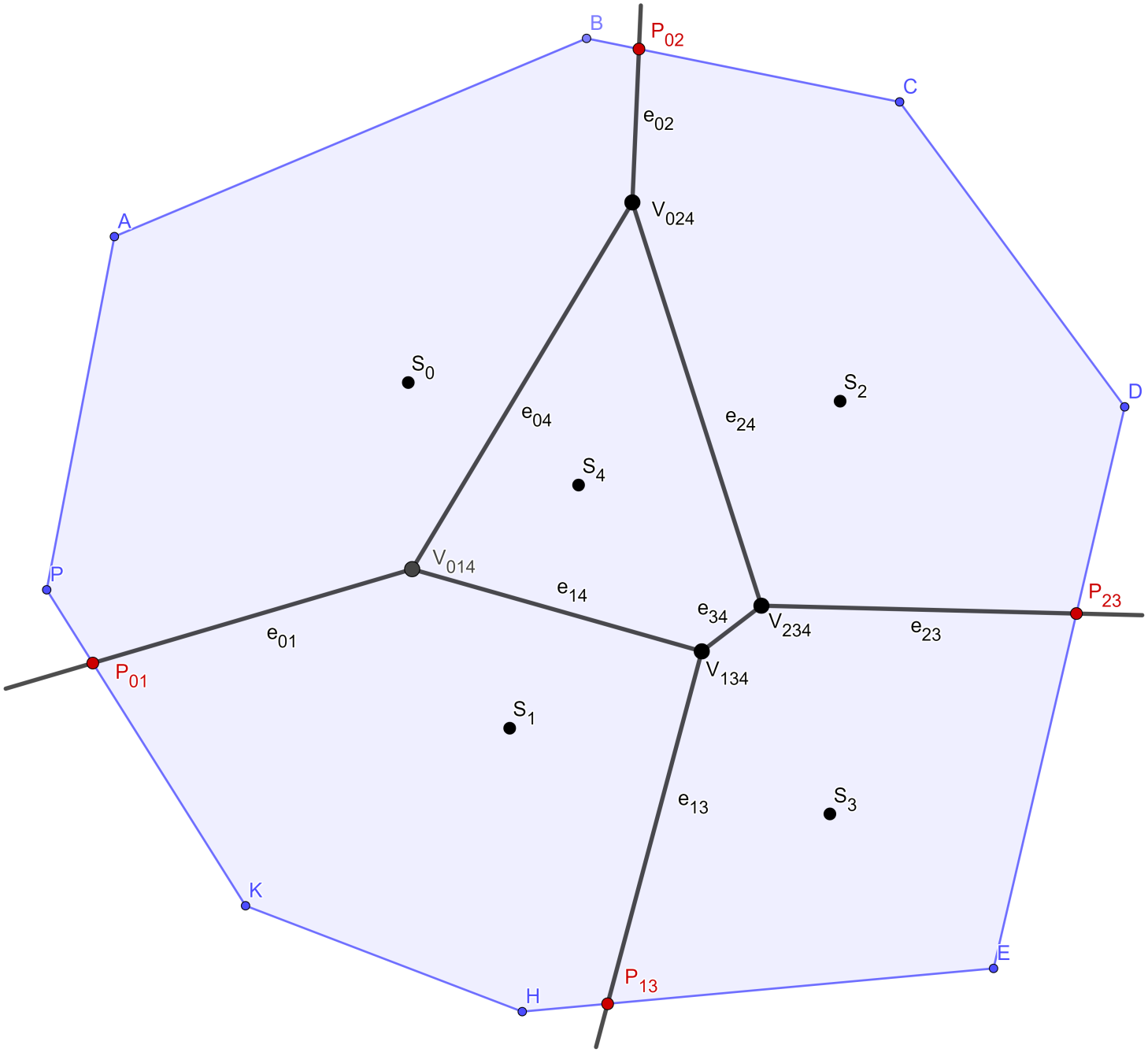
$dist(\cdot, \cdot)$ is the usual Euclidean distance between two points.

Voronoi edge and Voronoi vertex

A shared edge between two Voronoi cells is called a ***Voronoi edge***. If S_i and S_j are two sites whose Voronoi cells share an edge e_{ij}, then all the points on e_{ij} are equidistant to S_i and S_j. This means that a Voronoi edge is part of the perpendicular bisector between two sites S_i and S_j. Voronoi cells can share at most one Voronoi edge. In the figure the sites S_0 and S_4 share edge e_{04}.

A point at which the edges of three (or more) Voronoi cells meet is called a ***Voronoi vertex***. If S_i, S_j and S_k are three sites with shared Voronoi edges e_{ij}, e_{ik} and e_{jk} that meet in V_{ijk}, then the Voronoi vertex is the circumcentre of the triangle with vertices S_i, S_j and S_k, because the points S_i, S_j and S_k are all equidistant to V_{ijk}.

For example, the vertex V_{234} is shared by the edges e_{23}, e_{24} and e_{34} in the figure below.



