CSI 2110 Computer Science Fall 2016 University of Ottawa

Assignment #4 Programming (75 points, weight 7.5%); Due: Sunday, December 4, 11:59PM

The assignment must be uploaded on blackboard; please follow the submission instructions.

Late submission is only accepted from 1 min – 24 hs late for 30%off (i.e. your mark * 0.7).

Cluster Analysis or Clustering

Clustering involves grouping a collection of objects (documents, photographs, customers, animal species) into coherent groups called clusters so that objects in the same group are more similar, while objects in different groups are less similar to one another. In order to do this, we need a measure of similarity or "distance" between objects, which can be defined depending on the type of objects. If the objects are points in the physical world (for example, touristic sites in a map), the distance may be the physical distance between the points. In other problems, "distance" is defined more abstractly. "Distance between two species may be taken as the number of years since they diverged in the course of evolution; we could define distance between two images in a video stream as the number of corresponding pixels at which their intensity values differ by at least some threshold." (Kleinberg and Tardos, Algorithm Design, 2005). In data mining, data can be described by some main attributes, and distance can be measured by the ratio between the number of attributes that do not match over the total number of attributes (simple matching) or more complex definitions of distance can be used. The clusters can help identify structure in data to learn something about it. There are different approaches to clustering. We will explore clustering by maximizing the space of objects in different clusters, which can be efficiently solved using minimum spanning tree algorithms. In this assignment you will have programming practice with graph ADT, Kruskal's minimum spanning tree algorithm and union-find data structure.

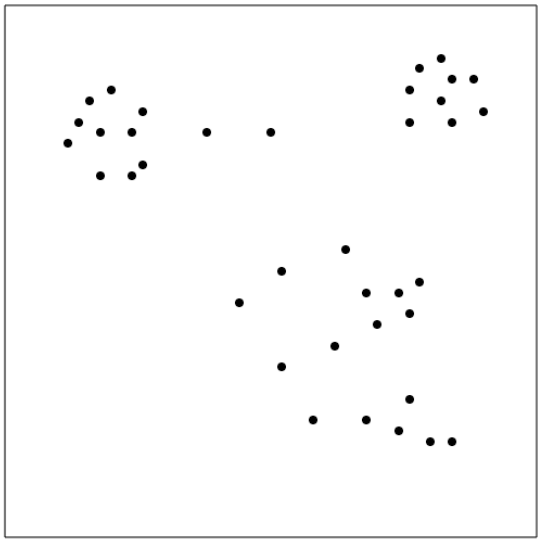


Figure 1: Points in a 2D space.

1 Clustering of maximum spacing and minimum spanning trees MST

W describe a clustering approach based on maximum spacing (Kleinberg&Tardos, 2005):

Given a set U of n objects, labeled p1, p2, …, p_n. For each pair pi and pj, we have a numerical distance d(pi,pj). We only need that this distance satisfies d(pi,pi)=0, d(pi,pj)>0 for distinct pi and pj, and that distances are symmetric: d(pi,pj)=d(pj,pi).
Given a parameter k, we want to divide the objects in U into k groups. We say that a *k-clustering* of U is a partition of U into k nonempty subsets C1, C2, …, Ck. We define *spacing* of a k-clustering to be the minimum distance between any pair of points lying in different clusters. We want to find a k-clustering of maximum possible spacing.

Finding a k-clustering of maximum spacing can be accomplished by a greedy algorithm that works like running the steps of Kruskal's algorithm until the spanning forest generated has exactly k components. See the book by Kleiberg and Tardos (Algorithm Design, 2005, Chapter 4.7) for a proof that this algorithm solves the maximum spacing k-clustering problem.

The following example shows the result of applying 3-clustering on the example from the previous page.

