DSC 190 - Homework 04

Due: Wednesday, February 2

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope at 11:59 PM.

Problem 1.

Suppose you initialize an LSH data for storing points in $\mathbb{R}^2$ by choosing $\ell = 1$, $w = 1$, and $k = 2$ random unit vectors (shown below):

$$\vec{u}^{(1)} = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)^T$$
$$\vec{u}^{(2)} = \left(\frac{\sqrt{2}}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right)^T$$

You insert the following points into the LSH data structure one-by-one:

x	y
-1.5	1.5
2.5	-1.5
-0.5	0.5
1.5	-0.5
-1.5	-1.5
0.5	0.5

Upon querying the point (-2, -2), what other point(s) will be in the same cell? Do your calculations by hand and show your work (though you can use code to check your answer, if you wish).

Solution: The only other point in the same cell will be (-1.5, -1.5). To see that these two points are in the same cell, compute the cell id by projecting each point onto the two unit vectors. For instance, the projection of $(-1.5, -1.5)$ onto $\vec{u}^{(1)}$ is

$$(-1.5, 1.5) \cdot (1/\sqrt{2}, 1/\sqrt{2}) = -2.12$$

The floor of this is -3. Likewise, the projection of this point onto the second unit vector is -0.35, the floor of which is -1. Therefore the cell id is (-3, -1). The same cell id will be found for the query point.

Programming Problem 1.

In a file named `dsf.py`, create a class named `DisjointSetForest`. This class should have `.find_set`, `.make_set`, and `.union` methods as described in lecture, as well as the following methods and attributes:

- `.size_of_set(x)`: return the size of the set containing `x` as an integer. This should be done efficiently in $O(\alpha(n))$ amortized time.
- `.number_of_sets`: an integer attribute containing the current number of disjoint sets (not elements!) contained in the collection.

For methods that require elements to be specified, such as `.find_set`, `.union`, and `.size_of_set`, a `ValueError` should be raised if the element does not exist within the collection.

Starter code is available on the course webpage. Note that you may need to modify some of the existing methods from lecture, such as `.make_set`!

Solution:

```python
class DisjointSetForest:

    def __init__(self):
        # BEGIN PROMPT
        self._parent = []
        self._rank = []
        self._size_of_set = []
        self.number_of_sets = 0
        # END PROMPT

    def make_set(self):
        """Create a new singleton set.

        Returns
        -------
        int
            The id of the element that was just inserted.

        Example
        -------
        >>> dsf = DisjointSetForest()
        >>> dsf.make_set()
        0
        >>> dsf.make_set()
        1
        """
        # BEGIN PROMPT
        # get the new element's "id"
        x = len(self._parent)
        self._parent.append(None)
        self._rank.append(0)
        self._size_of_set.append(1)
        self.number_of_sets += 1
        return x
        # END PROMPT

    def find_set(self, x):
        """Find the representative of the element.

        Parameters
        ----------
        x : int
            The element to look for.

        Returns
        -------
        int
            The representative of the set containing `x`. The representative
            of a set with more than one element is arbitrary, but it should be
            the same of all elements of the set.

        Raises
        ----
        ValueError
            If `x` is not in the collection.

        Examples
        --------
        >>> dsf = DisjointSetForest()
        >>> dsf.make_set()
        0
        >>> dsf.find_set(0) == 0
        True
        >>> dsf.find_set(99)
        Traceback (most recent call last):
        ValueError: 99 is not in the collection.

        """
        try:
            parent = self._parent[x]
        except IndexError:
            raise ValueError(f'{x} is not in the collection.')

        if self._parent[x] is None:
            return x
        else:
            root = self.find_set(self._parent[x])
            self._parent[x] = root
            return root

    def union(self, x, y):
        """Union the sets containing x and y, in-place.

        Parameters
        ----------
        x, y : int
            The elements whose sets should be unioned. They do not need
            to belong to different sets.

        Raises
        ------
        ValueError
            If x or y are not in the collection.

        Example
        -------
        >>> dsf = DisjointSetForest()
        >>> dsf.make_set()
        0
        >>> dsf.make_set()
        1
        >>> dsf.number_of_sets
        2
        >>> dsf.union(0, 1)
        >>> dsf.number_of_sets
        1
        >>> dsf.find_set(0) == dsf.find_set(1)
        True

        """
        # BEGIN PROMPT
        x_rep = self.find_set(x)
        y_rep = self.find_set(y)

        if x_rep == y_rep:
            return

        if self._rank[x_rep] > self._rank[y_rep]:
            self._parent[y_rep] = x_rep
            self._size_of_set[x_rep] += self._size_of_set[y_rep]
        else:
            self._parent[x_rep] = y_rep
            self._size_of_set[y_rep] += self._size_of_set[x_rep]
            if self._rank[x_rep] == self._rank[y_rep]:
                self._rank[y_rep] += 1

        self.number_of_sets -= 1
        # END PROMPT

    def size_of_set(self, x):
        """The size of the set containing x.

        Parameters
        ----------
        x : int
            The element whose set will be sized.

        Returns
        -------
        int
            The size of the set containing x.

        Raises
        ------
        ValueError
            If x is not in the collection.

        Example
        -------
        >>> dsf = DisjointSetForest()
        >>> dsf.make_set()
        0
        >>> dsf.make_set()
        1
        >>> dsf.union(0, 1)
        >>> dsf.size_of_set(0)
        2

        """
        # BEGIN PROMPT
        return self._size_of_set[self.find_set(x)]
        # END PROMPT
```

