Experiment-5: Agglomerative

Aim: Write a java program to implement agglomerative clustering

Source Code:

```
import  java.util.*;
import java.lang.Math.*;
class aglC {
	int numOfPoints, choice;
	String method;
	String names[] = new String[10];
	List<ArrayList<Integer>> clusters = new ArrayList<ArrayList<Integer>>();
	double distances[][] = new double[10][10];
	double tempDistances[][] = new double[10][10];
	int min[] = new int[] { 0, 1, 0 };
	Scanner sc = new Scanner(System.in);
	void input() {
		System.out.print("Enter number of points: ");
		numOfPoints = sc.nextInt();
		System.out.print("Enter points: ");
		for (int i = 0; i < numOfPoints; i++) {
			names[i] = sc.next();
			ArrayList<Integer> list = new ArrayList<Integer>();
			list.add(i);
			clusters.add(list);
		}
		System.out.println("Enter distance matrix");
		for (int i = 0; i < numOfPoints; i++) {
			for (int j = 0; j < numOfPoints; j++) {
				distances[i][j] = sc.nextDouble();
```

```
                }
            }
            System.out.println("\n1. Single Linkage\t2. Complete Linkage\t3. Average
Linkage");
            System.out.print("Choose Method: ");
            this.choice = sc.nextInt();
            switch (this.choice) {
            case 1:
                this.method = "Dendogram - Single Linkage";
                break;
            case 2:
                this.method = "Dendogram - Complete Linkage";
                break;
            case 3:
                this.method = "Dendogram - Average Linkage";
                break;
            default:
                break;
            }
        }
        // Single Linkage
        void calcDistMin() {
            double min, tempMin;
            for (int clust1 = 0; clust1 < clusters.size(); clust1++) {
                for (int clust2 = clust1 + 1; clust2 < clusters.size(); clust2++) {
                    min =
distances[clusters.get(clust1).get(0)][clusters.get(clust2).get(0)];
                    for (Integer point1 : clusters.get(clust1)) {
                        for (Integer point2 : clusters.get(clust2)) {
                            tempMin = distances[point1][point2];
                            if (tempMin < min) {
                                min = tempMin;
                            }
                        }
```

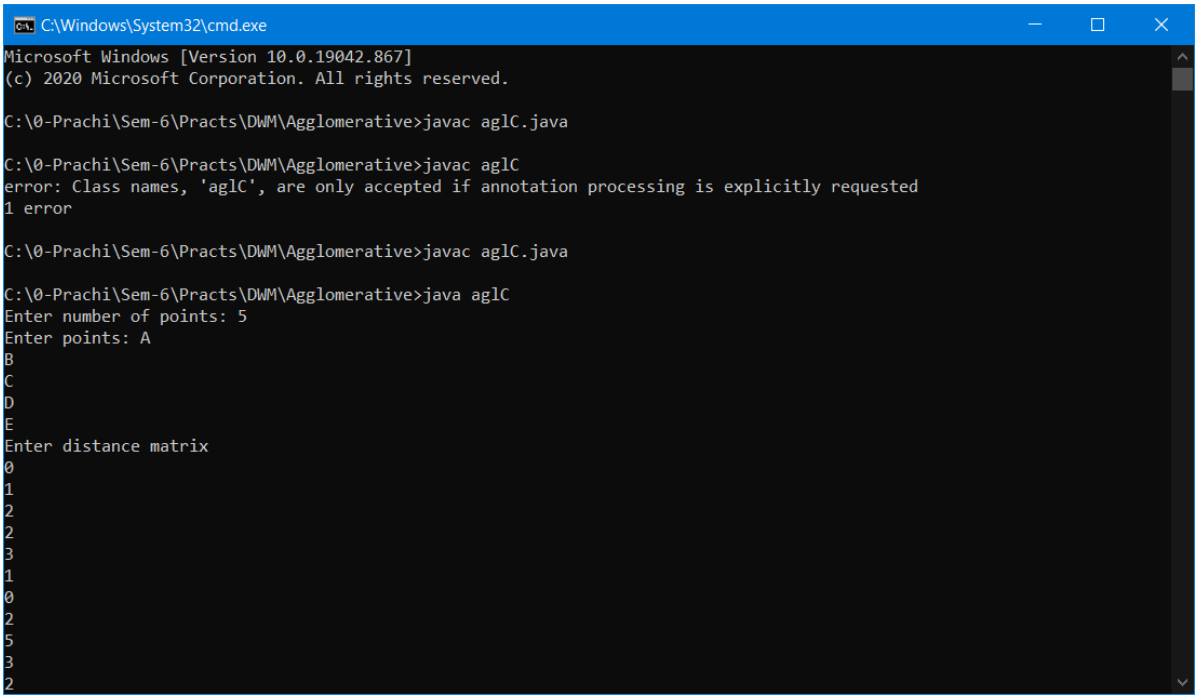
```
                        }
                        tempDistances[clust1][clust2] = min;
                        tempDistances[clust2][clust1] = min;
                }
        }
        printMatrix();
}
void calcDistMax() {
        double max, tempMax;
        for (int clust1 = 0; clust1 < clusters.size(); clust1++) {
                for (int clust2 = clust1 + 1; clust2 < clusters.size(); clust2++) {
                        max = 0;
                        for (Integer point1 : clusters.get(clust1)) {
                                for (Integer point2 : clusters.get(clust2)) {
                                        tempMax = distances[point1][point2];
                                        if (max < tempMax) {
                                                max = tempMax;
                                        }
                                }
                        }
                        tempDistances[clust1][clust2] = max;
                        tempDistances[clust2][clust1] = max;
                }
        }
        printMatrix();
}
void calcDistAvg() {
        double avg, count;
        for (int clust1 = 0; clust1 < clusters.size(); clust1++) {
                for (int clust2 = clust1 + 1; clust2 < clusters.size(); clust2++) {
                        avg = count = 0;
                        for (Integer point1 : clusters.get(clust1)) {
                                for (Integer point2 : clusters.get(clust2)) {
                                        avg += distances[point1][point2];
```