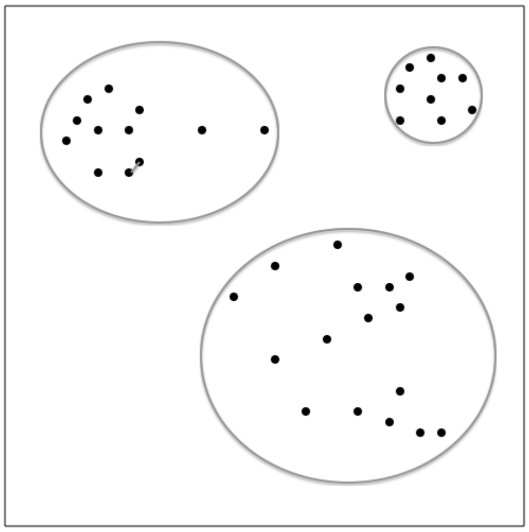


Figure 2: A 3-clustering of maximum spacing

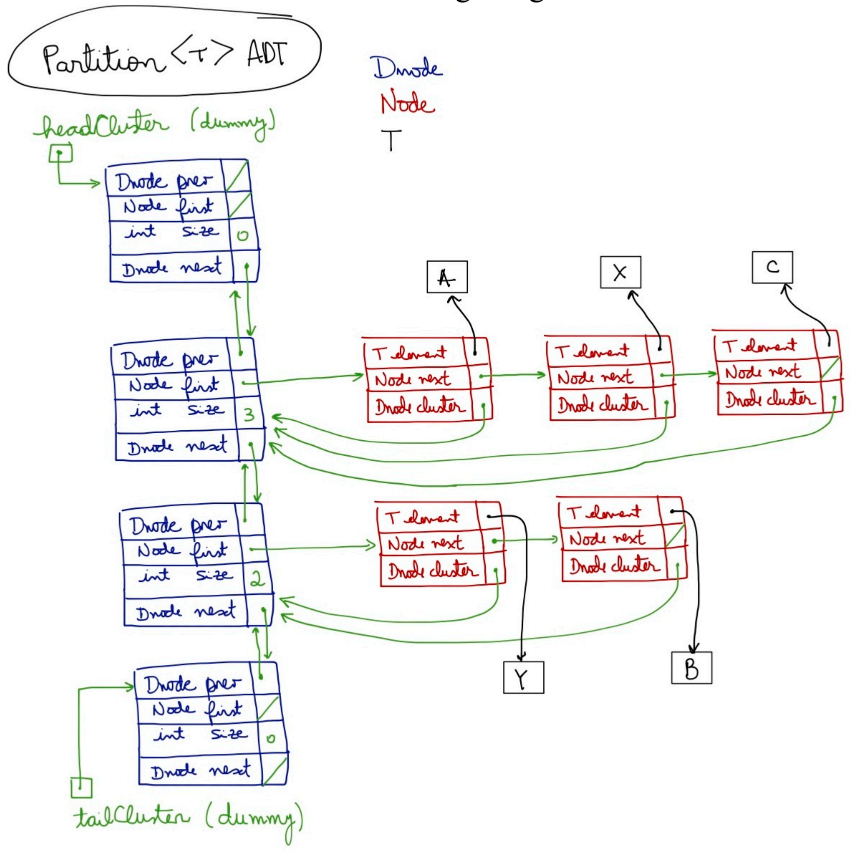
Imagine a graph where the vertices correspond to the points and there is an edge between any two vertices where the weight of the edge is the Euclidean distance between the points corresponding to the vertices that are endpoints of the edge. In the example of Figure 2, there are 37 vertices and 37*36/2=666 edges. The first edge added to the spanning tree is marked in the picture as it joins the vertices that are closest to each other. After adding the first 34 edges in Kruskal's algorithm we will have 3 clusters. In general, for a connected graph with n vertices, after adding the first n-k edges we have k connected components, which will form the k clusters for the k-clustering problem.

2 Implementation details and what you must do

You will implement the following classes, whose skeleton has been provided.

MyGraph : a graph using the Adjacency Matrix representation of a graph. It implements a simplified version of the textbook interface Graph<V,E>, where we removed the graph update methods. Explanations are included in MyGraph.java and you must implement all the operations indicated. Testing must be done specifically for this class (see example MyGraphTest.java for a sample). Note that we are using the Adjacency Matrix data structure since it is a simple structure that is good enough for dense graphs. Indeed, the graphs for the clustering application (Part 3) are complete graphs so every degree is n-1; methods like incidentEdges(v) that run in O(degree(v)) for the adjacency lists data structure would run in O(n) for this type of graphs, so there would be little advantage in using adjacency lists.

Partition<T>: a data structure that stores clusters of objects of type T and implements the union-find ADT via linked lists. Initially each object is it is own cluster. The operation union(x,y) merges the clusters of x and y and the operation find(x) returns the cluster that object x belongs to. Each cluster is represented by a Dnode in a doubly linked list of clusters with dummy head and tail; each Dnode (cluster) links to a singly linked list of Node (each node containing an element in the cluster). The Diagram below depicts this data structure representing two clusters{X,A,C} and {Y,B}. This should follow a similar idea as the sequence implementation of the Partition ADT in page 672 of the textbook. Operations makeCluster and find run in O(1) while operation union runs in O(min) where min is the size of the smallest of the two clusters being merged.



KruskalAlgorithms: the main method for you to implement is
public LinkedList <Edge<E>> kruskalClusters (Graph<V,E> g, int k, Partition<Vertex<V>> clusters)
It receives a graph g and the desired number of clusters k and computes a partition of the vertex set storing the vertices in each cluster and returns the edges on the spanning forest that connects vertices in each cluster.
This method runs some steps of Kruskal's algorithm until the spanning forest being created has exactly k components.
If g is the graph with 37 vertices and 666 edges corresponding to the points in the plane in Figure 2, and if k=3, then clusters will represent the 3 sets of points circled in Figure 2 and the method will return 34 edges that were added to the spanning forest.
If g is a graph and k=1, then clusters will represent a set with all the vertices and the method will return a minimum spanning tree (which by the way for the example it would have 36 edges).
A pseudocode of this method, based on Kruskal's pseudocode in the textbook, is given next.

```
Algorithm KruskalClusters(G,k)
Input: a simple connected weighted graph G with n vertices and m edges
Output: a spanning forest T equivalent to deleting the last (k-1) edges from a MST of G
        and a partition P that is a solution to the k-cluster problem.
Consider P a partition of vertices (see the partition ADT in section 14.7.1).
for each vertex v in G do
        P.makeCluster(v)   // this defines an elementary cluster C(v)={v}.
Initialize a priority queue PQ to contain all edges in G, using the weights as keys.
T = emptyset of edges
while T has fewer than (n-k) edges do
        (u,v)=edge returned by PQ.removeMin()
        C(u)=P.find(u); C(v)=P.find(v); //find the cluster containing u and the one containing v
        if C(u) ≠ C(v) then
                Add edge (u,v) to T.
                P.union(C(u),C(v)) // Merge C(u) and C(v) into one cluster
return forest edges T and partition P
```

3 Assignment Tasks and Mark Breakdown (75 marks)

PART 1) (25 marks) Implement the missing methods for class MyGraph which implements the interface Graph<Integer,Double> where the Integer is a vertex index (0..n-1) and the Double is the edge weight (distance). A skeleton has been provided.
Thoroughly test each method of this class for various types of graphs.
Basic test provided: MyGraphTest1.java

PART 2) (10 marks) Implement the methods `find` and `union` of class Partition<T>.
Thoroughly test each method of this class for various examples.
Basic test provided: PartitionTest2.java

PART 3) (15 marks) Implement the method kruskalClusters following the given skeleton and the explanations and pseudocode in section 2.
Thoroughly test each method of this class with different graphs.
Use tests for the k-clustering application. Some tester programs will be provided using data from the following dataset repository:
https://people.sc.fsu.edu/~jburkardt/datasets/hartigan/hartigan.html)
Basic test provided: MyGraphTest123.java, Clusters2DTest123.java

The remaining 25 marks will be distributed for results of more complete testing within the 3 parts above, or for any rebalancing of marks we may decide afterwards.

4 What to hand in and submission instructions

1) Create one directory with name being the letter "s" followed by your student number.
If your student number is 564789, your directory will be called s564789

2) Inside the directory include the following:

- the source code of your program, i.e. **all the *.java files for your program**.
  (these includes all files that we provided and where 3 of them contain your changes)
- files called **outputP1.txt outputP2.txt outputP3.txt** showing the output obtained with standard testing provided in blackboard learn (the recommended tests will be published later).
- A file called **README.txt containing:**
  - Any information relevant for the TA to mark your program. For example, if not all parts of the code are working you may inform the TA here which parts are working, explain known bugs, etc.

3) zip the directory containing all the files above and submit (in the example of the student number above the file submitted will be called s564789.zip) When the TA unzips the file, it should see the directory s564789 containing all files described above.