Homework 1

Algorithm Design 2018-19 - Sapienza

Luigi Russo 1699981

November 18, 2018

1 Cristina and centers

Cristina is interested in a metric space (X, d), where all distances d(x, y) are either 0 (in which case x = y), 1 or 3. Further, all distances are symmetric and obey the triangle inequality, that is $d(a, b) + d(b, c) >= d(a, c)$. Cristina wants to cluster the points of X. Unfortunately, she does not yet know how many clusters she has to use. She therefore will find a permutation $\Pi(X)$ and use, for every $k \in \{1, ..., |X|\}$, the first k elements $C = \pi(X)_1, ..., \pi(X)_k$ of the permutation as centers.

Goal: For any such k, the output should be optimal with respect to following objective function:

$$\max_{x \in X} \min_{c \in C} d(x, c). \tag{1}$$

Hint: Use a greedy algorithm. The running time should be no larger than $O(|X|^2)$

Solution algorithm

This problem can be seen as a variant of the well-known K-centers decision problem. I have designed a greedy algorithm that solves the above problem. It creates (and returns to Cristina) the desired permutation in this way.

1. $C := \emptyset$
2. pick a random point c1

3. for every point v *in* V, compute distance from c1 to v
4. pick the point c2 with highest distance from the points in C
5. add c2 to C and continue until $|V| = |C|$

Running time

The algorithm adds $|X|$ points to C. How much costs adding a point to C? A point is added to C if among all the points not in C it has the maximum distance from the ones in C. At each iteration we update the minimum distance from all points to C, simply considering if the new center can "decrease" the distance. It costs $|X - 1|$ for the second center, $|X - 2|$ for the third... so it's never greater than $|X|$. Since the iterations are $|X|$ and each one costs $|X|$ we have the cost is $|X^2|$

Proof of correctness

At each iteration the algorithm updates (if possible) the current maximum_distance from the set of centers. This leads to a non increasing sequence of max_values $= d_1, d_2, ...d_{|V|}$. Given an optimal solution O, we have to prove that the sequence computed with the greedy algorithm above is such that for every $i \in 1, 2, ...|V|$, $dg\_i = do_i$, where dg is the sequence of max distances of the greedy algorithm and do the one of the optimal algorithm. Since my greedy algorithm guarantees that for every k, $dg_k \leq 2 * do_k$ and since the only possible values of d are 0, 1 or 3, we have these possible cases: $do_k = 0 => do_k = 0 \leq dg_k \leq 2 * 0 = 0 => dg_k = 0 OK$ $do_k = 1 => do_k = 1 \leq dg_k \leq 2 * 1 = 2 => dg_k = 1 OK$, since 2 is not a possible value. $do_k = 3 => do_k = 3 \leq dg_k \leq 2 * 3 = 6 => dg_k = 3 OK$, since 3 is the only value greater or equal than 3 and less or equal than 3. This is true for every k. We have that the greedy algorithm computes as well as the optimal solution.

2

3

4

5 Federico and MST

Federico is a mathematician who doesn't like stories and wants the exercise to get to the point.
Goal: We are given a weighted graph G(V, E). Let e $\in$ E.

1. Design an algorithm that decides whether or not there exists a minimum spanning tree containing e. For full marks, the algorithm must run in time at most $O(|V| + |E|)$.

2. Design an algorithm that computes a minimum spanning tree containing e, if one exists. For full marks, the algorithm must run in time at most $O(|E|log|E|)$. Implement the algorithm with a programming language of your choice.

5.1

Given e = (u,v):

1. run a DFS from the endpoint u **considering only those edges that have weight less than that of e**.

2. Two possible cases:

   (a) If during the dfs there is an edge that leads to node v, then the edge e does not belong to any MST.

   (b) If the dfs terminates and case 1 never happens, then there exist some MST that contains e.

5.1.1 Pseudo-code

TODO

5.1.2 Proof of correctness

I have used the MST cycle property in this way: we know that given a cycle in a graph, the largest edge among the ones that form the cycle cannot belong to a MST. So, the dfs above tries to connect u and v with edges strictly lower than e: if this happens, it means that there exist some cycle in the MST that connects u and v and, of course, contains the edge e. Since all the edges considered during our "custom" dfs are strictly lower than e, e is the largest. So, by the MST cycle property, e cannot belong to a MST. Since this condition is necessary and sufficient, this is enough!

5.2

1. Run the algorithm 5.1: if e cannot belong to any MST throw an error
2. Run Boruvka, whit all disconnected components, except u and v, connected with edge e.

5.2.1 Code

boruvka.py

5.2.2 Proof of correctness

The algorithm 5.1 will be able to verify that a MST with edge e really exists. Once we know that such MST can be built, we run a custom version of Boruvka. Since we know that a MST with edge e exists, we also know that u and v belong to the same connected component; moreover we know that e must be part of the MST and e is the edge (the only one of course in our output) that has to connect u and v. So we start Boruvka initializing all the nodes (except u and v) as disconnected components: as for u and v, we assign them to the same connected component (with edge e). Now, Boruvka algorithm will run and produce a MST, since it finds a MST starting from disconnected components. **Note:** It will fail if the graph is not connected actually (we can assume the graph is always connected, or we can check easily at the end of the algorithm if the current MST is really a tree: the number of edges "returned" must be n-1; or we could run a connectivity algorithm check before the MST!).