1. ***Clustering.*** There are different ways to formalize the problem of *clustering*, where the goal is to divide up a collection of "objects" into groups that are similar to one another.

   A natural way to express the input to a clustering problem is via a set of objects $p_1, p_2, \ldots, p_n$ with a numerical distance $d(p_i, p_j)$ defined on each pair. We require only that $d(p_i, p_i) = 0$; that $d(p_i, p_j) > 0$ for distinct p_i and p_j; and that distances are symmetric: $d(p_i, p_j) = d(p_j, p_i)$.

   A reasonable formulation of the clustering problem can be defined as follows: Divide the objects in k sets so as to *maximize* the minimum distance between any pair of objects in distinct clusters. This turn out to be solvable by a nice application of the Minimum Spanning Tree Problem.

   A different but seemingly related way to formalize the clustering problem would be as follows: Divide the objects into k sets so as to *minimize* the maximum distance between any pair of objects in the same cluster. Notice that where the formulation in the previous paragraph sought clusters so that not two were "close" together, this formalization seeks clusters that none of them is too "wide" – that is, no cluster contains two points at a large distance from each other.

   Given the similarities, it's perhaps surprising that this new formulation is computationally hard to solve optimally. Let's write it first as a decision problem, the Clustering problem:

   > Given n objects $p_1, \ldots, p_n$ with distances $d(p_i, p_j)$ for each pair p_i, p_j as above, an integer $k \leq n$, and a bound B,
   >
   > decide whether the objects can be partitioned into k sets, so that no two points in the same set are at distance greater than B from each other?

   Prove that Clustering is NP-complete. Hence, if it would exist a polynomial time algorithm that computes a k-partition of the objects so as to minimize the maximum distance between any pair of objects in the same cluster, then P = NP.

El Clustering problem es NP ya que dado un verificador podemos comprobar si es la solución del problema en tiempo polinómico. El verificador que consiste en un vector con k posiciones diferentes y que cada posición i contiene todos los vértices que pertenecen al cluster i. La verificación consiste en ir de cluster en cluster mirando que la distancia entre cada par de vértices sea menor que B. Esto tiene coste $O(n^2)$.

Demostración de que Clustering es NP-hard.

Reducción a Colorability

Definimos el grafo que construimos del set de n objetos (p1,p2,p3 …. pn) que serán los vértices, y las distancias que son vértices con “pesos” (las distancias no tienen dirección porque son simétricas, entonces el grafo no es dirigido).

Sobre este grafo, aplicaremos una poda para reducir el problema de clustering al de K-colorability, que como ya sabemos es NP-completo.

K-colorability <k K-clustering

Dado el grafo que hemos construido anteriormente, podemos podar todas las aristas con distancia menor al bound B de la entrada del problema de K-clustering. Esto resulta en un grafo en que solo tenemos las aristas que sus extremos deben pertenecer a clusters distintos.

De esta manera podemos imaginar que el problema del clustering sobre este nuevo grafo es encontrar una serie de k sets que no contienen ninguna arista entre vértices del mismo set. Y esto es exactamente el problema de la colorabilidad.

Cada color representa un cluster, y la restricción de la colorability se encarga de que entre dos vértices de un mismo cluster (un mismo color) no pueda existir una arista. Y como las únicas aristas del grafo que quedan son las que unen elementos que deben pertenecer a clusters distintos, la restricción del colorability se encarga de que no existan dos elementos de un mismo cluster con distancia mayor al bound.

Resumiendo y repitiendo:

- Podamos el grafo de entrada quitando todas las aristas que tienen distancia menor al bound B. Entonces las aristas que quedan en el grafo tienen por extremos vértices que no pueden pertenecer al mismo cluster.
- Luego reducimos a k-color. Si teniamos k clusters, tendremos k colores:
  - Si existe una k-coloración del grafo podado, los clusters son los sets de vértices de un mismo color. Y como no hay aristas entre vértices del mismo color, quiere decir que en el grafo original las aristas entre vértices del mismo color han sido podadas. Es decir, que entre dos vértices del mismo color la arista era de distancia menor que B.

    Formalización:
    Hay solución del k-color $\Rightarrow$
    $\forall$ u, v vértices del mismo color $\nexists$(u,v) arista que las une $\Rightarrow$
    La arista entre esos dos elementos en el grafo original ha sido podada $\Rightarrow$
    $\forall$ u, v vértices del mismo color, la arista en el grafo original era menor que B

- ➢ Si no existe una k-coloración, significa que no podemos partir el grafo en k sets tal que todo set no tenga arista entre dos elementos de un mismo set.

  Formalización:
  No existe k-coloración ⇒
  No podemos hacer k grupos sin aristas con extremos en el mismo grupo ⇒
  Las aristas que quedan en el grafo
  determinan que dos vértices son de clusters distintos ⇒
  No podemos crear k clusters que separen los extremos (vértices) de las aristas de longitud mayor que B en el grafo original.

Finalmente, si pudiéramos resolver Clustering en tiempo P, entonces lo podríamos usar para resolver problemas de k-colorability en tiempo P. Concretamente para encontrar soluciones al problema de optimización de coloring (“cual es el número mínimo de colores para colorear este grafo” no el decisional). Y como k-colorability en NP-completo esto significa que P = NP. (esto era lo que te pedía comentar al final el enunciado)

Nota: las formalizaciones son una guia, entendemos que quizás deberíamos detallar más los cuantificadores y fórmulas lógicas usadas, pero hemos pensado mejor no complicarlo que suficiente simple es el ejercicio como para ponerse a rizar el rizo. Pero estamos abiertos a correcciones.