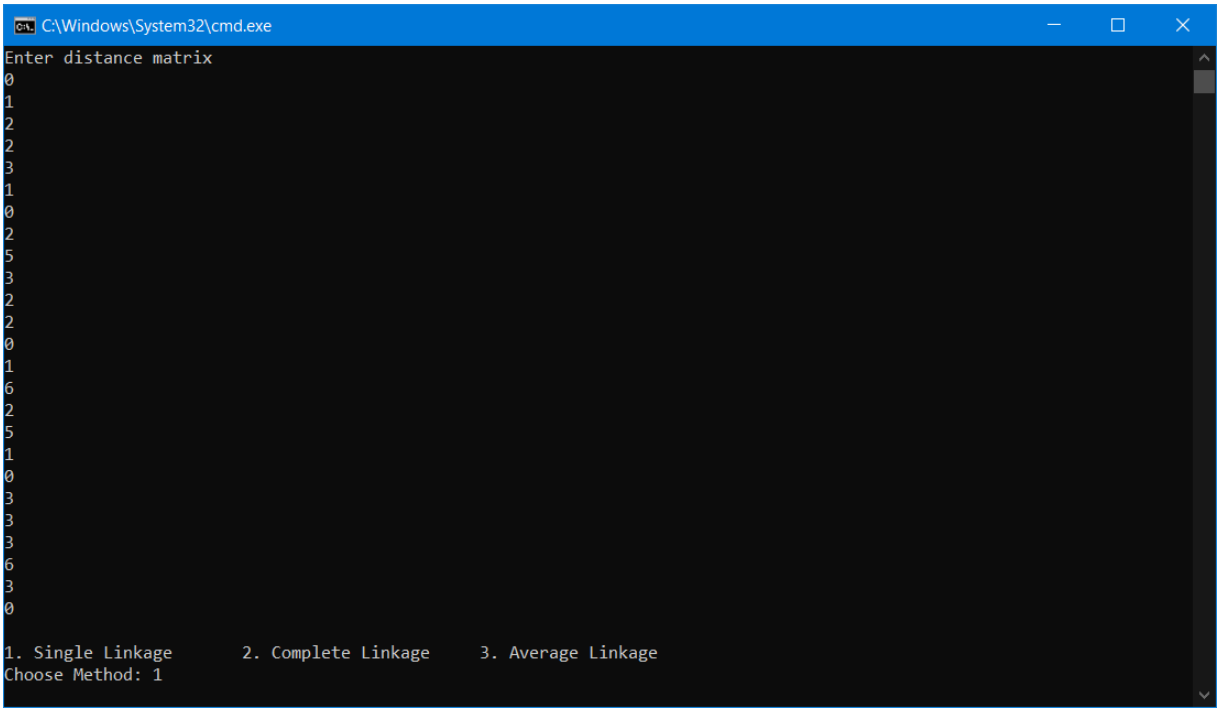
```
                                        count++;
                                }
                        }
                        avg /= count;
                        tempDistances[clust1][clust2] = avg;
                        tempDistances[clust2][clust1] = avg;
                }
        }
        printMatrix();
}
void printMatrix() {
        String toPrint = "";
        System.out.printf("%14s ", "");
        for (ArrayList<Integer> cluster : clusters) {
                toPrint = names[cluster.get(0)];
                for (int i = 1; i < cluster.size(); i++) {
                        toPrint += "-" + names[cluster.get(i)];
                }
                System.out.printf("%-14s ", toPrint);
        }
        for (int i = 0; i < clusters.size(); i++) {
                System.out.println();
                toPrint = names[clusters.get(i).get(0)];
                for (int j = 1; j < clusters.get(i).size(); j++) {
                        toPrint += "-" + names[clusters.get(i).get(j)];
                }
                System.out.printf("%-14s ", toPrint);
                for (int j = 0; j < clusters.size(); j++) {
                        System.out.printf("%-14.3f ", tempDistances[i][j]);
                }
        }
}
int calcMatrixMin() {
        double tempMin, min = tempDistances[0][1];
```

