Assignment 4:

Installation

1. To run the file, on a command prompt type ‘python disjoint.py’

Explanation

1. In computer science, a **disjoint-set data structure**, also called a **union–find data structure** or **merge–find set**, is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. It provides near-constant-time operations (bounded by the inverse Ackermann function) to add new sets, to merge existing sets, and to determine whether elements are in the same set
2. A disjoint-set forest consists of a number of elements each of which stores an id, a parent pointer, and, in efficient algorithms, a value called the "rank".
3. The parent pointers of elements are arranged to form one or more trees, each representing a set. If an element's parent pointer points to no other element, then the element is the root of a tree and is the representative member of its set. A set may consist of only a single element. However, if the element has a parent, the element is part of whatever set is identified by following the chain of parents upwards until a representative element (one without a parent) is reached at the root of the tree.
4. Forests can be represented compactly in memory as arrays in which parents are indicated by their array index.
5. **Make-Set Operation:** The MakeSet operation makes a new set by creating a new element with a unique id, a rank of 0, and a parent pointer to itself. The parent pointer to itself indicates that the element is the representative member of its own set. The MakeSet operation has O(1) time complexity
6. **Find:** Find(x) follows the chain of parent pointers from x upwards through the tree until an element is reached whose parent is itself. This element is the root of the tree and is the representative member of the set to which x belongs, and may be x itself.
7. **Union:** Union(x,y) uses Find to determine the roots of the trees x and y belong to. If the roots are distinct, the trees are combined by attaching the root of one to the root of the other. If this is done naively, such as by always making x a child of y, the height of the trees can grow as O(n). To prevent this union by rank is used.
8. **Path Compression:** The idea is to flatten the tree when find() is called. When find() is called for an element x, root of the tree is returned. The find() operation traverses up from x to find root. The idea of path compression is to make the found root as parent of x so that we don’t have to traverse all intermediate nodes again. If x is root of a subtree, then path (to root) from all nodes under x also compresses.
9. Use Tree representation of disjoint set ADT to do the following operations:
    a. for i=1 to i=8
        Make_set(i)

```
# Create Disjoint set in a for loop
disjoint_list = [i for i in range(1, 9)]
disjoint_set = DisjointSet(disjoint_list)
```

b. union(5,6)
union(7,8)
union(5,7)

```
disjoint_set.union(5, 6)
disjoint_set.union(7, 8)
disjoint_set.union(5, 7)
```

10. Give a non-recursive implementation of FIND-SET with path compression.

```
def find(self, elem):
    for item in self._disjoint_set:
        if elem in item:
            return
self._disjoint_set[self._disjoint_set.index(item)]
    return None
```

The output of the program is as follows:

```
The disjoint set is[[1], [2], [3], [4], [5], [6], [7], [8]]

The disjoint set after union(5,6) is [[1], [2], [3], [4], [6, 5], [7], [8]]

The disjoint set after union(7,8) is [[1], [2], [3], [4], [6, 5], [8, 7]]

The disjoint set after union(5,7) is [[1], [2], [3], [4], [8, 7, 6, 5]]

Finding element non-recursively with path compressions...

[8, 7, 6, 5]
```

A diagrammatic representation of the disjoint set is as follows: