Lab06: Vertex_Cover con árboles de búsqueda

1. **Validar un posible vertex_cover**

   En este ejercicio dispones del fichero `lab06_vertex_cover_2.py`, donde tienes que implementar la función `partial_validity_check`. Dicha función, dada

   - una lista de 0s, 1s y `Nones` que indica los nodos del cover: si i es la posición de un 1, el i-ésimo nodo está en el cover. Si i es la posicion de un 0, el i-esimo nodo no esta en el cover. Si i es la posición de un `None`, no sabemos si el i-ésimo nodo estará en el cover o no.
   - y un grafo no dirigido,

   decide si los nodos del cover pueden llegar a cubrir todas las aristas del grafo o no.

   Para probar la función debes eliminar los comentarios de todas las instrucciones `assert` correspondientes a `partial_validity_check` que se encuentran en `test()`.

2. **Encontrar un vertex_cover usando un árbol binario de búsqueda**

   En el fichero `lab06_vertex_cover_2.py` encontrarás la función `vertex_cover_tree` que dado un grafo, inicializa las variables y llama a la función `recursive_vertex_cover`, la cual debes implementar parcialmente.

   La función `recursive_vertex_cover` encuentra un mínimo vertex_cover, al elegir un nodo v que no se ha procesado y posteriormente generar las dos ramas de un árbol de búsqueda. Una rama corresponde a incluir v en el cover y la otra rama a no incluirlo.

   La parte que tienes que implementar debe comprobar si todavía es posible construir un cover válido. Si no es posible, debe devolver el cover `[1]*len(cover)`. Si todavía es posible, debe encontrar un nodo v que aún no se ha procesado. Si no existe ese v porque el cover está completo, debe devolver el cover. En otro caso, debe elegir v y continuar con la parte de código que ya está escrita en la función. Esta parte no se puede modificar.

   Para probar la función debes eliminar los comentarios de todas las instrucciones `assert` correspondientes a `vertex_cover_tree` que se encuentran en `test()`.

3. **Encontrar un vertex_cover usando un árbol ternario de búsqueda**

   Abre el fichero `lab06_vertex_cover_3.py`. Este fichero es como el del ejercicio anterior (`lab06_vertex_cover_2.py`) salvo que ahora, la función `recursive_vertex_cover` genera tres ramas en el árbol de búsqueda.

Esta función recursiva encuentra un mínimo vertex_cover, al elegir dos nodos u y v, que están conectados y no se han procesado. Tras elegir u y v genera tres ramas del árbol de búsqueda. La primera rama corresponde a incluir u y no v en el cover; la segunda corresponde a no incluir u e incluir v en el cover; y la tercera rama corresponde a incluir tanto a u como a v.

La parte que tienes que implementar debe comprobar si todavía es posible construir un cover válido. Si no es posible, debe devolver el cover `[1]*len(cover)`. Si todavía es posible, debe encontrar dos nodos u y v conectados y que aún no se han procesado. Si esto no es posible es que sólo quedan nodos sin procesar conectados a otros ya procesados. Por tanto, habrá que completar el cover y dar valores a estos últimos dependiendo de si sus nodos adyacentes están en el cover o no. Una vez hecho esto, debe devolver el cover completo. En otro caso, debe elegir u y v y continuar con la parte de código que ya está escrita en la función. Esta parte no se puede modificar.

Utiliza las funciones que has implementado previamente en el fichero `lab06_vertex_cover_2.py` y cópialas en `lab06_vertex_cover_3.py`. Por ejemplo te hará falta la función `partial_validity_check`.

4. **Tiempo de ejecución**

   Compara los tiempos de ejecución de la función `vertex_cover_tree` cuando se ejecuta con la versión de `recursive_vertex_cover` que genera un árbol binario y cuando se ejecuta con la versión que genera un árbol ternario.

   ¿Qué versión tarda más?

5. **Recorrido en profundidad frente a recorrido en anchura**

   En este laboratorio has usado árboles de búsqueda para encontrar un mínimo vertex-cover. El recorrido utilizado en cada árbol ha sido en profundidad. ¿Si hicieras el recorrido de los árboles en anchura, crees que ganarías en eficiencia?