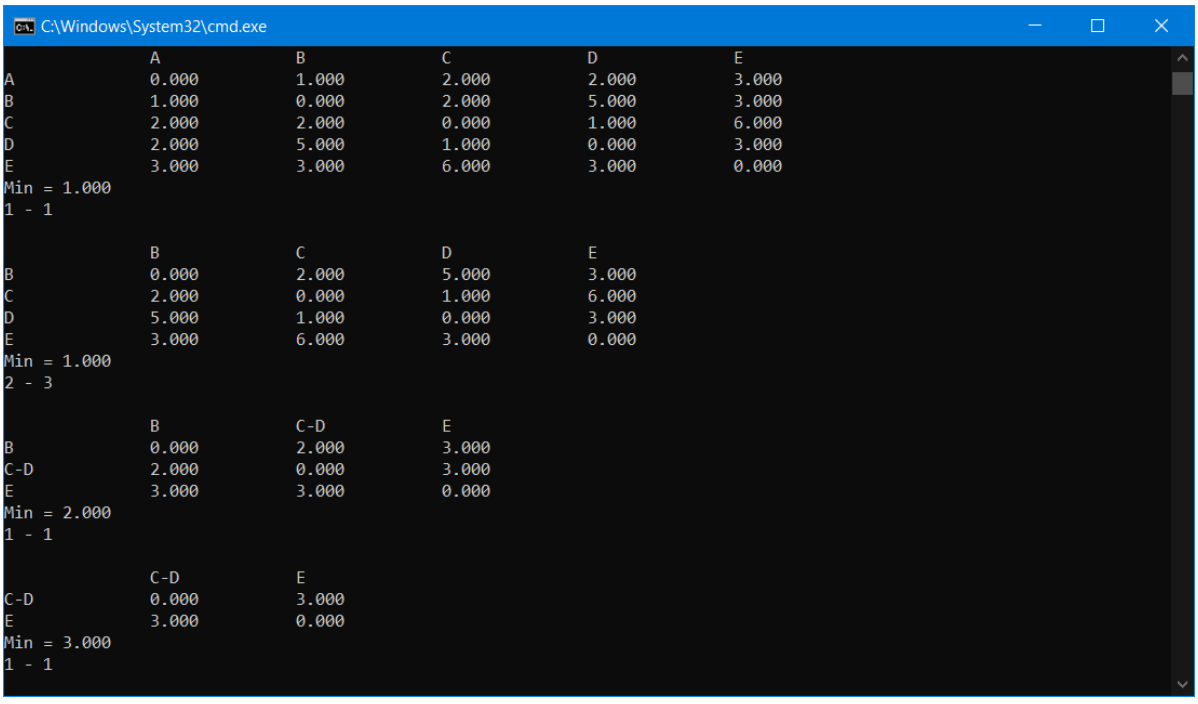
```
        int clust1 = 0, clust2 = 0;
        for (int i = 0; i < clusters.size(); i++) {
                for (int j = i + 1; j < clusters.size(); j++) {
                        tempMin =  tempDistances[i][j];
                        if (tempMin < min) {
                                min = tempMin;
                                clust1 =  i;
                                clust2 = j;
                        }
                }
        }
        System.out.printf("\nMin = %.3f\n", min);
        System.out.println((clust1 + 1) + " - " + (clust2 + 1));
        clusters.get(clust1).addAll(clusters.get(clust2));
        clusters.remove(clust2);
        return 0;

}
void solve() {
        while (clusters.size() > 1) {
                System.out.println();
                switch (this.choice) {
                case 1:
                        calcDistMin();
                        break;
                case 2:
                        calcDistMax();
                        break;
                case 3:
                        calcDistAvg();
                        break;
                default:
                        break;
                }
```