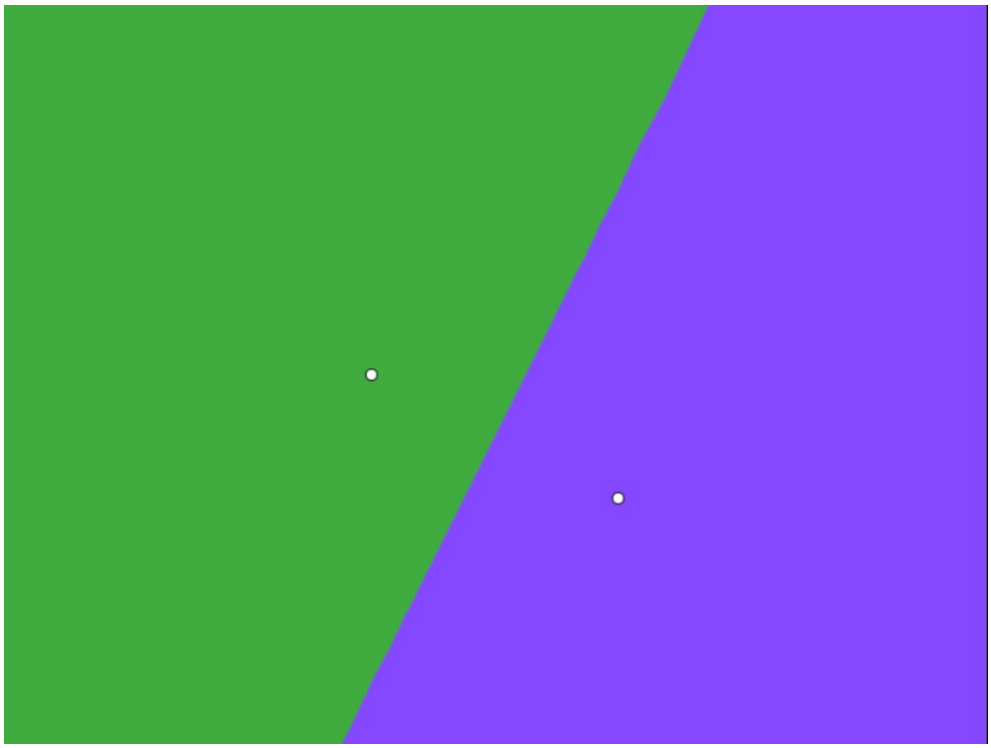
Usually, there are two types of Voronoi edges: those that connect two Voronoi vertices and those

start from a single vertex and are unbounded, i.e., are infinite. We will assume a (convex) polygon around our Voronoi sites, which clips all infinite edges. This means you can assume that all Voronoi edges are bounded and you use for the second point on the edge simply the intersection points of the polygon with the Voronoi edges.

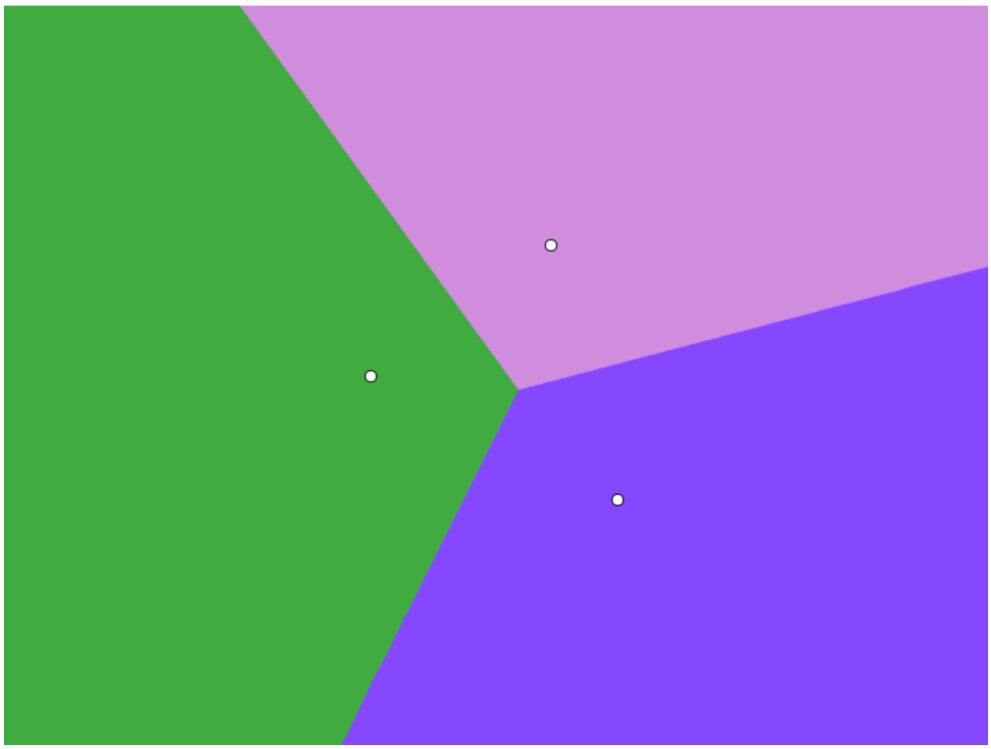
In the figure above, the Polygon is defined by the points A to P and the unbounded Voronoi edges e_{01}, e_{02}, e_{23}, e_{13} are described as line segments $\overline{V_{014}P_{01}}$, $\overline{V_{024}P_{02}}$, $\overline{V_{234}P_{23}}$, $\overline{V_{134}P_{13}}$, respectively.

By the way: there are as many Voronoi cells as we have sites, and if we have n sites, then there are $O(n)$ vertices and edges.

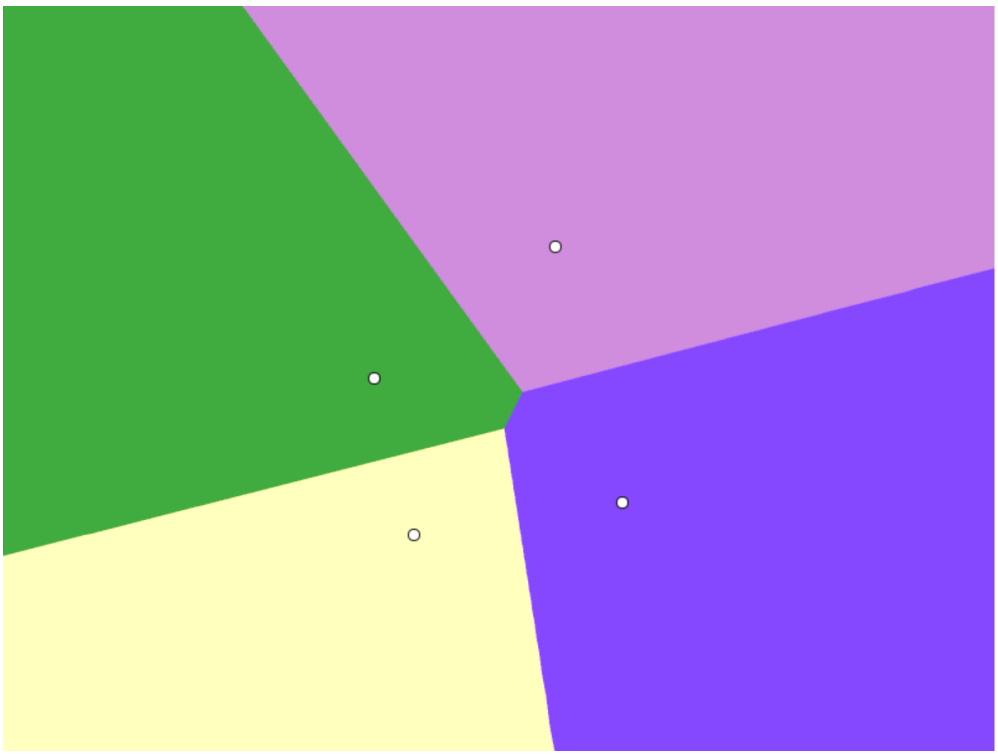
If we had two sites S_1 and S_2, what would the Voronoi diagram look like assuming a bounding rectangle as a polygon? Here is an example:



