Project 3 Problem1

Name: Zian Wang

(Here are the test cases and memory usage.)

Program Description:

There are 3 programs for this problem.

merge.java is for reading the two input files and merge them to one complete graph.'
Kruskal_Array.java is for implementing Kruskal algorithm using a two-dimension array as the adjacency matrix.

Kruskal_Linklist.java is for implementing Kruskal algorithm using a link list as the adjacency matrix.

We will see merge.java first: this program can read the two input files and make them a two-dimension array or a link list. The program will firstly read all the points' coordinates, we will use the coordinates to judge whether the two points are the same later. Then we will read the two input files, and the program will use the coordinates to find all the repeated points in the file2, and change their index to the index of the point in the file1. Finally, we will use the two graph which contains **no** repeated nodes to build the two-dimension array or the link list. Finally, when we building two-dimension array or link list, if we find the current index is smaller than last index, this means the current node is the repeated node (this is because the indexes in the file1 is always smaller than that in file2), we then judge whether the end node is also a repeated node, if both are repeated, it means this edge has been contained, so we will ignore this edge, if the second node is not repeated, we contain this edge into the adjacency matrix.

For example, the program will read all the points' coordinates, and if the 299 node in the file1 has the same coordinates as the 932 node in the file2, this means they are the same point in the different file. We know the file1 has 534 nodes, so after we read the files and put the data in the arrays, the index of 932 node is actually 932+534 = 1466 in the array. So then we will then change every 1466 in the arrays to 299. This will delete every repeated nodes in the file2.

This program has 8 primary functions:

public static double[][] getCoordinates(String path1, String path2)
Inputs: the path of the two input files.

Outputs: an array contains all the nodes' coordinates in the two input files.

Function: get the nodes' coordinates so later we can judge whether the two nodes in the different file are the same.

public static void replace(int[] node1, int[] node2, int n1, int n2, int row)
Inputs: two arrays, node1 contains the index of the starting nodes, node2 contains the index of the end nodes. n1 and n2 stand for which node should replace which node. row stands for the number of the rows of the two array.

Outputs: none.

Function: this function will replace every n2 nodes in the node1 and node2 array by n1. This function does not have outputs because the function directly changes the array, so it does not need to output them.

public static int normalize(int[] node1, int[] node2, int row)
Inputs: two arrays, node1 contains the index of the starting nodes, node2 contains the index of the end nodes. row stands for the number of the rows of the two array.

Outputs: the number of the repeated nodes in the two files.

Function: this function will delete the repeated rows in the two arrays.

public static Map merge(String path1, String path2)
Inputs: the path of the two input files.

Outputs: a map objects contains the starting and ending nodes without repeated nodes. The number of the rows in file1 and file2, the number of the repeated nodes.

Function: this function uses all the functions above to read the files and put the data into three arrays: node1 contains the starting nodes, node2 contains the ending nodes, distance contains the distances between the starting node and the ending node.

public static int getNumOfRows(String path)
Inputs: the path of the file.

Outputs: the number of the rows in the file. (The head row not included)

Functions: to get the number of the rows in the input file.

public static int getNumOfNodes(String path)
Inputs: the path of the file.

Outputs: the number of the nodes in the file.

Functions: to get the number of the nodes in the input file.

public static int[][] readArray(Map nodes)

Inputs: the result from the merge function.

Outputs: the two-dimension array as the adjacency matrix.

Functions: using node1, node2 and distance array to build the two-dimension array adjacency matrix.

public static link_list[] readLinkList(Map nodes)

Inputs: the result from the merge function.

Outputs: the link list array as the adjacency matrix.

Functions: using node1, node2 and distance array to build the link list adjacency matrix.

Now we will see the Kruskal_Array.java and the Kruskal_Linklist.java. These two programs are very similar except the first one uses a two-dimension array as the adjacency matrix, the second one uses link list as the adjacency matrix. Both of them has 5 primary functions, and the link list one has the 6 function to find the distance between the two given nodes by using the link list. The Kruskal algorithm is just as same as it in the text book.

public Stack[] initial(int n)

Inputs: the number of the nodes: n.

Outputs: an array of n stacks.

Function: initialization the sets of the nodes. Because we now have not run the algorithm, each node is in one separate set.

public int find(Stack[] V, int n, int target)

Inputs: the array of the stacks, the length of the array, the target node we want to find.

Outputs: the index of the set which contains the target nodes.

Function: to get the index of the set which contains the given node.

public void merge(Stack[] V, int i, int j)

Inputs: the array of the stacks. The indexes of the sets waited to be merge.

Outputs: none.

Function: to merge two given set. This function has no outputs because the function change the arrays directly.

public int[][] kruskal(int n, adjacency)

Inputs: the number of the nodes: n. The adjacency matrix.

Outputs: an array contains all the edges of the minimum-spanning tree.

Function: this function uses the Kruskal algorithm to calculate the minimum-spanning tree, and return the set of the edges in the tree. This function in

the Kruskal_Array.java receives a two-dimension array as the adjacency matrix, and in the Kruskal_Linklist.java, it receives an array of link list as adjacency matrix.

I use a class which is written myself, to define the link list used in the program. The class's name is link_list and I put it together in the .zip file. It is singly linked list. The structure of the link list is as follows:

This figure of link list stands for there is path from 1 to 2, and its length is stored in the weight of the node 2. Also,there is a path from 1 to 3, and its length is stored in the weight of node 3.

