7/10/20 ADA Week 4

Adith Ravi Pai
USN: 1BM18CS005

Given a boolean 2D Matrix, find the no. of islands.
A group of 1's forms an island. Ex:-
{1, 1, 0, 0, 0}, {0, 1, 0, 0, 1}, {1, 0, 0, 1, 1}, {0, 0, 0, 0, 0},
A cell in the 2D matrix can be connected to {1, 0, 1, 0, 1}
8 neighbours. Use disjoint sets to implement above scenario.

```
import java.io.*;
import java.util.*;
public class Main
{
    public static void main(String[] args) throws IOException
    { int[][] a = new int[][] {{1, 1, 0, 0, 0},
                               {0, 1, 0, 0, 1},
                               {1, 0, 0, 1, 1},
                               {0, 0, 0, 0, 0},
                               {1, 0, 1, 0, 1}};

    System.out.println("No. of islands is: " +
                                countIslands(a));
}
static int countIslands(int a[][])
{
    int n = a.length;
    int m = a[0].length;
    DisjointUnionSets dus = new DisjointUnionSets(n*m);
        //Check for neighbour and write indexes if both are 1
    for(int j=0; j<n; j++)
    { for(int k=0; k<m; k++) {
```

Adith Ravi Pai
1BM18CS005

```
// if cell is 0, dont do anything
if (a[j][k] == 0)
    continue;

// Check all 8 neighbours and do union with neighbour's
// set if neighbour is also 1
if (j+1 < n && a[j+1][k] == 1)
    dus.union(j*(m)+k, (j+1)*(m)+k);
if (j-1 >= 0 && a[j-1][k] == 1)
    dus.union(j*(m)+k, (j-1)*(m)+k);
if (k+1 < m && a[j][k+1] == 1)
    dus.union(j*(m)+k, (j)*(m)+k+1);
if (k-1 >= 0 && a[j][k-1] == 1)
    dus.union(j*(m)+k, (j)*(m)+k-1);
if (j+1 < n && k+1 < m && a[j+1][k+1] == 1)
    dus.union(j*(m)+k, (j+1)*(m)+k+1);
if (j+1 < n && k-1 >= 0 && a[j+1][k-1] == 1)
    dus.union(j*m+k, (j+1)*(m)+k-1);
if (j-1 >= 0 && k+1 < m && a[j-1][k+1] == 1)
    dus.union(j*m+k, (j-1)*m+k+1);
if (j-1 >= 0 && k-1 >= 0 && a[j-1][k-1] == 1)
    dus.union(j*m+k, (j-1)*m+k-1);
}
}
```

```
//Array to note down frequency of each set
int [] c = new int [n*m];
int numberOfIslands=0;
for(int j=0;j<n;j++)
{
  for(int k=0;k<m;k++)
  {
    if(a[j][k]==1)
    {
      int x = dus.find(j*m+k);
        // if frequency of set is 0, increment no. of islands
      if(c[x]==0)
      {
        numberOfIslands++;
          c[x]++;
      }
      else c[x]++;
    }
  }
}
  return numberOfIslands;
  }
}
class DisjointUnionSets
{ int[] rank,parent;
  int n;
  public DisjointUnionSets(int n)
  { rank = new int[n];
    parent = new int[n];
    this.n = n;
    makeSet();
  }
```

```
void makeset()
{ // initially, all elements are in their own set.
   for(int i=0; i<n; i++)
      parent[i] = i;
}

// finds representative of the set that x is an element of
int find(int x)
{
   if (parent[x] != x)
      return find(parent[x]);
   return x;
}
void union(int x, int y)
{
   int xRoot = find(x);
   int yRoot = find(y);
   if (xRoot == yRoot) return;
   if (rank[xRoooot] < rank[yRoot])
      parent[xRoot] = yRoot;
   else if (rank[yRoot] < rank[xRoot])
      parent[yRoot] = xRoot;
   else { parent[yRoot] = xRoot;
          rank[xRoot] = rank[xRoot] + 1;
      }
   }
}
```

Adith Pai
IBM18CS005