PREGUNTAS DE TEORÍA (EXAMEN JUNIO)

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

Falso, hay que marcar siempre los nodos visitados, pues no sabemos si el grafo que recibimos será acíclico o no.

En revisión con mayte es verdadera.

👍**2.- En el TAD Partición, en la representación de bosques de árboles, no conviene utilizar al mismo tiempo la unión por altura y la compresión de caminos**

Falso, esta técnica permite mejorar la eficiencia aún más. El fundamento de esta idea es que todos los nodos por los que pasamos durante una búsqueda se transformen en hijos de la raíz del árbol al que pertenecen, así disminuimos los tiempos de búsqueda

👍**3.- Las operaciones del TAD árbol binario permiten insertar y eliminar hojas y nodos internos**

Falso, la operación eliminar() no permite eliminar nodos internos, sólo podemos eliminar nodos hoja.

👍**4.- Al insertar en un Árbol B, si el nuevo elemento no cabe en el nodo que le correspondería, se divide el nodo en dos y se promociona un elemento al nodo padre, y en este caso, se permite que exista algún nodo con un solo elemento, independientemente del mínimo permitido según el orden del árbol.**

Falso, nunca puede quedar con menos claves que el mínimo permitido

👍**5.- Si implementamos un TAD Conjunto mediante un ABB, podríamos garantizar que la operación de pertenencia de un elemento a un conjunto siempre sea de coste logarítmico**

Falso, un ABB no implica que el tiempo de búsqueda sea de O(log n), pues éste depende de la ramificación del árbol.

👍**6.- 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, hemos conseguido que estén en O(1), ya que la unión de árboles es de dicho orden, pero la operación de encontrar es de O(log n).

👍**7.- El cálculo de la altura de un árbol es de O(n), excepto para los AVL, en cuyo caso el orden del cálculo de la altura es logarítmico**

Verdadero, porque al ser semicompleto sabes que accediendo siempre al hijo izquierdo vamos a llegar siempre a la profundidad máxima.

👍**8.- Es cierto que todos los AVL son ABB, pero no es cierto que algunos ABB no sean AVL**

Falso, la segunda afirmación es incorrecta.

👍**9.- 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, es necesario que Kruskall verifique que no se producen ciclos en la solución, esto se hace al comprobar que las dos aristas escogidas pertenezcan a diferentes árboles.

¿?👍**10.- 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, la suma de un elemento con infinito es infinito, y esta operación no podemos hacerla con solamente el operador '+'

👍**11.- Sea un árbol binario implementado mediante una representación vectorial. La destrucción del árbol binario debe eliminar los nodos uno a uno es postorden**

Falso, se eliminan todos los nodos sin importar el recorrido

👍**12.- 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**.

Verdadero, si ya que el algoritmo de Dijkstra es O(n^2), en cambio Floyd es O(n^3). Luego si tenemos definido un origen y destino, nos es más eficiente utilizar el algoritmo de Dijkstra.

(No responder)13.- En el algoritmo de eliminación de una clave en un árbol B, si dicha clave no está en una hoja, se sustituye por su sucesor más inmediato, que se encontrará siempre en el hijo derecho del nodo que contiene la clave a eliminar

