Solving the Percolation problem

This method utilises weighted quick-find (by size) with path compression. To track the root of the point represented by x and y, the value of array at (x , y) is set to a string "x' y'" where (x' , y') is the root of (x , y). Also all vartices of first row are joined together and so are all vertices of last row to find if the system percolates faster.

Percolation.java :

```
public class Percolation {
    int [][] arr;
    String [][] x;
    int[][] size;
    int openSites;
    int n;

    public void union(int i,int j,int ii,int jj){
        String a=root(i,j);
        String b=root(ii,jj);
        int a0= Integer.parseInt(a.split(" ")[0]);
        int a1= Integer.parseInt(a.split(" ")[1]);
        int b0= Integer.parseInt(b.split(" ")[0]);
        int b1= Integer.parseInt(b.split(" ")[1]);
        if(a!=b){
            if(size[a0][a1]>size[b0][b1]){
                x[b0][b1]=a;
                size[a0][a1]+=size[b0][b1];
            }
            else{
                x[a0][a1]=b;
                size[b0][b1]+=size[a0][a1];
            }
        }
    }

    public String root(int i,int j){
        String r="";
        int r0;
        int r1;
        while(!x[i][j].equals(""+i+" "+j)){
            r=x[i][j];
            r0=Integer.parseInt(r.split(" ")[0]);
            r1=Integer.parseInt(r.split(" ")[1]);
            x[i][j]=x[r0][r1];
            i=Integer.parseInt(r.split(" ")[0]);
            j=Integer.parseInt(r.split(" ")[1]);
```

```
    }
    return x[i][j];
}

public Percolation(int n){
    this.n=n;
    this.arr=new int[n][n];
    this.x=new String[n][n];
    this.openSites=0;
    for(int i=0;i<n;i++){
        if(i==0){for(int j=0;j<n;j++) x[i][j]= ""+0+" "+0;}
        else if(i==n-1){for(int j=0;j<n;j++) x[i][j]= ""+(n-1)+" "+(n-1);}
        else{
            for(int j=0;j<n;j++) x[i][j]= ""+i+" "+j;
        }
    }
    this.size=new int[n][n];
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++) size[i][j]=1;
    }
}

public void open(int i,int j){
    if(arr[i][j]==0){
        arr[i][j] = 1;
        if(i==0) union(i,j,0,0);
        if(i==n-1) union(i,j,n-1,n-1);
        if(j-1>=0 && isOpen(i, j-1)) union(i,j,i,j-1);
        if(j+1<=n-1 && isOpen(i, j+1)) union(i,j,i,j+1);
        if(i-1>=0 && isOpen(i-1, j)) union(i,j,i-1,j);
        if(i+1<=n-1 && isOpen(i+1, j)) union(i,j,i+1,j);
        openSites+=1;
    }

}

public boolean isOpen(int i, int j){
    return arr[i][j]==1;
}

public boolean isFull(int i, int j){
    return root(i,j)==root(0,0);
}

public int numberOfOpenSites(){
    return openSites;
```

```
    }

    public boolean percolates(){
        return root(0,0)==root(n-1,n-1);
    }
}
```

The Percolation stats calculate the mean, standard deviation and 95% confidence interval over n x n grid and T simulations which it recieves via arguments.

PercolationStats.java :

```
public class PercoltionStats {
    public static void calculate(double[] array) {

        // get the sum of array
        double sum = 0.0;
        for (double i : array) {
            sum += i;
        }

        // get the mean of array
        int length = array.length;
        double mean = sum / length;
        System.out.println("Mean =\t"+mean);
        // calculate the standard deviation
        double standardDeviation = 0.0;
        for (double num : array) {
            standardDeviation += Math.pow(num - mean, 2);
        }
        standardDeviation=Math.sqrt(standardDeviation / length);
        System.out.println("Std =\t"+standardDeviation);

        double confidenceLevel = 1.96;
        double temp = confidenceLevel * standardDeviation / Math.sqrt(length);
        System.out.println("95% confidence interval =\t["+(mean - temp)+","+ (mean + temp)+'
    }
    public static void main(String[] args) {
        int q=Integer.parseInt(args[0]);
        int T=Integer.parseInt(args[1]);
        double[] data=new double[T];
        for(int i=0;i<T;i++){
            Percolation p=new Percolation(q);
            int f,g;
            while(!p.percolates()){
                f=(int)(Math.random()*(q));
                g=(int)(Math.random()*(q));
```

```
                p.open(f,g);
            }
            data[i]=(double)p.openSites/(q*q);
        }

        calculate(data);
    }
}
```