Computational topology
Homework 1 (due: November 8th 2020)

Each problem is worth a certain amount of points. Some problems are theoretical, others require you also submit the code (that conforms to the requirements given in the problem description). You may choose which problems to solve, **15 points is equal to 100%**.

You have to submit your solutions before the deadline as **one .zip file** to the appropriate mailbox at https://ucilnica.fri.uni-lj.si/course/view.php?id=111 (near the top of the page).

This .zip file should contain:

1. **a namesurname.pdf file written in LaTeX** and containing the solutions to the theoretical problems you have chosen as well as solutions and explanations for the programming problems (also make sure you sign your name on the top of the first page),
2. **.py files** containing the code (one for each of the programming problems you have chosen).

1 Theoretical problems

1. **(3 points) Exploring different metrics.**

   On $\mathbb{R}^2$ we define the metrics $\alpha, \beta, \gamma \colon \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$,

$$\alpha((x_1, y_1), (x_2, y_2)) = \begin{cases} 0 & \text{if } (x_1, y_1) = (x_2, y_2), \\ \sqrt{x_1^2 + y_1^2} + \sqrt{x_2^2 + y_2^2} & \text{otherwise,} \end{cases}$$

$$\beta((x_1, y_1), (x_2, y_2)) = \begin{cases} \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} & \text{if } x_1 y_2 = x_2 y_1, \\ \sqrt{x_1^2 + y_1^2} + \sqrt{x_2^2 + y_2^2} & \text{otherwise,} \end{cases}$$

$$\gamma((x_1, y_1), (x_2, y_2)) = \begin{cases} |y_2 - y_1| & \text{if } x_1 = x_2, \\ |x_1 - x_2| + |y_1| + |y_2| & \text{if } x_1 \neq x_2. \end{cases}$$

   (a) Determine the distances between the points $(2, 1)$, $(4, 2)$ and $(0, 2)$ in all three metrics.

   (b) Draw the open balls $B((0, 0), 1)$, $B((1, 0), 2)$ and $B((0, 2), 6)$ in α metric. Careful, the centre of the ball is always contained in it! Why?

   (c) Draw the open balls $B((0, 0), 1)$, $B((1, 0), 2)$ and $B((2, 2), 3\sqrt{2})$ in β metric.

   (d) Draw the open balls $B((0, 0), 1)$, $B((1, 0), 2)$ and $B((2, 0), 3)$ in γ metric.

   This might help:

   α is called the *postman metric* because the distance from the postman $P(x_1, y_1)$ to the customer $C(x_2, y_2)$ is obtained by adding the (standard Euclidian) distance from the postman to the post office $O(0, 0)$ and the (standard Euclidian) distance from the post office to the customer.

   With the *radial metric* β, trips along the lines through origin are shorter than trips in other directions (if the two trips have the same length in the standard Euclidian metric).

   With the *river metric* (or the *French railroad metric*) simulates a sistem where you can move along the vertical lines directly, but you can only move in the horizontal direction along the x-axis. So, if you wish to move from one vertical line to another, you have to first travel from the starting point in the vertical direction until you reach the x-axis, then move along it and finally travel in the vertical direction to the second point.

2. **(2 points) Discrete metric.**

   The *discrete metric* on a space X is defined as $d\colon X \times X \to \mathbb{R}$,

   $$d(x,y) = \begin{cases} 0 & x = y, \\ 1 & \text{otherwise.} \end{cases}$$

   (a) Let $X = \mathbb{N}$. Describe $B(1, \frac{1}{2})$ and $B(2, 1)$.

   (b) Given three distinct integers a, b, c, when is the triangle with vertices at a, b and c equilateral?

3. **(2 points) Homeomorphic spaces.**

   Let $X = S^{n-1} \times [0,1] \subset \mathbb{R}^{n+1}$ and $Y = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \,;\, 1 \leq x_1^2 + \ldots + x_n^2 \leq 4\}$. Prove that X and Y are homeomorphic.

4. **(2 points) Homeomorphic spaces.**

   The upper hemisphere and the unit ball are defined as

   $$S^n_+ = \{(x_1, \ldots, x_{n+1}) \in S^n \,;\, x_{n+1} \geq 0\} \quad \text{and} \quad B^n = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \,;\, x_1^2 + \ldots + n_n^2 \leq 1\}.$$

   Show that they are homeomorphic.

2 Programming problems

5. **(3 points) Deciding connectivity**

   Let G be a simple graph with n vertices and m edges.

   Write a simple algorithm that returns the connected components of the graph. You can use either breadth-first search or a depth-first search to traverse the graph.

   Your file `graphcomponents.py` should contain a function `findComponents(V,E)` that returns a list `[C1,C2,...,Ck]` of all components. Each component `Ci` is a list of vertices `[v1,v2,...,vk]`.

   The input will consist of

   (a) a list `V = [1,2,..., n]` of n vertices and

   (b) a list of m 2-tuples `E = [(v1,v2),...]` that represent the m edges.

   You may want to pre-process the data into a more suitable data structure to speed up the algorithm.

