|  |  |
| --- | --- |
|  | **Manav Rachna University** |
| **Lab Assignment 04** |
| **Subject:** Analysis and Design of Algorithms **Subject Code**: **CSH326**  **Semester: IV** | |

**Learning Objective:** Students would be Able to learn the Disjoint sets.

**Learning Outcome:** To understand the applications of disjoint sets and their operations in solving various problems.

1. Program to implement the core Disjoint Set operations with union by rank for efficiency.

Code:

class DisjointSet:

    def \_\_init\_\_(self, n):

        self.parent = [i for i in range(n)]

        self.rank = [0] \* n

    def find(self, x):

        if self.parent[x] != x:

            self.parent[x] = self.find(self.parent[x])

        return self.parent[x]

    def union(self, x, y):

        root\_x = self.find(x)

        root\_y = self.find(y)

        if root\_x != root\_y:

            if self.rank[root\_x] > self.rank[root\_y]:

                self.parent[root\_y] = root\_x

            elif self.rank[root\_x] < self.rank[root\_y]:

                self.parent[root\_x] = root\_y

            else:

                self.parent[root\_y] = root\_x

                self.rank[root\_x] += 1

    def connected(self, x, y):

        return self.find(x) == self.find(y)

ds = DisjointSet(5)

ds.union(0, 1)

ds.union(1, 2)

print(ds.connected(0, 2))

print(ds.connected(0, 3))

Output:



2. Program uses Disjoint Sets to efficiently determine if a given graph contains a cycle.

Code:

class DisjointSet:

    def \_\_init\_\_(self, n):

        self.parent = [i for i in range(n)]

        self.rank = [0] \* n

    def find(self, x):

        if self.parent[x] != x:

            self.parent[x] = self.find(self.parent[x])

        return self.parent[x]

    def union(self, x, y):

        root\_x = self.find(x)

        root\_y = self.find(y)

        if root\_x == root\_y:

            return False

        if self.rank[root\_x] > self.rank[root\_y]:

            self.parent[root\_y] = root\_x

        elif self.rank[root\_x] < self.rank[root\_y]:

            self.parent[root\_x] = root\_y

        else:

            self.parent[root\_y] = root\_x

            self.rank[root\_x] += 1

        return True

def contains\_cycle(edges, n):

    ds = DisjointSet(n)

    for u, v in edges:

        if not ds.union(u, v):

            return True

    return False

graph\_edges = [(0, 1), (1, 2), (2, 3), (3, 0)]

graph\_nodes = 4

print(contains\_cycle(graph\_edges, graph\_nodes))

Output:

