Programming Assignment #2

Quiz, 2 questions

2/2 points (100%)

✓ **Congratulations! You passed!**

Next Item

✓ 1 / 1 point

1.

In this programming problem and the next you'll code up the clustering algorithm from lecture for computing a max-spacing k-clustering.

Download the text file below.

clustering1.txt

This file describes a distance function (equivalently, a complete graph with edge costs). It has the following format:

[number_of_nodes]

[edge 1 node 1] [edge 1 node 2] [edge 1 cost]

[edge 2 node 1] [edge 2 node 2] [edge 2 cost]

...

There is one edge (i, j) for each choice of $1 \leq i < j \leq n$, where n is the number of nodes.

For example, the third line of the file is "1 3 5250", indicating that the distance between nodes 1 and 3 (equivalently, the cost of the edge (1,3)) is 5250. You can assume that distances are positive, but you should NOT assume that they are distinct.

Your task in this problem is to run the clustering algorithm from lecture on this data set, where the target number k of clusters is set to 4. What is the maximum spacing of a 4-clustering?

ADVICE: If you're not getting the correct answer, try debugging your algorithm using some small test cases. And then post them to the discussion forum!

✓ 1 / 1 point

2.
In this question your task is again to run the clustering algorithm from lecture, but on a MUCH bigger graph. So big, in fact, that the distances (i.e., edge costs) are only defined *implicitly*, rather than being provided as an explicit list.

The data set is below.

clustering_big.txt

The format is:

[# of nodes] [# of bits for each node's label]

[first bit of node 1] ... [last bit of node 1]

[first bit of node 2] ... [last bit of node 2]

...

For example, the third line of the file "0 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1" denotes the 24 bits associated with node #2.

The distance between two nodes u and v in this problem is defined as the *Hamming distance*--- the number of differing bits --- between the two nodes' labels. For example, the Hamming distance between the 24-bit label of node #2 above and the label "0 1 0 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1" is 3 (since they differ in the 3rd, 7th, and 21st bits).

The question is: what is the largest value of k such that there is a k-clustering with spacing at least 3? That is, how many clusters are needed to ensure that no pair of nodes with all but 2 bits in common get split into different clusters?

NOTE: The graph implicitly defined by the data file is so big that you probably can't write it out explicitly, let alone sort the edges by cost. So you will have to be a little creative to complete this part of the question. For example, is there some way you can identify the smallest distances without explicitly looking at every pair of nodes?

Since number of points is quite big, we cannot write out the graph with all the distances between all the points explicitly. It will be $200000 * 199999/2 = 19999900000$ edges, which is too much. We will use another approach instead.

What we need to do in this task is put two points in one cluster iff the distance between them is 1 or 2. Let's call such points *neighbors*. How can we find all of the neighbors for a given point u? Let's think of every point as 24-bit representation of some integer number. Then neighbors of u are such points that differ from u in no more than 2 bytes. It means that x is a neighbor of u iff $xXORu$ is a number that has only 1 or 2 ones in the binary representation. We will call such numbers *simple*. Total number of *simple* numbers is $\binom{24}{2} + \binom{24}{1} = 300$.

We will use the following property of XOR: if $a\ XOR\ b = c$ than $a\ XOR\ c = b$. It means that if we $XOR\ u$ by all possible *simple* numbers we will get all theoretically possible neighbors of u. If we leave only those numbers that lie in S we will have the neighbors of u in S.

To store the data we will use array `points` of 2^{24} integers. Initially we fill it with zeroes. While reading next line from the file we will compute corresponding integer `curindex` , and write into `points[curindex]` next integer to denote initial division into clusters where each point is its own cluster. We will also save numbers corresponding to every point in S indo array `index` for future purposes.

How do we compute the whole cluster for u? We compute neighbors, assign them to the u-th cluster, then compute neighbors for every neighbor of u, leave only points that are not yet assigned to u-th cluster, assign them to the u-th cluster, then compute neighbors for every neighbor of neighbor of u, leave only new points, assign them to u-th cluster and so on, until on some step we won't get any new points for u-th cluster.

To perform complete clustering we will go through all of our points and if the next point does not yet belong to some previous full cluster we will find full cluster for that point. We will save leaders of the clusters into `name` array and in the end the length of this array (minus 1 for the initial zero) will give us the answer.