Falso, dependerá del orden lineal de las claves

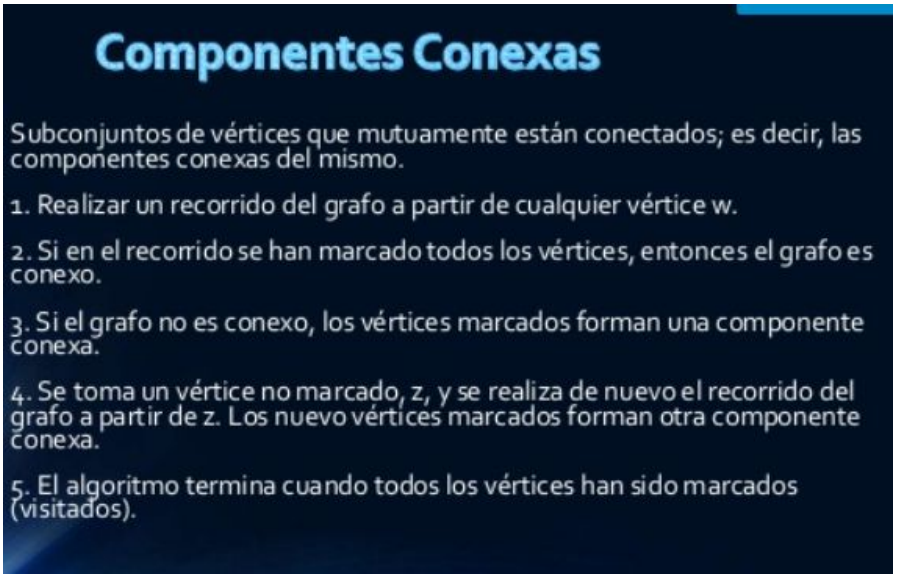
Verdad

👍**14.- La inserción en el mismo orden de un conjunto de elementos en un ABB y un AVL daría como resultado árboles de la misma altura**

Falso, la inserción en un avl siempre se hace de tal manera de que quede equilibrado el árbol. En un ABB no hay equilibrio.

👍**15.- Un recorrido en anchura desde un vértice origen permite encontrar el conjunto de vértices de un componente conexo**

Verdadero



👍**16.- En un subárbol de un árbol cualquiera se cumple la siguiente propiedad: en cada nodo la suma de su altura y profundidad es constante y coincide con la profundidad de la hoja más profunda**

Falso, no se cumple para cada nodo, pero si que se cumple con los nodos que se encuentren en la misma rama de la hoja más profundo del árbol

👍**17.- Supongamos un ABB con el elemento x en una hoja cuyo padre tiene el valor y. Entonces, y es el menor elemento mayor que x o bien, y es el mayor elemento menor que x.**

Verdadero

👍**18.- Los elementos de un APO se obtienen en orden mediante la extracción sucesiva de estos.**

Verdadero, en la cima siempre encontramos el elemento menor. Cuando lo eliminamos, se inserta el siguiente elemento menor.

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

Falso, debe escogerse cualquiera de las dos(en el algoritmo coge el último de los costes). El algoritmo no verifica cual une más nodos.

👍**20.- Ni Dijkstra ni Floyd funcionan correctamente con costes negativos, al contrario que Prim y Kruskal.**

Verdadero

👍**21.- Prim y Kruskall no devuelven el mismo resultado siempre, excepto en el caso que haya empates entre diferentes árboles generadores mínimos.**

Falso, Prim y Kruskall devuelven el mismo resultado siempre, excepto en el caso que haya empates, que pueden devolver diferentes árboles generadores.(La pregunta está planteada muy hijo de puta)

👍**22.- La propiedad de equilibrio de un AVL permite encontrar un elemento en un tiempo de O(log n) en el caso peor**

Verdadero

¿?23.- **La representación vectorial de posiciones relativas es adecuada solamente para árboles parcialmente ordenados**

Falso

Información encontrada: la representación vectorial de posiciones relativas es adecuada para árboles completos, es decir, se dice que un árbol binario de altura que está completo si está lleno hasta altura k-1 y el último nivel está ocupado de izquierda a derecha

👍**24.- Para conseguir que la anchura del árbol B sea menor, me interesa crear nodos con el mayor tamaño posible.**

Verdadero, de esta manera conseguimos que el árbol no se divida en más nodos

👍**25.- En la implementación vectorial de árbol binario, que tiene como invariante que se colocan todos los elementos al principio del vector, la inserción es de coste O(1), pero el borrado de coste O(n)**

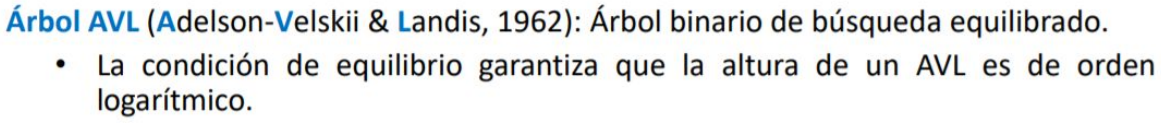
Falso, se inserta al final del vector y la eliminación es de coste O(1)