Problem 2.

Consider the following model for iteratively generating a random connected graph with n nodes. Start with a graph containing n nodes and no edges. Next, randomly select an edge from the uniform distribution on the set of all possible edges and add it to the graph (if it isn't already in the graph). Repeat this process until the graph is connected[1].

How many edges will the resulting graph have? It is a random number, of course, but it must be at least $n - 1$. Let's visualize the distribution of this quantity through an empirical experiment, *a la* DSC 10.

Write a function to simulate the above procedure and return the number of edges in the resulting graph, assuming $n = 100$. Run your function 10,000 times and plot a histogram of the results. Include your figure, as well as your code. Your code should be fast enough so that all 10,000 trials take less than 1 minute or so in total.

Hint: you can use DFS/BFS to determine whether the graph is connected, but if you do your code will take forever to run... Is there a better way of keeping track of connected components?

Solution: We'll use a disjoint set forest to keep track of the connected components. If we tried to use a graph data structure and DFS/BFS, it would take forever to run. Here's some code for running the experiment once:

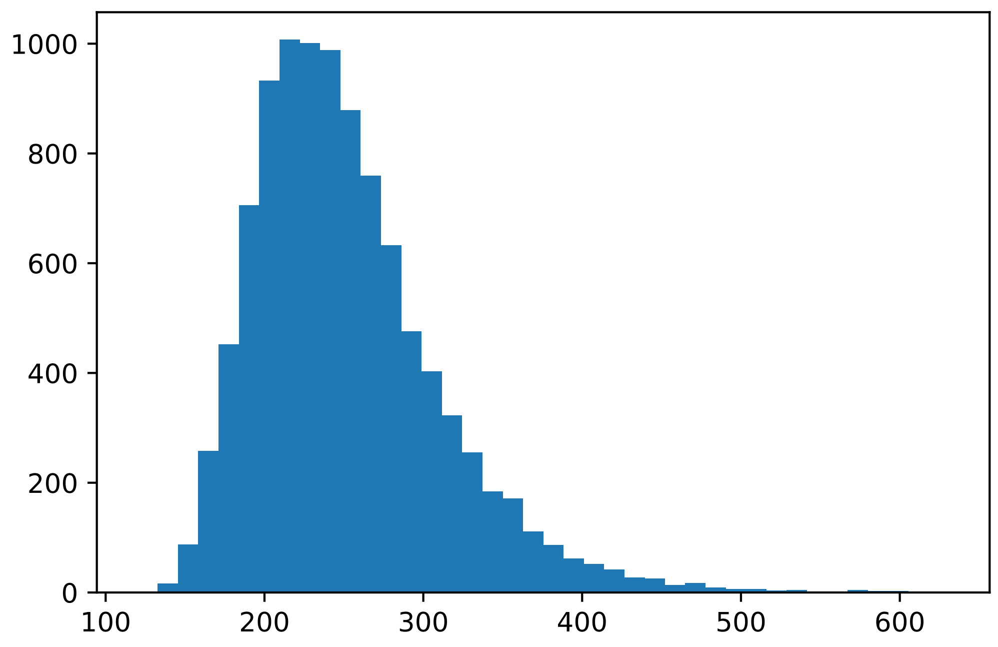
```
def experiment(n):
    dsf = DisjointSetForest()
    for i in range(n): dsf.make_set()

    count = 0
    seen = np.zeros((n, n))
    while dsf.number_of_sets > 1:
        u = random.choice(range(n))
        v = random.choice(range(n))
        if seen[u, v] or u == v:
            continue
        seen[u, v] = seen[v, u] = 1
        dsf.union(u, v)
        count += 1

    return count
```

When we run this 10,000 times and plot a histogram, we see the following:

[1]This random graph model is related to (but not the same as) the famous Erdös-Renyi random graph model. While simple, the E-R model has been the subject of a lot of theoretical analysis.



It looks like this might be a log-normal distribution...