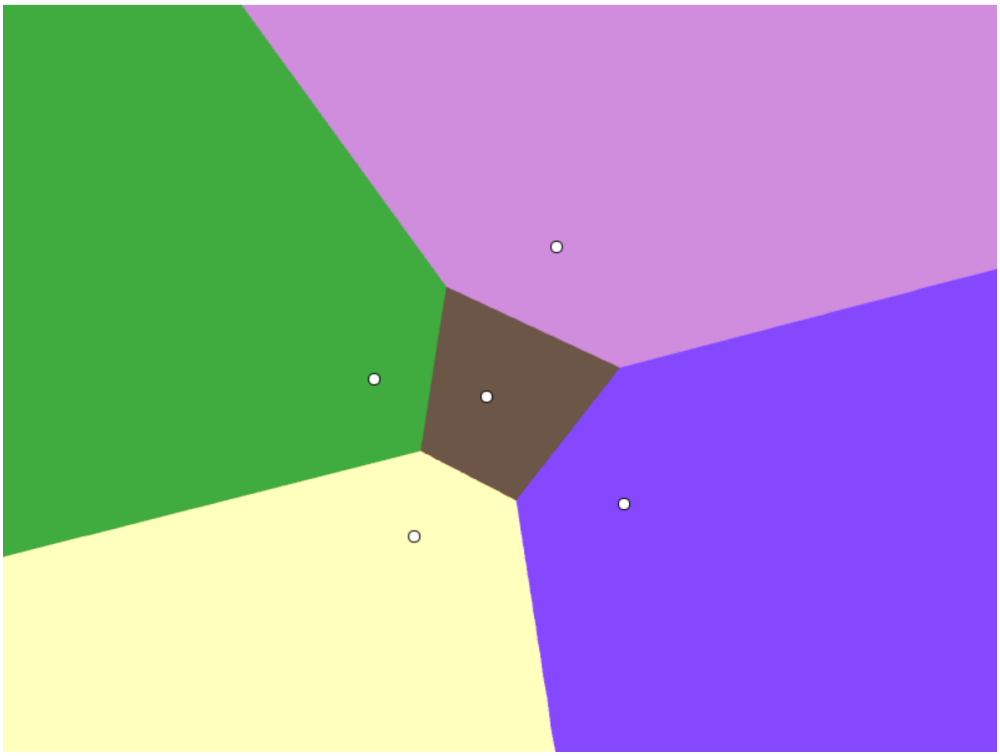
The two Voronoi cells are simply the result of inserting the bisector between those two sites. All the points in the green cell are closer to the left site, and all the points in the purple cell are closer to right site. In the next figure, we have inserted a third site. Since we have three sites, we get exactly one Voronoi vertex.



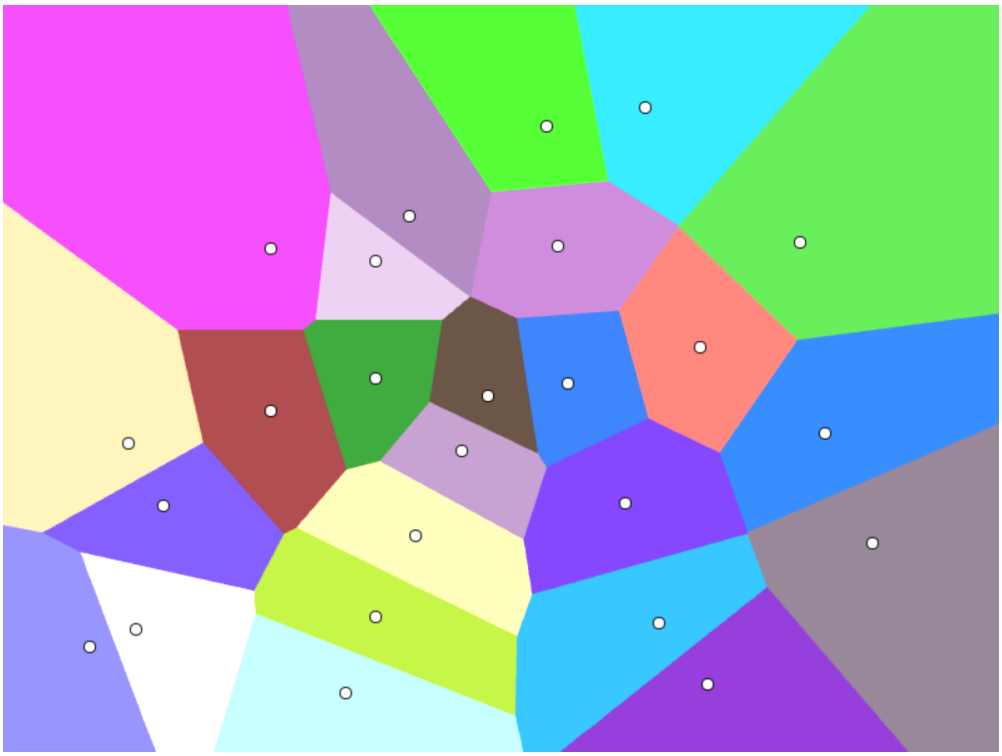
We insert another site and obtain four Voronoi cells.