👍**26.- Un AVL es ABB y el recíproco es cierto.**

Falso, un ABB no es un AVL porque puede estar NO equilibrado

👍**27.- La propiedad de equilibrio de un AVL no implica que su altura sea la mínima posible**

Falso, la condición de equilibrio garantiza que la altura de un AVL es de orden logarítmico.



http://es.tldp.org/Tutoriales/doc-programacion-arboles-avl/avl-trees.pdf

👍**28.- En la representación por bosque de árboles 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**

Verdadero.

👍**29.- Todo APO min-max cumple estrictamente con las condiciones que hemos definido para un APO, pero el recíproco no es cierto**

Verdadero, no cumple TODAS las condiciones que hemos definido

👍**30.- La eficiencia espacial de la representación de un árbol binario mediante un vector de posiciones relativas será mayor cuantos más nodos falten en el nivel inferior**

Falso, la eficiencia espacial será menor cuantos menos nodos falten en el último nivel.

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

Verdadero, es trivial.

👍**32.- 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**

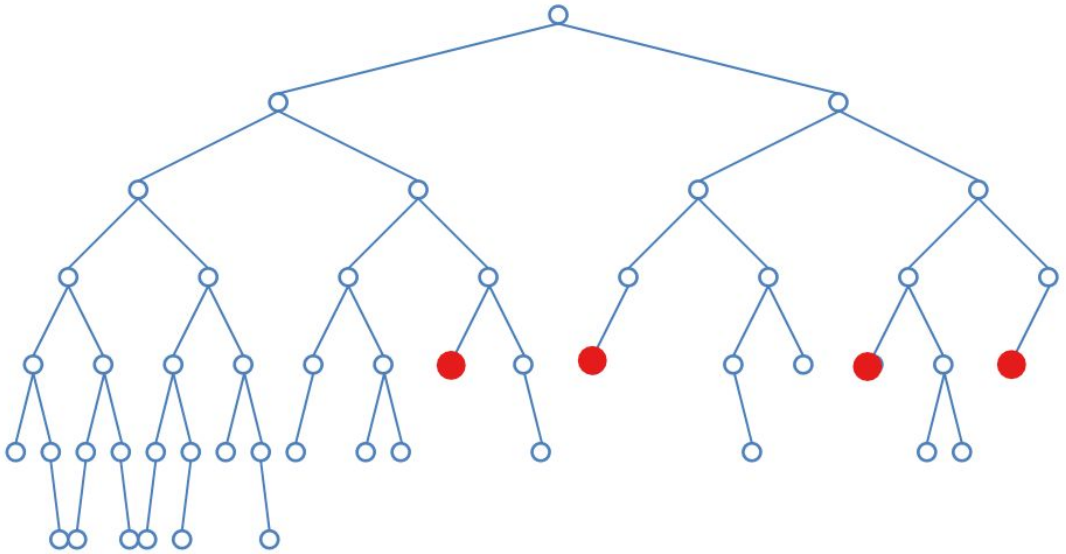
Falso, podemos determinar diferentes árboles de expansión que no sean de coste mínimo, pues pueden haber varios caminos que unan todos los vértices sin provocar ciclos.

👍**33.- 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**

Falso, carecería de sentido usar un APO en el algoritmo.

👍**34.- En un AVL cada nodo tiene un factor de equilibrio de 1 0 o -1 y ello significa que todas las hojas se van a encontrar en el último nivel o en el penúltimo**

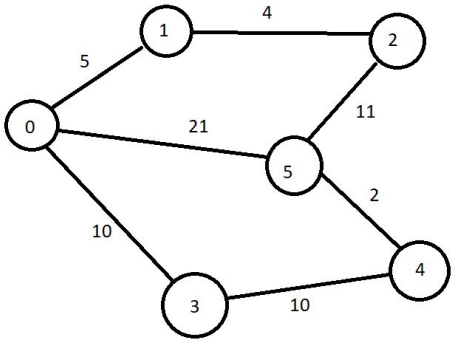
Falso, los nodos marcados en rojo no se encuentran en el último ni en el penúltimo nivel.