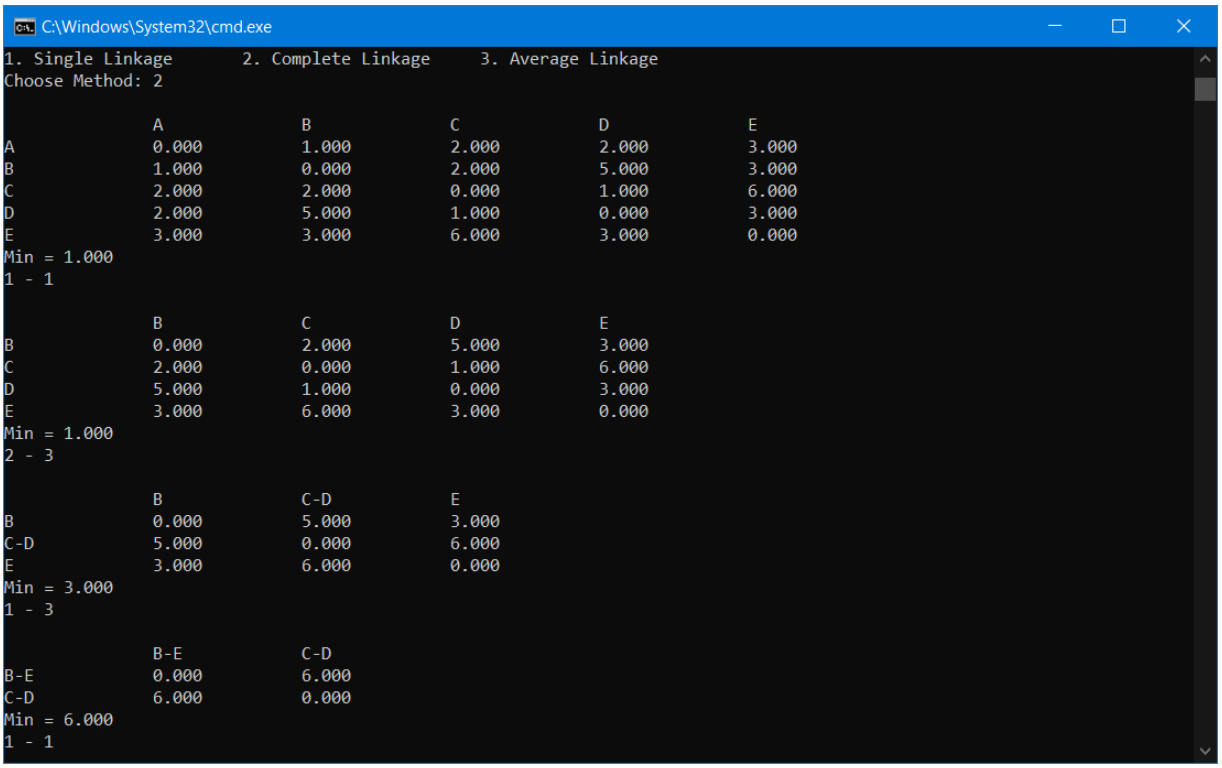
```
                    calcMatrixMin();
                }
        }

        public static void main(String[] args) {
                aglC ac = new aglC();
                ac.input();
                ac.solve();
                System.out.println();
        }
}
```

