Exercise 1

Persistence pairing algorithm At each of the time steps 1, 2, 3, 4, 5 a new connected component is born. At time step 6 an edge between $\{a\}$ and $\{e\}$ is added and so one component ”dies”. The destructor 6 is paired with $\{e\}$ since it is the youngest unpaired creator between itself and $\{a\}$. Following the same reasoning, we then pair the destructor 7, 8, 9 with respectively the vertices $\{d\}, \{c\}, \{b\}$. Next in the filtration we add the edge between $\{a\}$ and $\{c\}$. Since between the two vertices is $\{c\}$ the youngest but is already paired, we search an other vertex to pair going backward as in slide 6 of Lecture 10. We then conclude that 10 must be a creator. Similarly we can also conclude that 11 is a creator. The next simplex filtration 12 can be paired with 11 and is so a destructor. Similarly to the above, 13 is a creator and 14 is a destructor. We therefore have the following persitences:

0-Persistence: $(e, 6), (d, 7), (c, 8), (b, 9), (a, \infty)$
1-Persistence: $(11, 12), (13, 14), (10, \infty)$.
Unpaired: $(a, \infty), (10, \infty)$
Creators: $a, b, c, d, e, 10, 11, 13$

Matrix reduction algorithm The algorithm has the following pseudocode:

Algorithm 1 Matrix reduction algorithm

```
for j = 1, ..., n do                          ▷ n is the number of columns of matrix
    while ∃ j_0 < j with low(j_0) = low(j) do
        Add column j_0 to column j
    end while
end for
```

$j = 7$

	a	b	c	d	e	ae	de	cd	bc	ac	ab	abc	ad	ade
a						1				1	1		1	
b									1		1			
c								1	1	1				
d							1	1					1	
e						1	1							
ae														1
de														1
cd														
bc												1		
ac												1		
ab												1		
abc														
ad														1
ade														

	a	b	c	d	e	ae	de	cd	bc	ac	ab	abc	ad	ade
a						1	1			1	1		1	
b									1		1			
c								1	1	1				
d							1	1					1	
e						1	0							
ae														1
de														1
cd														
bc												1		
ac												1		
ab												1		
abc														
ad														1
ade														

Following the procedure in Algorithm 1 for the remaining values of j yield the reduced matrix with the corresponding pivots.

	a	b	c	d	e	ae	de	cd	bc	ac	ab	abc	ad	ade
a						1	1	1	1	0	0		0	
b									1		0			
c								1	0	0				
d							1	0					0	
e						1	0							
ae														1
de														1
cd														
bc												1		
ac												1		
ab												1		
abc														
ad														1
ade														

We therefore obtain the same persistences as in the previous algorithm.

Exercise 2

This solution is taken from section 3.5.3 of [1]. First observe that we only need the 1-skeleton of K to compute $\mathrm{Dgm}_0 f$. So, in what follows, assume that K contains only vertices V and edges E. Assume that all vertices in V are sorted in non-decreasing order of their f-values. As before, let K_i be the union of lower-stars of all vertices v_j where $j \leq i$. Since we are only interested in the 0 -th homology, we only need to track the 0 -th homology group of K_i, which essentially embodies the information about connected components. Assume we are at vertex v_j. Consider $\mathrm{Lst}\,(v_j)$. There are three cases.

C-1 $\mathrm{Lst}\,(v_j) = \{v_j\}$. Then v_j starts a new connected component in K_j. Hence v_j is a creator.

C-2 All edges in $\mathrm{Lst}\,(v_j)$ connect to vertices from the same connected component C in K_{j-1}. In this case, the component C grows in the sense that it now includes also vertex v_j and its incident edges in the lower-star. However, $\mathrm{H}_0\,(K_{j-1})$ and $\mathrm{H}_0\,(K_j)$ are isomorphic where $K_{j-1} \subseteq K_j$ induces an isomorphism.