Es AVL

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

Verdadero




```
GrafoP<int> G(6);
vector<typename GrafoP<int>::vertice> P;
G[0][1] = 5;
G[1][0] = 5;
G[0][3] = 10;
G[3][0] = 10;
G[0][5] = 21;
G[5][0] = 21;
G[1][2] = 4;
G[2][1] = 4;
G[2][5] = 11;
G[5][2] = 11;
G[3][4] = 10;
G[4][3] = 10;
G[4][5] = 2;
G[5][4] = 2;

vector<tCoste> origen1 = Dijkstra(G,0,P);
vector<tCoste> origen2 = DijkstraInv(G,0,P);

for(int i = 0; i <= origen1.size()-1; i++)
{
    cout<<origen1[i]<<" ";
}
cout<<endl;
for(int i = 0; i <= origen2.size()-1; i++)
{
    cout<<origen2[i]<<" ";
}
```

```
0 5 9 10 20 20
0 5 9 10 20 20
--------------------------------
Process exited after 0.2108 seconds with return value 0
Presione una tecla para continuar . . .
```

👍36.- La propiedad de orden parcial de un APO implica que siempre va a estar equilibrado

Falso. La propiedad de orden parcial implica que siempre va a estar parcialmente ordenado.

👍37.- Una cola con prioridad mediante un APO permite eliminar el elemento prioritario con un coste O(1) en el caso peor

Falso, todas las eliminaciones tienen un coste O(log2 n) en el peor caso.

👍38.- La profundidad del nodo más profundo del árbol es la altura de dicho árbol menos 1

Falso, la profundidad del nodo más profundo coincide con la altura del árbol.

👍39.- A partir de los recorridos en PreOrden e InOrden de un árbol general es posible reconstruir el árbol original, si además se conoce el grado del árbol.

Verdadero, siempre y cuando nos den el recorrido en inorden y otro recorrido podemos reconstruir el árbol original.

👍**40.- La representación del TAD Árbol general mediante listas de hijos es más ineficiente cuanto más alto es el árbol.**

Falso, es más eficiente cuanto mayor sea el grado del árbol.

PREGUNTAS DE TEORÍA (EXAMEN SEP)

👍**1.Es necesario marcar los nodos visitados en el recorrido de un grafo para evitar provocar ciclos en el mismo.**
Verdadero

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

👍**3.La propiedad de búsqueda de un ABB permite encontrar un elemento en un tiempo de O (log n) en el caso peor.**
Falso. // Log n es en el caso promedio en el caso peor es O(n)

👍**4-No aporta nada la utilización de la compresión de caminos cuando ya estamos usando la unión por altura, no es posible combinar adecuadamente ambas técnicas.**
Falso , ya que si aporta está en la dispositiva 15 de kruskall.

👍**5-La eliminación de elemento en un ABB puede llegar a tener un coste O(n). Verdadera.**
Verdadero.Si que puede cuando esta degenerado en una lista, cuando esta equilibrado la op eliminar nunca será mayor que O (Log n).

👍**6-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, escoge cualquiera de las dos

👍**7-El recorrido en preorden de un ABB determina unívocamente el ABB del que procede.**
Falso

👍**8-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, ya que en el ejemplo de Floyd y Dijkstra los valores negativos no son válidos, también hay que tener en cuenta el rango de valores que se van a tomar con las operaciones aritméticas, por lo tanto el único valor que nos queda posible es el INFINITO.