After inserting another site, we get the first Voronoi cell that has no infinite edges (the brown cell).



This is the Voronoi diagram after inserting 20 more sites (25 in total):



Here is a movie that shows that the insertion of another site only impacts a few selected Voronoi cells in its neighborhood.

A final note: finding the Voronoi cell for a given point location can be achieved in $O(\log n)$ time given n sites. The reason is that every Voronoi cell is convex. However, we will not implement this algorithm here but it shows that the original problem of locating m points -- stated in the introduction above -- can be done in $O(m \log n)$.

Task 1: Compute and output equations for bisectors

The description above motivated the use of an incremental algorithm to compute the Voronoi diagram. We first build the Voronoi diagram for 3 sites through the use of bisectors. Bisectors were described in the previous entry. Your task is to compute and output equations for bisectors given pairs of points in a file.

Your implementation will receive three arguments, one for the stage and two filenames. The first filename argument is a file that will contain a list of point pairs, one pair per line where the x and y coordinates are separated by a blank. The second filename argument is an output file that contains the equations for all bisectors.

Example Input

An example of the file containing a list of point pairs is:

```
145.6 -34.2 145.6 -35.2
145.6 -34.2 145.6 -36.2
145.6 -35.2 148.6 -35.2
147.6 -35.2 146.6 -35.2
148.6 -35.2 146.6 -35.2
148.6 -34.2 146.6 -32.2
```

Example Output

After running `./voronoi2 1 pp_inside.txt 1-outfile-inside.txt` the contents of 1-outfile-inside.txt would be:

```
y = 0.000000 * (x - 145.600000) + -34.700000
y = 0.000000 * (x - 145.600000) + -35.200000
x = 147.100000
x = 147.100000
x = 147.600000
y = 1.000000 * (x - 147.600000) + -33.200000
```

Task 2: Compute and output intersection points for bisectors against a given polygon

In this task you will output the intersections between bisectors and a given polygon. Your implementation will receive four arguments, one representing the task number and three filenames. The first filename argument is a file that will contain a list of point pairs, one pair per line where the x and y coordinates are separated by a blank. The second argument is a file that will contain the initial polygon to be stored as a DCEL. You have developed the DCEL in assignment 1 but we have provided a DCEL implementation for you that you might like to use instead. The third argument is an output file that contains all intersections of the bisectors with the provided polygon. These will specify which edge in the DCEL the bisectors intersected, and the points these occurred at.

Example Input

An example of the file containing a list of point pairs is:

```
145.6 -34.2 145.6 -35.2
145.6 -34.2 145.6 -36.2
145.6 -35.2 148.6 -35.2
147.6 -35.2 146.6 -35.2
148.6 -35.2 146.6 -35.2
148.6 -34.2 146.6 -32.2
```

An example of the polygon file is:

```
140.9 -39.2
140.9 -33.9
150.0 -33.9
150.0 -39.2
```

Example Output

After running `./voronoi2 2 pp_inside.txt polygon_square.txt 2-outfile.txt` the contents of 2-outfile.txt would be:

```
From Edge 0 (140.900000, -34.700000) to Edge 2 (150.000000, -34.700000)
From Edge 0 (140.900000, -35.200000) to Edge 2 (150.000000, -35.200000)
From Edge 1 (147.100000, -33.900000) to Edge 3 (147.100000, -39.200000)
From Edge 1 (147.100000, -33.900000) to Edge 3 (147.100000, -39.200000)
From Edge 1 (147.600000, -33.900000) to Edge 3 (147.600000, -39.200000)
From Edge 1 (146.900000, -33.900000) to Edge 3 (141.600000, -39.200000)
```

Provided Intersection Code

The intersection code is non-trivial, and even a single error is difficult to spot visually. We have provided this code for you here, you are welcome to treat it as a magic black box - the areas to fill in are "..." . You just need to add in your half-edges start and end, and bisector segment start and end. This gives a more detailed diagnosis, but it is sufficient to check if it DOESNT_INTERSECT to determine intersection.

```c
    SAME_LINE_OVERLAP = 2, // Lines are the same
    ENDS_OVERLAP      = 3  // Intersects at exactly one point (endpoint
};

/*
This intersection is based on code by Joseph O'Rourke and is provided f
COMP20003 Assignment 2.

The approach for intersections is:
- Use the bisector to construct a finite segment and test it against th
- Use O'Rourke's segseg intersection (https://hydra.smith.edu/~jorourke
    to check if the values overlap.
*/
/*
```

Task 3: Computation of the Voronoi Diagram

An incremental algorithm to compute the Voronoi diagram

How do we enhance our algorithm to work with $n > 3$ sites assuming we have a Voronoi diagram for 3 sites already? The algorithm works as follows: assume a new site S_m (see the Figure below) is inserted into a Voronoi diagram that has already k sites. The Voronoi cell of S_m is then created as follows:

- Find the Voronoi cell $VC(S_i)$ that contains S_m.
- Compute the bisector b_{im} of S_i and S_m.
- The bisector b_{im} intersects two edges of $V(S_i)$, say e_{i_0} and e_{i_1} in counter-clockwise direction.
- If both edges are Voronoi edges, i.e., have a twin (opposite, pair) edge, then the algorithm proceeds as follows:
    - Store the new Voronoi edge of $VC(S_m)$ that connects e_{i_0} and e_{i_1} at their intersection points.
    - Retrieve the Voronoi site using the DCEL of the edge e_{i_1}, say S_{i_1}.
    - Process the second edge intersection (in counter-clockwise direction) compute the second edge of S_{i_1} that intersects $b_{i_1 m}$, say e_{i_2}.
    - If the next edge is also a Voronoi edge, retrieve the Voronoi site using the DCEL of the edge e_{i_2}, say S_{i_2}. If every encountered edge is a Voronoi edge, repeat the algorithm until $e_{i_j} = e_{i_0}$.
    - If the algorithm intersects at any stage an edge that is not a Voronoi edge, i.e., an edge of the enclosing polygon, it terminates this search for further Voronoi edges. *Note that could happen even at the beginning and we will have only Voronoi vertex because the Voronoi cell would be unbounded if we did not assume an initial polygon.*
    - Instead, the algorithm may intersect a non-Voronoi edge and continue its search along the initial polygon and terminates its search until it visits e_{i_0} again.

