Recorridos

- **Un recorrido en anchura desde un vértice origen permite encontrar el conjunto de vértices de un componente conexo.**
  Verdadero. Cualquier recorrido, es lo que hacen los recorridos de grafos, recorrer todos los vértices de todas las partes conexa que formen parte del grafo, solo una vez.

- **Es necesario marcar los nodos visitados en el recorrido de un grafo para evitar provocar ciclos en el mismo.**
  ¿Yo puedo evitar provocar un ciclo en un grafo? NO, si un grafo es cíclico, no se puede evitar que tenga ese ciclo. Se puede evitar procesar más de una vez el nodo. Tú no puedes provocar un ciclo en el grafo, el grafo es el que es.
  En el recorrido de un grafo, se marcan los nodos para evitar, si hay ciclo, no procesar más de una vez un nodo, y para si hay distintas componentes conexas, saber que las he recorrido todas.

- **Al realizar el recorrido de un grafo es necesario ir marcando los nodos visitados, excepto si el grafo es conexo y acíclico.**
  Verdadero, cuando se recorre un grafo, se van marcando los nodos visitados, por si el grafo contiene algún ciclo, evitar recorrer un nodo más de una vez. Pero al ser el grafo acíclico, que no tiene ciclos, esto deja de ser necesario.

- **Es necesario marcar los nodos visitados en el recorrido de un grafo para evitar provocar ciclos en el mismo.**
  Falso.

- **Al realizar el recorrido de un grafo es necesario ir marcando los nodos visitados, excepto si el grafo es conexo y acíclico.**
  Verdadero.

- **Es necesario marcar los nodos visitados en el recorrido de un grafo para evitar provocar ciclos en el mismo.**
  Falso

Dijkstra

- **Se podría optimizar el algoritmo de Dijkstra usando un APO, en particular a la hora de buscar el nodo más cercano al origen que todavía no ha sido usado para mejorar a los demás.**

Verdadero. (Buscar en el foro)

- **Ni Dijkstra ni Floyd funcionan correctamente con costes negativos, al contrario que Prim y Kruskall.**
  Verdadero. Prim y Kruskall funcionan con costes negativos.

- **Siempre que tengamos un origen o un destino definido, debemos usar Dijkstra o Dijkstra inverso en vez de Floyd, por cuestiones de eficiencia, a la hora de calcular el árbol de expansión mínimo.**
  Falso, ninguno de estos algoritmos devuelve un árbol de expansión mínimo.
  Floyd devuelve un matriz de coste de ir de todos a todos.
  Dijkstra devuelve un vector de coste de ir de un nodo origen al resto.
  Dijkstra inverso devuelve un vector de costes de ir del resto a un nodo destino.

- **En un grafo no dirigido, los resultados devueltos por Dijkstra y Dijkstra inverso aplicados al mismo nodo como origen y destino son iguales.**
  Verdadero. Es no dirigido, el coste de ir de A a B es el mismo que de ir de B a A.

- **Ni Dijkstra ni Floyd funcionan correctamente con costes negativos, al contrario que Prim y Kruskall.**
  Verdadero.

- **En un grafo no dirigido, los resultados devueltos por Dijkstra y Dijkstra inverso aplicados al mismo nodo como origen y destino son iguales.**
  Verdadero.

- **Los resultados devueltos por Dijkstra y Dijkstra inverso son idénticos, siendo el grafo no dirigido y tomando el mismo nodo como origen y destino.**
  Verdadero

- **Para calcular los caminos y costes mínimos entre todos los nodos de un grafo, seria una mejora llamar n veces a Dijkstra en vez de una a Floyd, básicamente porque podemos ir reutilizando el vector de devuelve Dijkstra y por tanto ahorrar en espacio. No se hace, sobre todo, por comodidad de llamar solamente a un algoritmo.**
  Falso

- **El parámetro de entrada/salida de Dijkstra es un vector de caminos.**
  Falso

- **Ni Dijkstra, ni Floyd, ni Prim, ni Kruskall funcionan correctamente con costes negativos.**
  Falso

- **Se podría optimizar el algoritmo de Dijkstra usando un APO, en particular a la hora de buscar el nodo mas cercano al origen que todavía no ha sido usado para mejorar a los demás.**
  Vardadero

Floyd

- **Si no pusiéramos la función suma en los algoritmos de Floyd o Dijkstra, en el fondo no pasaría nada, pero aporta legibilidad al código, y eso es muy importante.**
  Falso. Se estarían sumando los costes de infinito, contemplándose los caminos inexistentes entre nodos, y dando por hecho que el grafo es fuertemente conexo.

- **El parámetro de entrada/salida de Floyd es una matriz de caminos.**
  Falso.

- **Si no pusiéramos la función suma en los algoritmos de Floyd o Dijkstra, en el fondo no pasaría nada, pero aporta legibilidad al código y eso es muy importante.**
  Falso.

- **Es muy importante escoger un valor de infinito que este fuera del rango de valores de las operaciones aritméticas con costes que va a hacer, por ejemplo, el algoritmo de Floyd o el de Dijkstra. En caso contrario no sirve.**
  Verdadero.