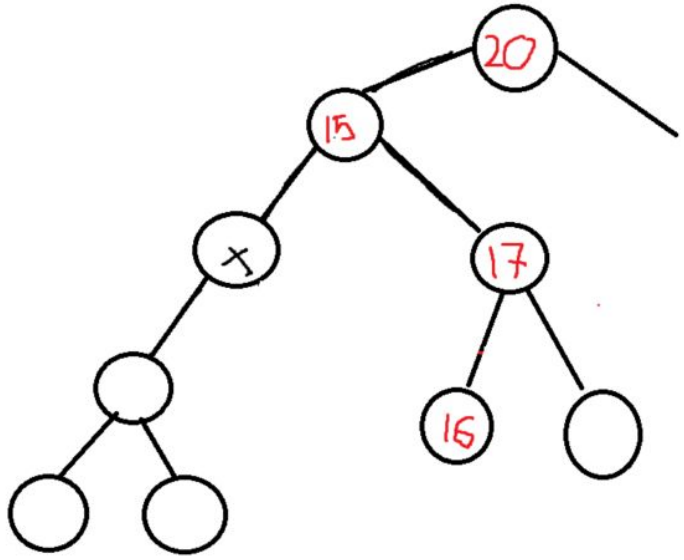
👍**9-Árbol de expansión de coste mínimo solo hay uno, eso sí, arboles de expansión en general (que son sean de coste mínimo), por supuesto hay muchos.**
Falso.

👍**10-Es cierto que todos los AVL son ABB, pero el reciproco no es cierto.**
Verdadera.

👍**11-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, ya que la eliminación es de tiempo constante y el APO hace que este ordenado.

👍**12-Sea x un elemento de un ABB. El sucesor de x se encuentra ascendiendo hacia la raíz hasta encontrar un nodo (que puede ser el propio x) con hijo derecho y a continuación buscando el menor elemento en el subárbol derecho.**

Falso



👍**13-La eficiencia espacial de la representación de un árbol binario mediante un vector de posiciones relativas será mayor cuantos menos nodos falten en el nivel inferior.**
Verdadero (Corregida).

👍**14-Sea un árbol binario implementado mediante una representación vectorial. La destrucción del árbol binario no debe eliminar los nodos uno a uno en postorden.**
Verdadero

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

👍**16-La representación del TAD Árbol general mediante listas de hijos es mas ineficiente cuanto mas alto es el árbol.**
Falso, cuanto mayor sea el grado del árbol

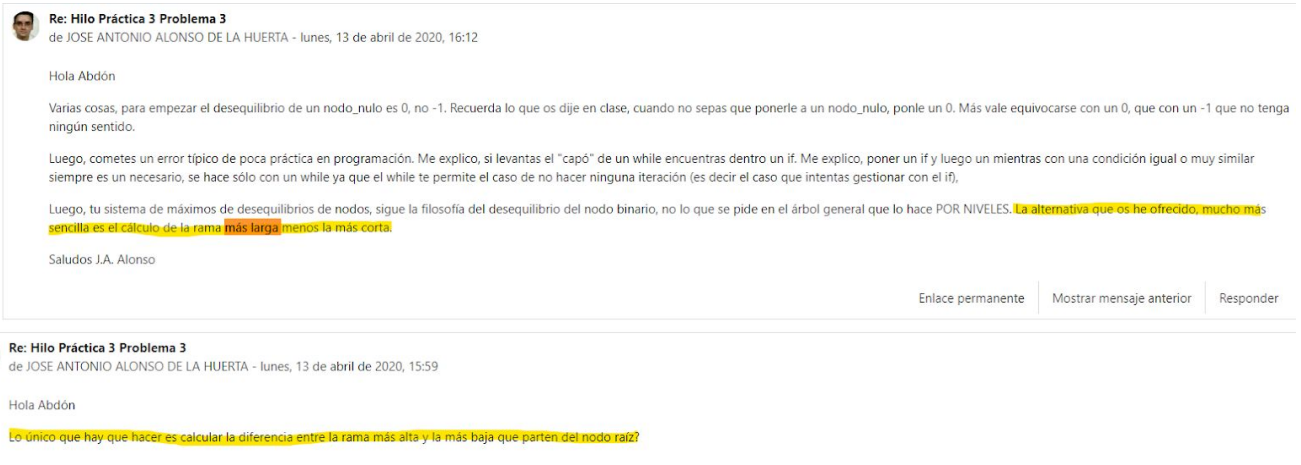
👍**17-Los nuevos elementos de un árbol B se insertan en las hojas y , si es necesario, se reorganiza el árbol.**
Verdadero

👍**18- En un árbol B de orden m, todos los nodos contienen un mínimo de m-1/2 (es acotación inferior) claves y un máximo de m-1.**
Falso ya que la raíz no lo tiene.