Incrementally updating the DCEL

Of course you need to update all edges in the DCEL, whenever you compute an edge for S_m. You will also need to delete all old Voronoi vertices that are enclosed by the new Voronoi cell $VC(S_m)$. There are basically two cases that will happen:

- The bisector intersects two edges that are adjacent, i.e., share a single vertex. Then you need to delete the shared vertex, split the intersected edges and store the updated edges for the Voronoi cell that is intersected by the bisector.
- The bisector intersects two non-adjacent edges. Then you need to traverse all edges -- starting from the second intersected edges -- in clockwise order using the *.next* operation until you

encounter the first intersected edge. All edges and their shared vertices have to removed. Then you need to split the intersected edges as before and and store the updated edges for the Voronoi cell that is intersected by the bisector.

Finally, you need to insert the new site into the DCEL including the new edges and their start points.

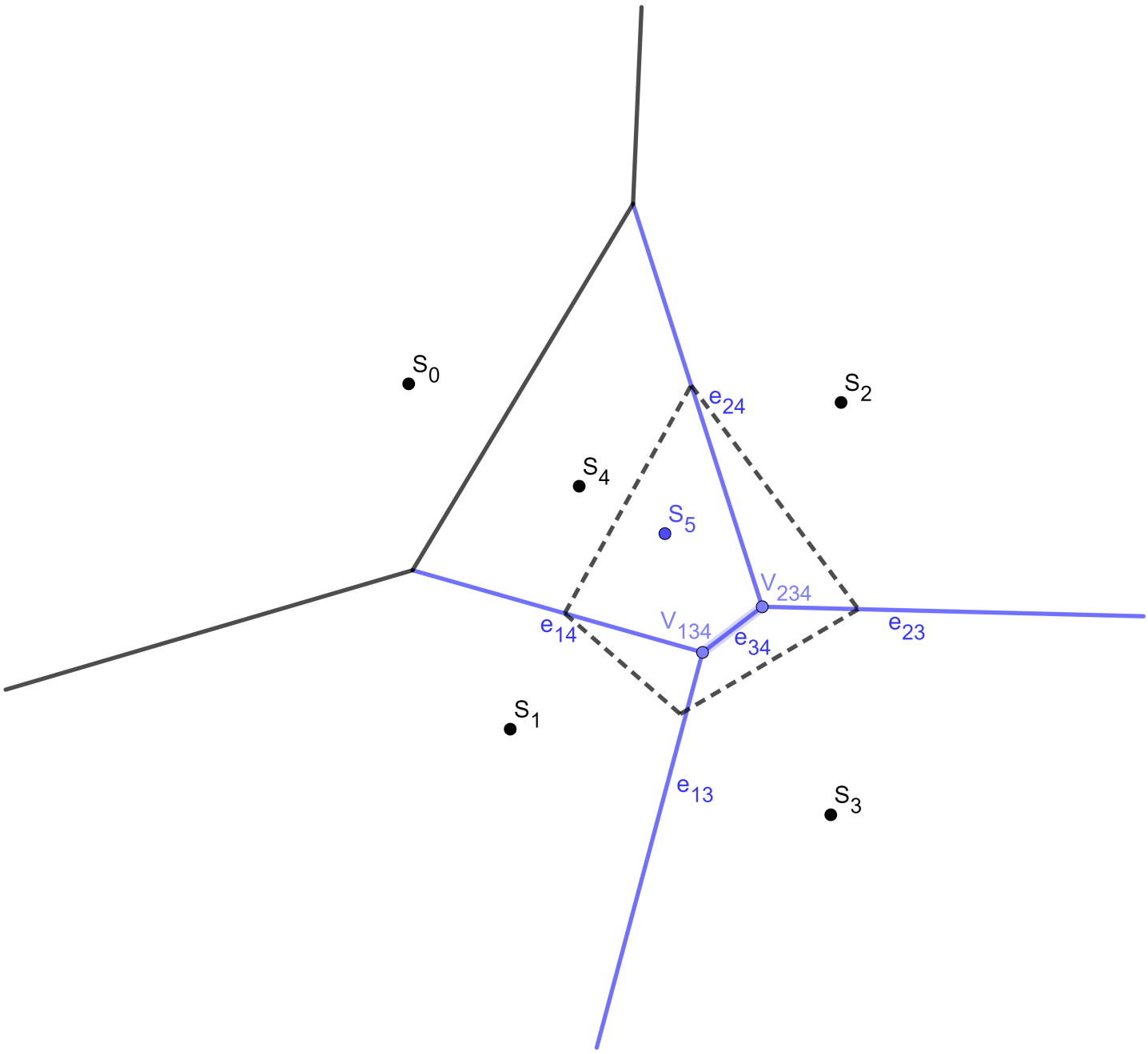
An example

In the example below we have already 5 existing sites $S_0, \ldots, S_4$ and wish to insert site S_5.

- In the first step we compute that S_5 is in $VC(S_4)$ and compute the bisector of S_5 and S_4, which intersects the edges e_{24} and e_{14} in counter-clockwise order.
- Since e_{14} is edge between S_4 and S_1, we compute the bisector between S_1 and S_5, which intersects edge e_{13}. This implies that the next site is S_3.
- We then apply the algorithm and compute the next intersection of the bisector of S_5 and S_3, which is the edge e_{23}. Thus, the next site is S_2 and the intersection of the bisector S_5 and S_2 is the edge e_{24}.
- Since we have discovered the edge e_{24} before, the algorithm terminates.

We now have to update all purple edges (and vertices), and have to insert the new edges, highlighted as dashed edges.

- This means that we have to delete the vertices V_{134} and V_{234}.
- We also have to delete the edge e_{34} connecting V_{134} and V_{234}.
- Finally, we have to apply the corresponding split operations you have studied in assignment 1 on e_{24}, e_{34} and e_{14}.
- Finally, we insert the new dashed edges e_{45}, e_{25}, e_{35} and e_{15} into the DCEL for the Voronoi cell $VC(S_5)$.