C-3 Edges in $\mathrm{Lst}\,(v_j)$ link to two or more components, say $C_1, \ldots, C_r$, in K_{j-1}. In this case, after the addition of $\mathrm{Lst}\,(v_j)$, all $C_1, \ldots, C_r$ are merged into a single component

$$C' = C_1 \cup C_2 \cup \cdots \cup C_r \cup \mathrm{Lst} v_j$$

Hence inclusion $K_{j-1} \hookrightarrow K_j$ induces a surjective homomorphism $\xi : \mathrm{H}_0\,(K_{j-1}) \to \mathrm{H}_0\,(K_j)$ and $\beta_0\,(K_j) = \beta_0\,(K_{j-1}) - (r-1)$. That is, we can consider that $r-1$ number of components are destroyed, only one stays on as C'.

Proposition 1. *Suppose Case-3 happens where edges in* $\mathrm{Lst}\,(v_j)$ *merges components* $C_1, \ldots, C_r$ *in* K_{j-1}. *Let* v_{k_i} *be the global minimum of component* C_i *for* $i \in [1, r]$. *Assume w.l.o.g that* $f\,(v_{k_1}) \leq f\,(v_{k_2}) \leq \cdots \leq f\,(v_{k_r})$. *Then the node* v_j *participates in exactly* $r-1$ *number of persistence pairings* $(v_{k_2}, v_j)\,, \ldots, (v_{k_r}, v_j)$ *for the 0 -dimensional persistent diagram* $\mathrm{Dgm}_0 f$, *corresponding to points* $(f\,(v_{k_2})\,, f\,(v_j))\,, \ldots, (f\,(v_{k_r})\,, f\,(v_j))$ *in* $\mathrm{Dgm}_0 f$.

Intuitively, when Case-3 happens, consider the set of 0 -cycles $c_2 = v_{k_2} + v_{k_1}, c_3 = v_{k_3} + v_{k_1}, \ldots, c_r = v_{k_r} + v_{k_1}$. On one hand, it is easy to see that their corresponding homology classes $[c_i\ ]$'s are distinct in $\mathrm{H}_0\left(K_{j-1}\right)$. Furthermore, each c_i is created upon entering K_{k_i} for each $i \in [1, r]$. On the other hand, the homology classes $[c_2], \ldots, [c_r]$ become trivial in $\mathrm{H}_0\left(K_j\right)$ (thus they are destroyed upon entering K_j). Hence $\mu_0^{k_i, j} > 0$ for $i \in [2, r]$, corresponding to persistence pairings $\left(v_{k_2}, v_j\right), \ldots, \left(v_{k_r}, v_j\right)$. Furthermore, consider any 0 -cycle $c_1 = v_{k_1} + c$ where c is a 0 -chain from $K_{k_1 - 1}$. The class $[c_1]$ is created at K_{k_1} yet remains non-trivial at K_j. Hence there is no persistence pairing (v_{k_1}, v_j).

Based on the above proposition, we only need to maintain connected components information for each K_i, and potentially merge multiple components. We would also need to be able to query the membership of a given vertex u in the components of the current sublevel set. Such operations can be implemented by using the union-find data structure.

Definition 1 (Union-find data structure)**.** *A union-find data structure is a standard data structure that maintains dynamic disjoint sets. Given a set of elements U called the universe, this data structure typically supports the following three operations to maintain a set $\mathcal{S}$ of disjoint subsets of U, where each subset also maintains a representative element: (1)* $\mathrm{MakeSet}(x)$ *which creates a new set* $\{x\}$ *and adds it to $\mathcal{S}$; (2)* $\mathrm{FindSEt}(x)$ *returns the representative of the set from $\mathcal{S}$ containing x; and (3)* $\mathrm{Union}(x, y)$ *merges the sets from $\mathcal{S}$ containing x and y respectively into a single one if they are different.*

We now present Algorithm ZeroPerDg. Here the universe U is the set of all vertices V of K. Note that each vertex v is also associated with its function value $f(v)$. In this algorithm, we assume that the representative of a set C is the minimum in it, i.e, the vertex with the smallest f-value, and the query $\mathrm{RepSet}(v)$ returns the representative of the set containing vertex v. We assume that this query takes the same time as $\mathrm{FindSET}(v)$. Given a disjoint set C, we also use $\mathrm{RePSET}(C)$ to represent the representative (minimum) of this set. One can view a disjoint set C in the collection $\mathcal{S}$ as the maximal set of elements sharing the same representative.