👍**19-La condición de equilibrio no perfecto de un AVL, no asegura que la inserción de un elemento se pueda hacer a un coste de O (log n) en el peor caso.**
Falso. Si lo asegura

👍**20-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

👍**21-Se define el desequilibrio de un árbol general como la máxima diferencia entre las alturas de los subárboles mas bajo y mas alto de cada nivel. Esta definición y la diferencia de longitudes entre la rama mas larga y mas corta de dicho árbol son equivalentes.**
Falso, La primera afirmación verdadera, aunque la segunda es falsa.



Re: Hilo Práctica 3 Problema 3
de JOSE ANTONIO ALONSO DE LA HUERTA - lunes, 13 de abril de 2020, 16:12

Hola Abdón

Varias cosas, para empezar el desequilibrio de un nodo_nulo es 0, no -1. Recuerda lo que os dije en clase, cuando no sepas que ponerle a un nodo_nulo, ponle un 0. Más vale equivocarse con un 0, que con un -1 que no tenga ningún sentido.

Luego, cometes un error típico de poca práctica en programación. Me explico, si levantas el "capó" de un while encuentras dentro un if. Me explico, poner un if y luego un mientras con una condición igual o muy similar siempre es un necesario, se hace sólo con un while ya que el while te permite el caso de no hacer ninguna iteración (es decir el caso que intentas gestionar con el if),

Luego, tu sistema de máximos de desequilibrios de nodos, sigue la filosofía del desequilibrio del nodo binario, no lo que se pide en el árbol general que lo hace POR NIVELES. La alternativa que os he ofrecido, mucho más sencilla es el cálculo de la rama más larga menos la más corta.

Saludos J.A. Alonso

Enlace permanente | Mostrar mensaje anterior | Responder

Re: Hilo Práctica 3 Problema 3
de JOSE ANTONIO ALONSO DE LA HUERTA - lunes, 13 de abril de 2020, 15:59

Hola Abdón

Lo único que hay que hacer es calcular la diferencia entre la rama más alta y la más baja que parten del nodo raíz?

👍**22-Un árbol completo se puede almacenar muy eficientemente en un vector de posiciones relativas.**
Verdadero.

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

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

👍**25-El parámetro de entrada/salida de Floyd es una matriz de caminos.**
Falso. Es una matriz de costes o de vértices, pero no caminos.

👍**26-La inspección en el mismo orden de un conjunto de elementos en un ABB y un AVL daría como resultado árboles de la misma altura.**
Falso

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

👍**28-En un AVL cada nodo tiene un factor de equilibrio de 1,0 ó -1 y ello significa que todas las hojas se van a encontrar en el ultimo nivel o en el penúltimo.**
Falso

👍**29-Si un árbol es un APO, tiene un desequilibrio en valor absoluto menor o igual que 1, pero el reciproco no es cierto. reciproco ->(Un arbol equilibrado es un apo).**
Verdadero

👍**30-En un subárbol de un árbol cualquiera, tal que todas las hojas del subárbol están en el mismo nivel, se cumple la siguiente propiedad: en cada nodo la suma de su altura y profundidad es constante y coincide con la profundidad de sus hojas.**
Verdadero. Todas las hojas están al mismo nivel

👍**31-Dados los recorridos en preorden y postorden de un árbol binario se puede reconstruir unívocamente en árbol original.**
Falso, necesitamos siempre el inorden y después uno de los otros dos o preorden o postorden.