The diameter of a Voronoi cell

You will need to retrieve all Voronoi cells from the DCEL and compute their diameter. The ***diameter*** of a set S is the least upper bound of all distances between point pairs in S. Fortunately, all Voronoi cells are convex polygons (remember that a set is convex if for every point pair the segment connecting the points is also in the set). This means that the diameter is easy to compute:

> **i** You just need to compute the distances of all pairs of vertices of a Voronoi cell and select the largest distance.

It is clear that this algorithm has quadratic complexity in the number of vertices of a Voronoi cell and we have n Voronoi cells for n sites.

If you are curious: there are faster ways to compute the diameter of a Voronoi cell (or in fact any convex polygon) that are based on the concept of supporting lines. However, this proved to be more difficult than initially thought and a few incorrect algorithms have been published as a consequence!

This shows again how important it is to verify the correctness of your algorithms. You can find more information about this here.

Your task

Your task is to compute the Voronoi diagram iteratively. In addition to the argument specifying your program should run task 3, your implementation will receive three filenames as an arguments and will build the Voronoi diagram reading from the first two files and outputting the site/watchtower data and the diameter of each Voronoi cell to the output file.

- The first file contains all Voronoi sites, i.e., the watchtower from the first assignment, again stored in csv format, one per line, representing the fields associated with each site. Again, your program must read in the records line by line.
- The second file will contain a list of (x, y) coordinates that describe the vertices of a region R (such as the state of Victoria) as a polygon. Each vertex (x, y) of the region's boundary is stored on a separate line. The coordinates are separated by a space on each line.
- After processing the second input file, a Voronoi diagram is constructed, separating all points into their own cell with all points in the cell being closest to the site/watchtower in the cell. The site/watchtower data and its diameter(s) are written to the output file.

Note

The algorithm described above makes a few assumptions to avoid dealing with special cases:

- Not three watchtowers are collinear.
- No Voronoi vertex has more than three Voronoi edges.
- The initial polygon is large enough to contain all Voronoi vertices.

Running Task 3

When run with the number 3 as the first argument, your program should take four arguments. The first argument is this task indicator. The other three arguments are files with the same meaning as the first assignment. The second argument will be the filename of a csv-format list of watchtowers with the same structure as the first assignment, the third argument will be a list of points, one per line, with each coordinate separated by a single space. The fourth argument will be the file to output to.

Example Input

An example of the file containing a list of watchtowers:

```
Watchtower ID,Postcode,Population Served,Watchtower Point of Contact Name,x,y
WT3953SGAEI,3953,1571,Ofelia Kadlec,145.77800174296402,-38.55984015166651
WT3765SHSPB,3765,3380,Eilene Horner,145.36201379669092,-37.81894302945288
WT3530RJWDT,3530,63,Troy Clark,143.0834668479817,-35.79299394885817
```

An example of the polygon file is:

```
140.9 -39.2
140.9 -33.9
150.0 -33.9
150.0 -39.2
```

Example Output

After running `./voronoi2 3 dataset_3.csv polygon_square.txt 3-outfile.txt` the contents of 3-outfile.txt would be:

```
Watchtower ID: WT3953SGAEI, Postcode: 3953, Population Served: 1571, Watchtower Point of Contact N
ame: Ofelia Kadlec, x: 145.778002, y: -38.559840, Diameter of Cell: 7.144748
Watchtower ID: WT3765SHSPB, Postcode: 3765, Population Served: 3380, Watchtower Point of Contact N
ame: Eilene Horner, x: 145.362014, y: -37.818943, Diameter of Cell: 9.518041
Watchtower ID: WT3530RJWDT, Postcode: 3530, Population Served: 63, Watchtower Point of Contact Nam
e: Troy Clark, x: 143.083467, y: -35.792994, Diameter of Cell: 7.935830
```

Note the addition of the **Diameter of Cell** field.

Task 4: Computing the diameter of all Voronoi cells and sort them in ascending order by diameter

Your task has the same input and output files as task 3 but this time you need to output the sites/watchtowers in order of the length of the diameter of their corresponding Voronoi cell in ascending order, i.e., smallest to largest. Your sorting algorithm has to be stable, which means that for two sites with equal diameter the one with a smaller ID is stored first.

To sort the Voronoi cells by diameter, you will implement *Insertion Sort*. Here is its pseudocode (note that the ⟵ sign is used to assign values to a variable):

```
InsertionSort(A[0..n - 1])

for i ← 1 to n - 1 do

    v ← A[i]
    j ← i - 1

    while j >= 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j - 1

    A[j + 1] ← v
```

The idea of *Insertion Sort* is that we assume that a smaller problem of sorting the array $A[0..k-2]$ has already been solved. We take advantage of that and simply insert a new element $A[k-1]$ at the appropriate position so that the array $A[0..k-1]$ is now sorted. Note that the basic operation is the key comparison $A[j] > v$. Please convince yourself that this algorithm is indeed stable.

Example Input

An example of the file containing a list of watchtowers:

```
Watchtower ID,Postcode,Population Served,Watchtower Point of Contact Name,x,y
WT3953SGAEI,3953,1571,Ofelia Kadlec,145.77800174296402,-38.55984015166651
WT3765SHSPB,3765,3380,Eilene Horner,145.36201379669092,-37.81894302945288
WT3530RJWDT,3530,63,Troy Clark,143.0834668479817,-35.79299394885817
```

An example of the polygon file is:

```
140.9 -39.2
140.9 -33.9
150.0 -33.9
150.0 -39.2
```

Example Output

After running `./voronoi2 4 dataset_3.csv polygon_square.txt 4-outfile.txt` the contents of 4-outfile.txt would be:

```
Watchtower ID: WT3953SGAEI, Postcode: 3953, Population Served: 1571, Watchtower Point of Contact N
ame: Ofelia Kadlec, x: 145.778002, y: -38.559840, Diameter of Cell: 7.144748
Watchtower ID: WT3530RJWDT, Postcode: 3530, Population Served: 63, Watchtower Point of Contact Nam
e: Troy Clark, x: 143.083467, y: -35.792994, Diameter of Cell: 7.935830
Watchtower ID: WT3765SHSPB, Postcode: 3765, Population Served: 3380, Watchtower Point of Contact N
ame: Eilene Horner, x: 145.362014, y: -37.818943, Diameter of Cell: 9.518041
```

Plagiarism

This is an individual assignment. The work must be your own.

