DSC 190 - Discussion 05

Problem 1.

Modify the `DisjointSetForest` data structure so that it keeps track of the maximum and minimum keys within each disjoint set without adding to the time complexity of any of the operations.

Solution:

```
class DisjointSetForest:

    def __init__(self):
        self._parent = []
        self._rank = []
        self._min = []
        self._max = []

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

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

        """

        #{0} - min = 0, max = 0
        #{1} - min = 1, max = 1

        # get the new element's "id"
        #Creates {x}
        x = len(self._parent)
        self._parent.append(None)
        self._min.append(x)
        self._max.append(x)
        self._rank.append(0)
        return x

    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.
```

```
        Raises
        ----
        ValueError
            If x 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.

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

        """
        x_rep = self.find_set(x)
        y_rep = self.find_set(y)

        if x_rep == y_rep:
            return
        # y_rep's parent to x_rep
        # x_rep is now the repr element of the disjoint set
        # x_rep -> {0,3,4} y_rep -> {2,5,6}
        if self._rank[x_rep] > self._rank[y_rep]:
            self._parent[y_rep] = x_rep
            self._min[x_rep] = min(self._min[x_rep], self._min[y_rep])
            self._max[x_rep] = max(self._max[x_rep], self._max[y_rep])
        else:
            self._parent[x_rep] = y_rep
            self._min[y_rep] = min(self._min[x_rep], self._min[y_rep])
            self._max[y_rep] = max(self._max[x_rep], self._max[y_rep])
            if self._rank[x_rep] == self._rank[y_rep]:
                self._rank[y_rep] += 1

    def min_of_set(self, x):
```

```
        return self._min[self.find_set(x)]

    def max_of_set(self, x):
        return self._max[self.find_set(x)]
```

Problem 2.

The *fractional* knapsack problem is as follows. You have a bag that can hold B liters. In front of you are n piles of gold dust, silver dust, etc. The ith pile contains s_i liters of dust, and the dust in the pile is worth w_i dollars in total. You may choose any amount of dust from any pile to put in your bag (up to s_i). Your goal is to maximize the value of the dust in your bag.

Describe a greedy algorithm for solving this problem. Is it guaranteed to find the optimal answer?

Solution: Start by calculating the *price per liter* for each type of dust. Then, starting with the most expensive dust in dollars per liter, place as much as you can into your bag until 1) your bag is full or 2) the pile is gone. Then move to the next most valuable pile by price per liter and repeat.

This is optimal.