Algorithm 5 ZeroPerDg($K = (V, E)$, f)

Input:
K: a 1-complex with a vertex function f on it
Output:
Vertex pairs generating $\mathrm{Dgm}_0(f)$ for the PL function given by f

```
1:  Sort vertices in V so that f(v_1) ≤ f(v_2) ≤ ... ≤ f(v_n)
2:  for j = 1 → n do
3:      CreateSet(v_j)
4:      flag := 0
5:      for each (v_k, v_j) ∈ Lst(v_j) do
6:          if (flag == 0) then
7:              Union(v_k, v_j)
8:              flag := 1
9:          else
10:             if FindSet(v_k) ≠ FindSet(v_j) then
11:                 Set ℓ_1 = RepSet(v_k) and ℓ_2 = RepSet(v_j)
12:                 Union(v_k, v_j)
13:                 Output pairing (argmax{f(ℓ_1), f(ℓ_2)}, v_j)
14:             end if
15:         end if
16:     end for
17: end for
18: for each disjoint set C do
19:     Output pairing (RepSet(C), ∞)
20: end for
```

Let n and m denote the number of vertices and edges in K, respectively. Sorting all vertices in V takes $O(n \log n)$ time. There are $O(n+m)$ number of CreAteSet, FindSet, Union and RePSet operations. By using the standard union-find data structure, the total time for all these operations are $(n+m)\alpha(n)$ where $\alpha(n)$ is the inverse Ackermann function that grows extremely slowy with n. Hence the total time complexity of Algorithm ZeroPerDG is $O(n \log n + m\alpha(n))$.

Note that lines 18-20 of algorithm ZeroPerDg inspect all disjoint sets after processing all vertices and their lower-stars; each of such disjoint sets corresponds to a connected component in K. Hence each of them generates an essential pair in the 0 -th persistence diagram.

Theorem 1. *Given a PL-function $f : |K| \to \mathbb{R}$, the 0 -dimensional persistence diagram $\mathrm{Dgm}_0 f$ for the lower-star filtration of f can be computed by the algorithm ZeroPerDg in $O(n \log n + m\alpha(n))$ time, where n and m are the number of vertices and edges in K respectively.*

Connection to minimum spanning tree. If we view the 1-skeleton of K as a graph $G = (V, E)$, then ZeroPerDG (K, f) essentially computes the minimum spanning forest of G with the following edge weights: for every edge $e = (u, v)$, we set its weight $w(e) = \max\{f(u), f(v)\}$. Then, we can get the persistent pairs output of ZeroPerDg by running the well known Kruskal's algorithm on the weighted graph G. When we come across an edge $e = (u, v)$ that joins two disjoint components in this algorithm, we determine the two minimum vertices ℓ_1, ℓ_2 in these two components and pair e with the one among ℓ_1, ℓ_2 that has the larger f-value. After generating all such vertex-edge pairs (u, e), we convert them to vertex-vertex pairs (u, v) where $e \in \mathrm{Lst}(v)$. We throw away any pair of the form (u, u) because they signify local pairs.

Exercise 3

The algorithm ZeroPerDg can be easily adapted to compute persistence for a given filtration of a graph. In this case, we process the vertices and edges in their order in the filtration and maintain connected components using union-find data structure as in ZEROPERDG. For each edge $e = (u, v)$, we check if it connects two disconnected components represented by vertices ℓ_1 and ℓ_2 (line 11) and if so, e is paired with the younger vertex between ℓ_1 and ℓ_2 (line 13). We output all vertex-edge pairs thus computed. The vertices and edges that remain unpaired provide the infinite bars in the 0-th and 1-st persistence diagrams. The algorithm runs in $O(n\alpha(n))$ time if the graph has n vertices and edges in total. The $O(n \log n)$ term in the complexity is eliminated because sorting of the vertices is implicitly given by the input filtration.

References

[1] Tamal Krishna Dey and Yusu Wang. *Computational Topology for Data Analysis*. Cambridge University Press, 2022.