While you may discuss your program development, coding problems and experimentation with your classmates, you must not share files, as this is considered plagiarism.

If you refer to published work in the discussion of your experiments, be sure to include a citation to the publication or the web link.

“Borrowing” of someone else’s code without acknowledgment is plagiarism. Plagiarism is considered a serious offense at the University of Melbourne. You should read the University code on Academic integrity and details on plagiarism. Make sure you are not plagiarizing, intentionally or unintentionally.

You are also advised that there will be a C programming component (on paper, not on a computer) in the final examination. Students who do not program their own assignments will be at a disadvantage for this part of the examination.

Late Policy

The late penalty is 10% of the available marks for that project for each day (or part thereof) overdue. Requests for extensions on medical grounds will need to be supported by a medical certificate. Any request received less than 48 hours before the assessment date (or after the date!) will generally not be accepted except in the most extreme circumstances. In general, extensions will not be granted if the interruption covers less than 10% of the project duration. Remember that departmental servers are often heavily loaded near project deadlines, and unexpected outages can occur; these will not be considered as grounds for an extension.

Students who experience difficulties due to personal circumstances are encouraged to make use of the appropriate University student support services, and to contact the lecturer, at the earliest opportunity.

Finally, we are here to help! There is information about getting help in this subject on the LMS. Frequently asked questions about the project will be answered on Ed.

Requirements

The following implementation requirements must be adhered to:

- You must write your implementation in the C programming language.
- You must write your code in a modular way, so that your implementation could be used in another program without extensive rewriting or copying. This means that the Doubly Connected Edge List operations are kept together in a separate .c file, with its own header (.h) file, separate from the main program.
- Your implementation must read the input file once only.
- Your program should store strings in a space-efficient manner. If you are using malloc() to create the space for a string, remember to allow space for the final end of string '\0' (NULL).
- A full Makefile is not provided for you. The Makefile should direct the compilation of your program. To use the Makefile, make sure it is in the same directory as your code, and type `make voronoi1` to make the dictionary. You must submit your makefile with your assignment.
- Comments should be present in your code and aim to be useful for the target audience of your code, so should be in English, and can assume functional understanding of C and its library functions.

Hint: If make doesn't work, read the error messages carefully. A common problem in compiling multifile executables is in the included header files. Note also that the whitespace before the command is a tab, and not multiple spaces. It is not a good idea to code your program as a single file and then try to break it down into multiple files. Start by using multiple files, with minimal content, and make sure they are communicating with each other before starting more serious coding.

Hints

A number of hints you may find useful are collected here.

Starting Voronoi Diagrams

The first three steps in creating the Voronoi diagram are simple enough, but may need careful care:

1. For the first tower (or Voronoi site), can simply be stored in the face (and the reverse).
2. For the second tower (or Voronoi site), we have a special case, as the fact that there is only a single face doesn't alone determine whether we are inserting the first or second tower, so you'll need to check.
3. After inserting the first two towers, the third and beyond can simply check the number of faces.

Splits

If you used a mid-point vertex, you can use the split process from Assignment 1 to perform your splitting, simply set the position of these vertices to the location of the splits instead of the midpoint, the rest of the logic will work out.

Constructing New Face for Voronoi Cell

The face you construct following the algorithm must be connected together, a few parts of the process can significantly simplify this process.

- When using splits to construct the geometry, order the start and end edges in the split such that the watchtower is *outside the half-edge by the half-plane test*. This allows you to record the number of faces initially, perform all the splits, and then you know every face which will ultimately form the new face.
- Because of recommended choices during the previous assignment, each new face created will have a pointer to its half-edge in the DCEL, this means you can connect these directly.

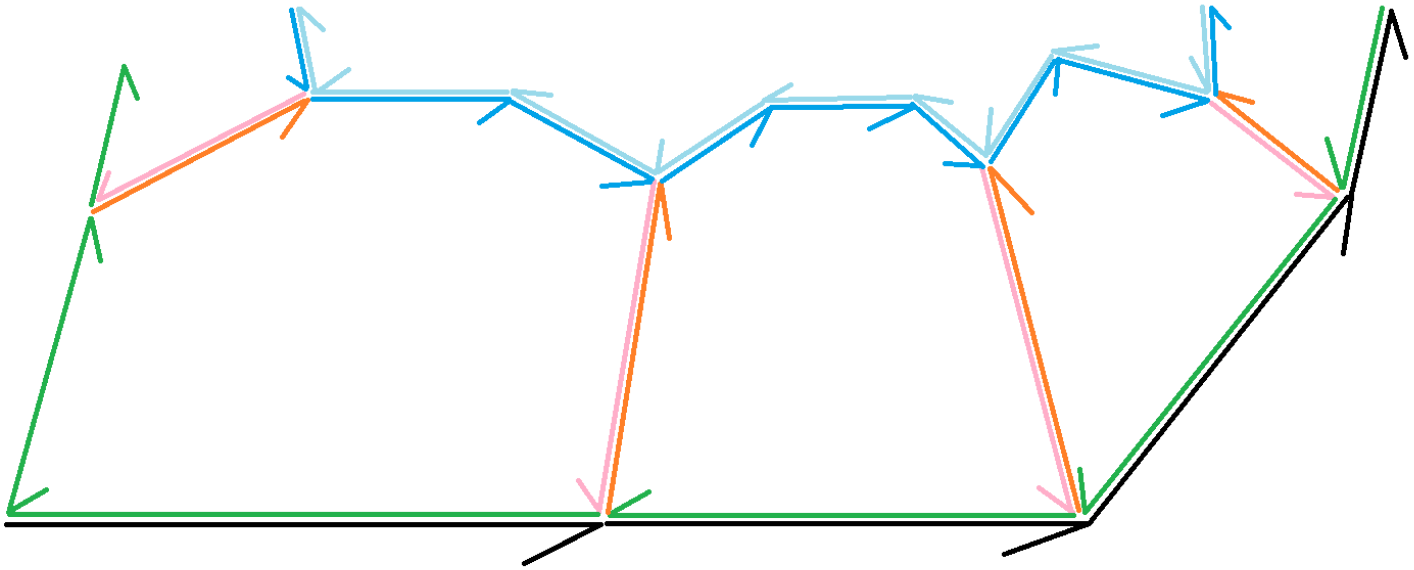
Cleaning Contained Geometry

After incremental Voronoi algorithm has completed, there will be edges in your DCEL which go unused, these don't do any harm, but may cause confusion if you want to visualise your progress. A simple process for cleaning up the contained geometry also allows us to connect all faces in the Voronoi cell.