- **No pasaría nada si sumáramos directamente los costes, en lugar de utilizar la función suma, en los algoritmos de Floyd o Dijkstra, pero aporta legibilidad al código y eso es muy importante.**
  Falso

TAD Partición y Compresión de Caminos

- **No ha sido fácil, pero en la última representación del TAD Partición, por fin hemos conseguido al mismo tiempo garantizar que tanto la operación de unión como la de encontrar estén en O(1).**

Falso, la operación de unir si es de orden constante, pero la de encontrar, lo podría llegar a ser a medio/largo plazo, tras realizar muchas veces la técnica de comprensión de caminos y que el árbol tuviera altura 1.

- **No aporta nada la utilización de la compresión de caminos cuando ya estamos usando la compresión por alturas, dado que, al no actualizar la altura, no es posible combinar adecuadamente ambas técnicas.**
  Falso.

- **En la representación por bosque de arboles del TAD Partición, dado que la función encontrar lo que recorre es el camino hasta la raíz, es en el caso peor de orden logarítmico.**
  Falso.
- **No aporta nada la utilización de la compresión de caminos cuando ya estamos usando la unión por tamaño. El hecho de no poder llevar actualizado el tamaño del árbol es un verdadero problema.**
  Falso

- **En el TAD partición, en la representación de bosque de arboles no conviene utilizar al mismo tiempo la unión por altura y la compresión de caminos, ya que la primera hace mucho menos efectiva la segunda.**
  Falso

- **En la representación por bosque de arboles del TAD Partición, dado que la función encontrar lo que recorre es el camino hasta la raíz, en el caso mejor es de orden logarítmico.**
  Falso

- **En la última representación del TAD Partición, hemos conseguido garantizar que tanto la operación de unión como la de encontrar estén en O(1).**
  Falso

Kruskall

- **No es necesario que Kruskall verifique que no se producen ciclos en la solución, pues al pertenecer los nodos unidos por la arista seleccionada al mismo árbol, ello simplemente no es posible.**
  Falso. ¿Si ya está en el mismo árbol, como va a seleccionar esa arista? No puede. Si comprueba que se produzcan ciclos.
  (F por la redacción y por todos aquellos que la dejamos en blanco por no entenderla)

- **Árbol de expansión de coste mínimo solo hay uno, eso sí, arboles de expansión en general (que no sean de coste mínimo), por supuesto hay muchos.**
  Falso. Arboles de expansión de coste mínimo puede haber más de uno.

- **Es necesario que Kruskall verifique que no se producen ciclos en la solución, lo cual queda garantizado al seleccionar la arista de menor coste cuyos nodos pertenecen a diferentes árboles.**
  Verdadero.

- **Árbol de expansión de coste mínimo solo hay uno, eso si, arboles de expansión en general (que no sean de coste mínimo), por supuesto hay muchos.**
  Falso.

- **Es necesario que kruskall verifique que no se producen ciclos en la solución. Al unir la arista nodos de diferentes arboles se garantiza así que no puede suceder.**
  Verdadero

- **En caso de empate entre dos aristas candidatas, es decir, del mismo coste y ambas validas, el algoritmo de Kruskall debe escoger la que consiga unir más nodos.**
  Falso

- **Árbol de expansión de coste mínimo solo hay uno, eso sí, arboles de expansión en general (que no sean de coste mínimo), por supuesto hay muchos.**
  Falso

Prim

- **Prim y Kruskall no devuelven el mismo resultado siempre, excepto en el caso que haya empates entre diferentes arboles generadores mínimos (arboles cuyo coste mínimo es el mismo).**
  Falso. Si no hay empates también pueden devolver el mismo resultado. Revisar, min7

- **En Prin, en caso de empates entre dos aristas candidatas, es decir del mismo coste y ambas validas, debe escogerse la que consiga unir más nodos.**
  Falso. No tiene un criterio de elección, coge la primera valida que encuentra.

- **Prim y Kruskall devuelven el mismo resultado, excepto posiblemente en el caso que haya empates entre diferentes arboles generadores mínimos.**
  Falso.

- **Hemos utilizado un APO en la implementación de Prim y Kruskall para seleccionar la arista candidata de menor coste en un tiempo O(1).**
  Verdadero.

- **En Prim, en caso de empate entre dos aristas candidatas, es decir del mismo coste y ambas validas, debe escogerse la que consiga unir más nodos.**
  Falso.

- **Es muy importante escoger un valor de infinito que este fuera del rango de las operaciones aritméticas con costes a realizar en los algoritmos de Prim y Kruskall.**
  Falso

- **Independientemente de la implementación concreta que hagamos, es obvio que Prim y Kruskall devuelven el mismo resultado siempre.**
  Falso

- **Hemos utilizado un APO en la implementación de Prim y Dijkstra ya que nos es necesario tener todas las aristas del grafo de entrada ordenadas por costes de menor a mayor.**
  Falso