Problem 1. Let $G = (V, E)$ be an undirected simple graph where each edge $e \in E$ is associated with a non-negative weight $w(e)$. For any vertices $u, v \in V$, define $spdist(u, v)$ as the shortest path distance between u and v. Given a subset $C \subseteq V$, define its cost as $cost(C) = \max_{u \in V} \min_{c \in C} spdist(c, u)$. Fix an integer $k \in [1, |V|]$. Let OPT be the smallest cost of all subsets $C \subseteq V$ with $|C| = k$. Design an algorithm to find a size-k subset with cost at most $2 \cdot \text{OPT}$. Your algorithm must run in time polynomial to $|V|$.

Solution. First, calculate the shortest path distances between all pairs of vertices in V. This can be done in polynomial time by resorting to Dijkstra's algorithm. Then, run the k-center algorithm on V. Specifically, initialize an empty set C and add to C an arbitrary vertex. Then, repeat the following step until $|C| = k$: add to C the vertex u maximizing $min_{c \in C} spdist(c, u)$.

The approximation ratio 2 remains valid as long as the distance function satisfies the triangle inequality. It is clear that shortest path distances satisfy the triangle inequality.

Problem 2. Consider the k-center problem on a set P of n 2D points. Modify our 2-approximate algorithm to make it run in polynomial time without the assumption that the Euclidean distance of any two points can be computed precisely in polynomial time.

Solution. Note that the precise value of distance of two points does not matter, and we only need to ensure the relative size of distance is preserved. Instead of using the Euclidean distance, we can use the squared-Euclidean distance as $f(x) = x^2$ is strictly increasing. It is clear that the distance can be computed in $O(1)$ time.

Problem 3. Let P be a set of n 2D points. Given a subset $C \subseteq P$, define:

- $dist_C(p) = \min_{c \in C} dist(c, p)$, where $dist(c, p)$ represents the Euclidean distance between c and p, for each point of p;
- $cost(C) = \max_{p \in P} dist_C(p)$.

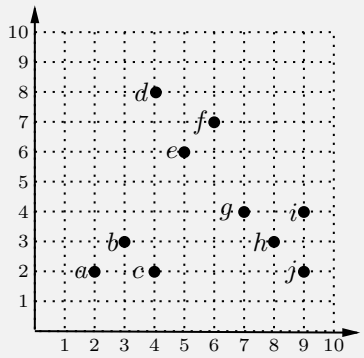
Fix a real value $r > 0$. Call a subset $C \subseteq P$ an r-feasible subset if $cost(C) \leq r$. Prove: unless $\mathbf{P} = \mathbf{NP}$, there does not exist an algorithm that can find an r-feasible subset with the smallest size in time polynomial to n. You can assume that the Euclidean distance of any two points can be computed in polynomial time. (Hint: Show that the existence of such an algorithm implies a polynomial time algorithm for the k-center problem.)

Proof. Suppose the otherwise that we are given an algorithm $\mathcal{A}$ that can solve the r-feasible subset problem in polynomial time. We claim that the k-center problem can be solved in polynomial time, by using $\mathcal{A}$ as a blackbox.

Create a set D of the size $\binom{n}{2}$ containing the distance of every pair of points. We sort the distance in ascending order in $O(|P| \log |P|)$ time. Then, we run $\mathcal{A}$ on each $r \in D$, if $\mathcal{A}$ successfully returns a solution, $cost(C)$ is at most r. This bound is tight, as $cost(C) = \max_{p \in P} \min_{c \in C} dist(c, p)$, and $cost(C)$ can only come from the set D.

Hence, we have a polynomial-time algorithm for the k-center problem, which is impossible unless $\mathbf{P} = \mathbf{NP}$. □

Problem 4. Consider the following set P of points:



Run the k-center algorithm on P with $k = 3$. Suppose that the first center has been chosen to be f. Show what are the second and third centers found by the algorithm?

Solution. a and j, respectively.

Problem 5. Extend the k-center algorithm to 3D space and design a 2-approximate algorithm.

Solution. The algorithm can be directed applied to the 3D space with the only modification of the distance computation function to 3D points. The approximation ratio is direct from the 2D case as the correctness of the k-center algorithm has no dependence on the dimension of the distance functions.

Problem 6. Explain how the k-center algorithm can be implemented in $O(nk)$ time. You can assume that the Euclidean distance between any two points can be calculated in $O(1)$ time.

Solution. We maintain a distance array D dynamically that stores $\min_{p \in C_i} d(p, p_j)$ for every point $p_j \in P$ which ensures that we can find a good point to be picked in the next iteration $i + 1$ in $O(n)$ time. Then, the algorithm can be finished in $O(nk)$ time.

Initially, set $D[p] = \infty$ for all $p \in P$. Hence, we have to update the array after any point p' is picked in the i-th iteration. As

$$\min_{p \in C_i} d(p, p_j) = \min \left\{ \min_{p \in C_i \setminus \{p'\}} d(p, p_j), d(p', p_j) \right\},$$

we only need to consider the distance $d(p', p_j)$ to update $D[p_j]$.