- For each of the new faces, traverse all half-edges until you reach the original half-edge, update as you traverse with the following rules:
    - (Pink Half-Edge) If a half-edge's pair/twin/opposite half-edge is in a face which is one of

the new faces created, or has been assigned to be not in any face (e.g. NOFACE in the sample solution), and its following half-edge's pair/twin/opposite is NULL (one of the polygon initial half-edges, in green without pair), or joins to a face which is not one of the new faces (one of the joining half-edges, marked in green with black pair), then connect the *previous* pointer of the following edge (green) to the pair (orange)'s preceding half-edge (green in the other face). Set the face of the half-edge to not in any face.

- (Orange Half-Edge) If a half-edge's pair/twin/opposite half-edge is in a face which is one of the new faces created, or has been assigned to be not in any face (e.g. NOFACE in the sample solution), and its preceeding half-edge's pair/twin/opposite is NULL (one of the polygon initial half-edges, in green without pair), or joins to a face which is not one of the new faces (one of the joining half-edges, marked in green with black pair), then connect the *next* pointer of the preceding half-edge (green) to the pair (pink)'s following half-edge (green in the other face). Set the face of the half-edge to not in any face.
- (Blue and Light Blue Half-Edges) If the half-edge's pair/twin/opposite is in a face which is one of the new faces, set this half-edge's face to not in any face.
- (Green Half-Edges) Otherwise, set the half-edge's face to the stored new face value.



- After the traversal is complete, set the deleted faces to longer point to their half-edges.

Helper Functions

You may find functions to make traversing the DCEL easier useful, e.g. find next face, find next half-edge, etc.

Programming Style

Below is a style guide which assignments are evaluated against. For this subject, the 80 character limit is a guideline rather than a rule - if your code exceeds this limit, you should consider whether your code would be more readable if you instead rearranged it.

```c
/** ***********************
* C Programming Style for Engineering Computation
* Created by Aidan Nagorcka-Smith (aidann@student.unimelb.edu.au) 13/03
* Definitions and includes
* Definitions are in UPPER_CASE
* Includes go before definitions
* Space between includes, definitions and the main function.
* Use definitions for any constants in your program, do not just write
* in.
*
* Tabs may be set to 4-spaces or 8-spaces, depending on your editor. Th
* Below is ``gnu'' style. If your editor has ``bsd'' it will follow the
* style. Both are very standard.
*/
```

Some automatic evaluations of your code style may be performed where they are reliable. As determining whether these style-related issues are occurring sometimes involves non-trivial (and sometimes even undecidable) calculations, a simpler and more error-prone (but highly successful) solution is used. You may need to add a comment to identify these cases, so check any failing test outputs for instructions on how to resolve incorrectly flagged issues.

Additional Support

Your tutors will be available to help with your assignment during the scheduled workshop times. Questions related to the assignment may be posted on the Ed discussion forum, using the folder tag Assignments for new posts. You should feel free to answer other students' questions if you are confident of your skills.

A tutor will check the discussion forum regularly, and answer some questions, but be aware that for some questions you will just need to use your judgment and document your thinking. For example, a question like, "How much data should I use for the experiments?", will not be answered; you must try out different data and see what makes sense.

If you have questions about your code specifically which you feel would reveal too much of the assignment, feel free to post a private question on the discussion forum.

Submission

Your C code files (including your Makefile and any other files needed to run your code) should be submitted through Ed to this assignment. Your programs must compile and run correctly on Ed. You may have developed your program in another environment, but it still must run on Ed at submission time. For this reason, and because there are often small, but significant, differences between compilers, it is suggested that if you are working in a different environment, you upload and test your code on Ed at reasonably frequent intervals.

A common reason for programs not to compile is that a file has been inadvertently omitted from the submission. Please check your submission, and resubmit all files if necessary.

Assessment

There are a total of 15 marks given for this assignment.

Your C program will be marked on the basis of accuracy, readability, and good C programming structure, safety and style, including documentation (2 marks). Safety refers to checking whether opening a file returns something, whether mallocs do their job, etc. The documentation should explain all major design decisions, and should be formatted so that it does not interfere with reading the code. As much as possible, try to make your code self-documenting, by choosing descriptive variable names. The remainder of the marks will be based on the correct functioning of your submission.

Note that these correct functioning-related marks will be based on passing various tests. If your program passes these tests without addressing the learning outcomes (e.g. if you fully hard-code solutions or otherwise deliberately exploit the test cases), you may receive less marks than is suggested but your marks will otherwise be determined by test cases.

```
| Marks | Task                                                                                                                                  |
|-------|---------------------------------------------------------------------------------------------------------------------------------------|
| 4     | Compute and output equations for bisectors given pairs of points in a file.                                                           |
| 2     | Compute and output intersection points for bisectors against a given polygon.                                                         |
| 6     | Implement the incremental voronoi algorithm and output the diameter of each voronoi cell with its associated information in the original output order. |
| 2     | Sort the watchtowers by the diameter of those cells.                                                                                  |
| 1     | Program style consistent with Programming Style slide. Memory allocations and file opens checked.                                    |
```

Note that code style will be manually marked in order to provide you with the most meaningful feedback for the second assignment.

Dataset Download

This workspace slide does not have a description.

Assignment Submission

Upload your solution here!

Testing for the first 14 marks will be here. =)