There will be an array of link_list, named adjacency_list[], to store the adjacency matrix, and every link_list stores all the paths from one vertex. For example, adjacency_list[x] stands for all the paths starts from vertex x. So the length of the adjacency_list[] is the number of the vertexes in the graph.

Every link list has several nodes, every node has 3 members:
The 1st member: "vertex" stands for the vertex of the node.
The 2nd member: "weight" stands for the weight of the path between the first node's vertex and the vertex of this node, for example: in the link list starts with vertex 0, weight1 stands for length from 0 to 1, weight2 stands for length from 0 to 2, length n stands for length from 0 to n .
The 3rd member: "next" is a pointer points to the next node of the current node.

Test Cases:

Because there are 1993 nodes totally (do not contain repeated nodes), there are 1993-1 = 1992 edges in the minimum-spanning tree, so the printed result is pretty long, so I only put the head and the tail of the result here.

Two-dimension array program:

```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe"
Kruskal using two-dimension array
Node:       Node:       Distance:
79          205         16
1697        1832        20
1617        1919        21
477         511         25
1062        1399        25
1316        1317        25
1117        1119        26
50          404         27
157         188         27
1167        1266        27
1188        1189        27
1730        1753        27
1912        1926        27
927         1494        29
180         506         30
947         1312        30
1077        1616        30
1231        1232        30
485         504         31
1147        1440        31
1427        1428        31
1627        1847        31
394         503         32
843         1006        32
60          98          33
239         429         33
752         780         33
1465        1466        33
946         1476        34
1039        1155        34
1694        1881        34
718         1583        35
1202        1615        35
1540        1541        35
1068        1249        36
1018        1171        37
1159        1160        38
1584        1607        38
1641        1643        38
1726        1727        38
1909        1922        38
1243        1326        40
1501        1505        40
1754        1844        40
195         198         42
557         1926        42
1127        1962        42
584         1918        43
773         774         43
989         1251        43
```

And the tail:

```
1555        1972        642
1691        1983        643
1908        1992        650
1855        1990        652
1463        1464        666
541         1923        669
87          516         693
20          513         733
656         1937        733
1696        1824        740
1749        1987        742
1001        1890        774
706         1004        779
865         866         780
1637        1908        780
83          90          842
565         1927        999
1674        1982        1013
798         1943        1031
1347        1967        1032
1593        1927        1178
1091        1957        1205
22          96          1208
1395        1929        1252
83          487         1395
1890        1991        3703
TotalDistance is: 371466

Process finished with exit code 0
```

The link list program:

```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe"
Kruskal using link list
Node:       Node:       Distance:
79          205         16
1697        1832        20
1617        1919        21
477         511         25
1062        1399        25
1316        1317        25
1117        1119        26
50          404         27
157         188         27
1167        1266        27
1188        1189        27
1730        1753        27
1912        1926        27
927         1494        29
180         506         30
947         1312        30
1077        1616        30
1231        1232        30
485         504         31
1147        1440        31
1427        1428        31
1627        1847        31
394         503         32
```

```
843         1006        32
60          98          33
239         429         33
752         780         33
1465        1466        33
946         1476        34
1039        1155        34
1694        1881        34
718         1583        35
1202        1615        35
1540        1541        35
1068        1249        36
1018        1171        37
1159        1160        38
1584        1607        38
1641        1643        38
1726        1727        38
1909        1922        38
1243        1326        40
1501        1505        40
1754        1844        40
195         198         42
557         1926        42
1127        1962        42
584         1918        43
773         774         43
989         1251        43
```

And the tail:

```
1908        1992        650
1855        1990        652
1463        1464        666
541         1923        669
87          516         693
20          513         733
656         1937        733
1696        1824        740
1749        1987        742
1001        1890        774
706         1004        779
865         866         780
1637        1908        780
83          90          842
565         1927        999
1674        1982        1013
798         1943        1031
1347        1967        1032
1593        1927        1178
1091        1957        1205
22          96          1208
1395        1929        1252
83          487         1395
1890        1991        3703
TotalDistance is: 371466

Process finished with exit code 0
```

Memory usage:

This merge program uses 3 int arrays: node1, node2 and distance to store the data in the two files. Each of them has the same length, it is 4456+1620 = 6076 rows, and one int variable need 4 bytes, so these three arrays totally need 3*6076*4 = 72912 bytes. Also, the adjacency matrix needs memory too. The two-dimension array is sized 1993*1993 because there are total 1993 nodes in the two graphs (do not contain the repeated nodes), so it needs 1993*1993*4 = 15888196 bytes. For the link list, there are (4456+1620)/2 = 3038 edges, and there are 1993 nodes, so there are totally 3038+1993 = 5031 nodes in the link lists. Every node of the link list contains 3 variables, two of them are int, and another one is a pointer, so it will totally cost 5031*(4+4) = 40248 bytes. The final result is a set of the edges, it is an int array of the edges, it is sized (1993-1)*2 = 3984. Therefore, it will cost 3984*4 = 15936 bytes.

In the conclusion, the merge program uses about 72912 bytes to store the data from the input files. The two-dimension array program needs 15888196 bytes to store the adjacency matrix, and the link list program needs 40248 bytes to store the adjacency matrix. Both of them needs 15936 bytes to store the result.