   Sample inputs:

```
V = [1,2,3,4,5,6,7,8]
E = [(1,2),(2,3),(1,3),(4,5),(5,6),(5,7),(6,7),(7,8)]
```

```
V = [1,2,3,4,5]
E = [(1,2),(1,3),(1,4),(1,5)]
```

```
V = [1,2,3,4,5,6,7,8,9]
E = [(1,2),(1,3),(1,8),(3,7),(4,5),(4,6),(4,9),(5,6),(5,9),(7,8)]
```

   Corresponding outputs:

```
[[1, 2, 3], [4, 5, 6, 7, 8]]
```

```
[[1, 2, 3, 4, 5]]

???
```

Run these test cases to determine if your program works correctly (work out the final correct output on your own). Then put together two more test cases and include them in your report (inputs and outputs at the least, images would be nice).

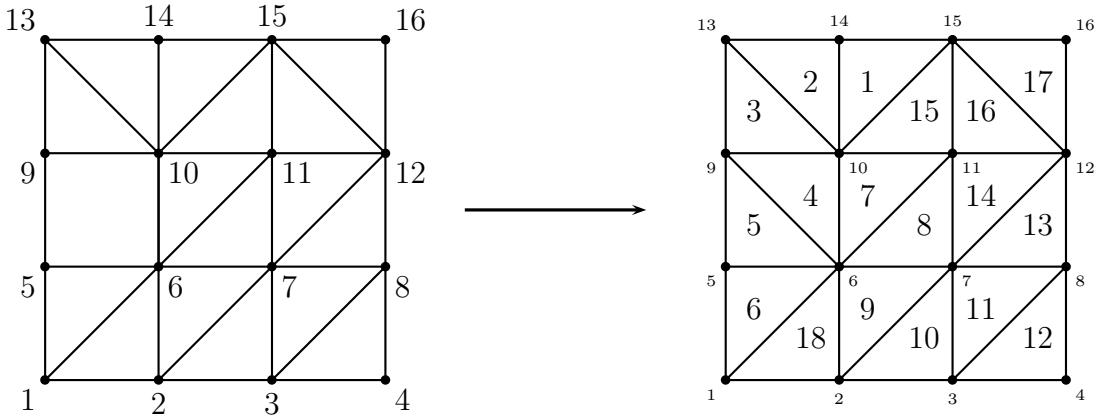
6. **(3 points) Shelling disks**

Let P be a simple closed polygon in the plane. A *polygon* is a plane figure that is bounded by a finite chain of straight line segments closing in a loop to form a closed polygonal chain. These segments are called its edges or sides, and the points where two edges meet are the polygon's vertices or corners. A polygon is simple if it does not have self-intersections.

We triangulate P, possibly adding vertices in the interior. The input will consist of a list `T` of triangles, for example:

```
T = [(1,2,6),(1,5,6),(2,3,7),(2,6,7),(3,4,8),(3,7,8),(5,6,9),(6,7,11),
     (6,9,10),(6,10,11),(7,8,12),(7,11,12),(9,10,13),(10,13,14),
     (10,11,15),(10,14,15),(11,12,15),(12,15,16)]
```

A *shelling* is a sequence of all triangles of P, such that any initial sequence is homeomorphic to a closed disk. Find an algorithm that produces this sequence.



Your file `shelling.py` should contain a function `shelling(T)` that outputs the list of triangles sorted in the correct order. Of course, a given polygon can have several different shelling sequences. You only need to find one. You should build the sequence incrementally. Think about the conditions that allow you to add a triangle to your sequence and the conditions that prevent you from doing so. Explain them in your report.

Run your problem on the example given above and then check that it produces the correct output. Then make up one more example of your own and do the same for that one. Include your example in the report (input, output and an image with the order of the triangles clearly marked).

7. **(3 points) Jordan curve theorem**

A simple closed curve in the plane is a connected curve with no self-intersections. We will consider a special case of a finite chain of straight line segments closing in a loop to form a simple closed polygonal curve.

The Jordan Curve Theorem states that every simple closed curve in $\mathbb{R}^2$ decomposes $\mathbb{R}^2$ into two components, the bounded inside and the unbounded outside.

Your file `jordan.py` should contain a function `insideQ(P,T)`, that returns `True` if the point T lies inside the polygonal curve P and `False` otherwise. To determine this, count the number of intersections of an infinite ray starting at T with the segments forming the curve P. Make sure you handle the cases where the ray is parallel to one of the segments or passes through a vertex.

The curve P is given as a list of coordinates of the vertices:

`P = [(x1,y1),(x2,y2),...,(xn,yn)]`.

Any two consecutive vertices are connected by a line segment and the first vertex is connected to the last to close the curve. The point T has coordinates `T = (x0,y0)`.

Sample input:

```
T = (2.33,0.66)

P = [(0.02,0.10),(0.98,0.05),(2.10,1.03),(3.11,-1.23),(4.34,-0.35),

     (4.56,2.21),(2.95,3.12),(2.90,0.03),(1.89,2.22)]
```

Test your algorithm on this example to ensure the output is `True`. Then come up with two more examples on your own. Include the inputs, the outputs and the images in your report, as well as an explanation of how your algorithm works. It should be clear from your explanation how you test if the segment and the ray are intersecting.