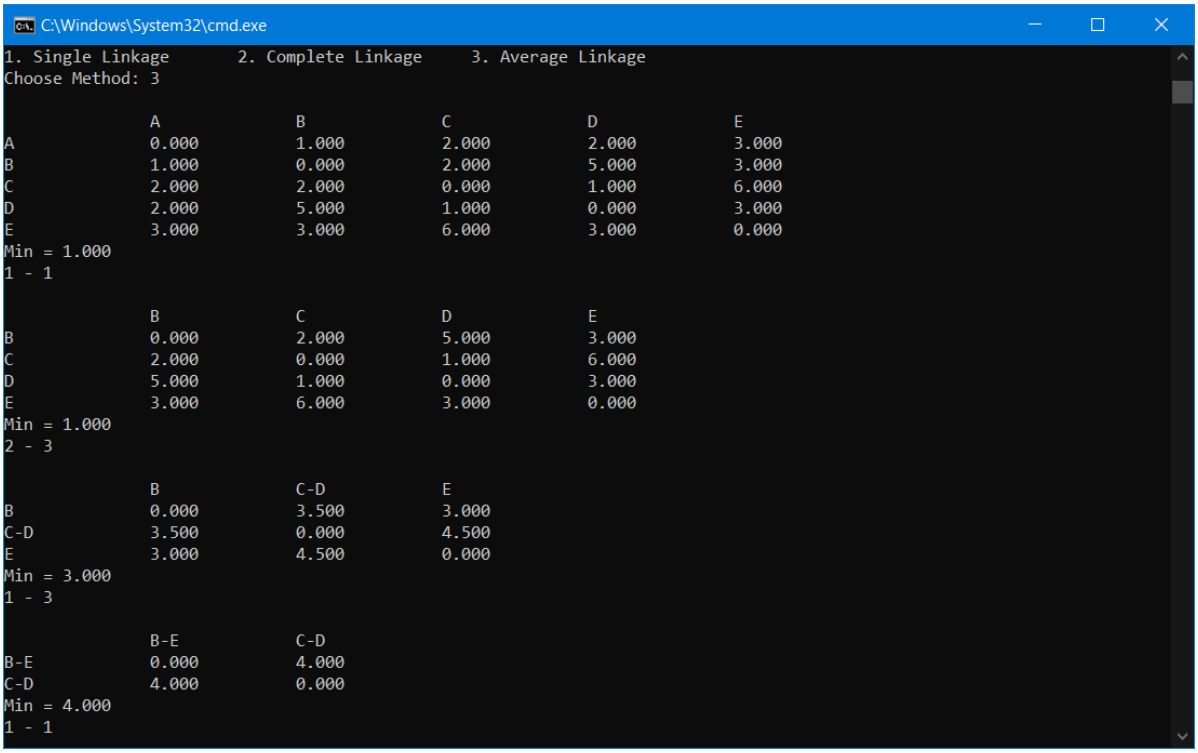
Output:




C:\Windows\System32\cmd.exe
Enter distance matrix
0
1
2
2
3
1
0
2
5
3
2
2
0
1
6
2
5
1
0
3
3
3
6
3
0
1. Single Linkage 2. Complete Linkage 3. Average Linkage
Choose Method: 1


C:\Windows\System32\cmd.exe
A B C D E
A 0.000 1.000 2.000 2.000 3.000
B 1.000 0.000 2.000 5.000 3.000
C 2.000 2.000 0.000 1.000 6.000
D 2.000 5.000 1.000 0.000 3.000
E 3.000 3.000 6.000 3.000 0.000
Min = 1.000
1 - 1
B C D E
B 0.000 2.000 5.000 3.000
C 2.000 0.000 1.000 6.000
D 5.000 1.000 0.000 3.000
E 3.000 6.000 3.000 0.000
Min = 1.000
2 - 3
B C-D E
B 0.000 2.000 3.000
C-D 2.000 0.000 3.000
E 3.000 3.000 0.000
Min = 2.000
1 - 1
C-D E
C-D 0.000 3.000
E 3.000 0.000
Min = 3.000
1 - 1


C:\Windows\System32\cmd.exe
1. Single Linkage 2. Complete Linkage 3. Average Linkage
Choose Method: 2
A B C D E
A 0.000 1.000 2.000 2.000 3.000
B 1.000 0.000 2.000 5.000 3.000
C 2.000 2.000 0.000 1.000 6.000
D 2.000 5.000 1.000 0.000 3.000
E 3.000 3.000 6.000 3.000 0.000
Min = 1.000
1 - 1
B C D E
B 0.000 2.000 5.000 3.000
C 2.000 0.000 1.000 6.000
D 5.000 1.000 0.000 3.000
E 3.000 6.000 3.000 0.000
Min = 1.000
2 - 3
B C-D E
B 0.000 5.000 3.000
C-D 5.000 0.000 6.000
E 3.000 6.000 0.000
Min = 3.000
1 - 3
B-E C-D
B-E 0.000 6.000
C-D 6.000 0.000
Min = 6.000
1 - 1


C:\Windows\System32\cmd.exe
1. Single Linkage 2. Complete Linkage 3. Average Linkage
Choose Method: 3
A B C D E
A 0.000 1.000 2.000 2.000 3.000
B 1.000 0.000 2.000 5.000 3.000
C 2.000 2.000 0.000 1.000 6.000
D 2.000 5.000 1.000 0.000 3.000
E 3.000 3.000 6.000 3.000 0.000
Min = 1.000
1 - 1
B C D E
B 0.000 2.000 5.000 3.000
C 2.000 0.000 1.000 6.000
D 5.000 1.000 0.000 3.000
E 3.000 6.000 3.000 0.000
Min = 1.000
2 - 3
B C-D E
B 0.000 3.500 3.000
C-D 3.500 0.000 4.500
E 3.000 4.500 0.000
Min = 3.000
1 - 3
B-E C-D
B-E 0.000 4.000
C-D 4.000 0.000
Min = 4.000
1 - 1