32-Al insertar un Árbol B, si el nuevo elemento no cabe en el nodo que le correspondería, se le divida el nodo en dos y se promociona la mediana al nodo padre, y en el caso de que el árbol conserve su altura, se permite que existan nodos con menos elementos de la mitad (por defecto) de la capacidad de un nodo.
Verdadero.

👍**33-La propiedad de completitud de un APO implica que siempre va a estar equilibrado.**
Verdadero

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

👍**35-Una cola con prioridad representada mediante un APO permite inserciones con un coste en O (Log n) en el caso peor.**
Verdadera, está en APO pagina 3.

👍**36-En la representación por bosque de árboles del TAD Partición, dado que la función encontrar lo que recorre es el camino hasta la raíz, es en el peor caso peor de orden logarítmico.**
Falso es orden n, con control de altura es O(Log n)

👍**37-El recorrido en anchura de un APO proporciona el acceso en orden a los elementos almacenados.**
Falso, que un APO será un árbol ordenado no quiere decir que estos se obtengan en orden por anchura.

👍**38-La función ContarNodos es normalmente de coste n, excepto que sepamos que el árbol binario tiene como máximo en valor absoluto un desequilibrio de 1, en cuyo caso su orden es logarítmico.**
Falso, hay que contar todos los nodos si o si.

👍**39-Todo un APO min-max cumple estrictamente las condiciones que hemos definido para un APO, y el reciproco también es cierto.**
Falso, un apo no cumple las condiciones del APO min-max

👍**40-La extracción sucesiva de los elementos de un APO no tiene por qué devolverlos en orden.**
Falso, Los devuelve en orden.

PREGUNTAS DE TEORÍA (FORO)

PREGUNTA: ¿Funciona Kruskal y Prim con Grafos dirigidos?
RESPUESTA: No, solo funciona con grafos no dirigidos es una precondición del grafo que le llega a Prim o Kruskal.

PREGUNTA: ¿Prim sería de O(n^2)?
RESPUESTA: No, es de O(log n * n^2). Ten en cuenta que la inserción en un APO es de log n.

PREGUNTA: ¿Por qué no implementamos el TAD partición como una plantilla?
RESPUESTA: Porque no tiene sentido. El tCoste es un tipo muy concreto, no puede ser cualquier tipo que se nos ocurra ni mucho menos. Si los elementos son de un tipo genérico T, no se pueden usar como índices de los vectores.

PREGUNTA: ¿Porque los nodos los estamos representando con enteros?
RESPUESTA: Por comodidad y manejo de índices de vectores y matrices, recorrer un grupo de elementos que se llaman Madrid, Soria, es mucho más difícil que 0,1,2...

PREGUNTA: Dado que las búsquedas en listas son de orden n, las listas de listas de adyacencia son ineficientes, no solo para buscar a partir de un vértice v si tiene una arista con otro vértice w, sino para además encontrar ese vértice v. ¿O me equivoco?
RESPUESTA: Sí, te equivocas, para buscar ese vértice se hace en el vector, y es de orden 1. Por cierto, ¿qué buscas exactamente? Ya sabes que el nodo existe, ¿no?.

PREGUNTA: Tras hacer un número gigantesco de llamadas a encontrar del TAD partición implementado con los bosques de árboles, unión por altura y encontrar con compresión de caminos, el coste de encontrar se reduciría a casi de orden constante. ¿Me he equivocado en algo?
RESPUESTA: No, y sin un número gigantesco, el mero uso de encontrar irá haciendo que poco a poco, todos los nodos sean hijos del representante, y por tanto sí, acercándose al orden 1.

PREGUNTA: ¿Funcionan correctamente Prim y Kruskal con costes negativos?
RESPUESTA: Sí perfectamente, aunque habría que encontrar una semántica donde tengan sentido los costes negativos.

PREGUNTA: ¿Dado los requisitos que exige el TAD lista, en la lista de adyacencia para grafos ponderados el tipo T debe tener constructor predeterminado no?
RESPUESTA: Sí y además el tipo T debe tener sobrecargado el operador de comparación == (se necesita para las búsquedas).

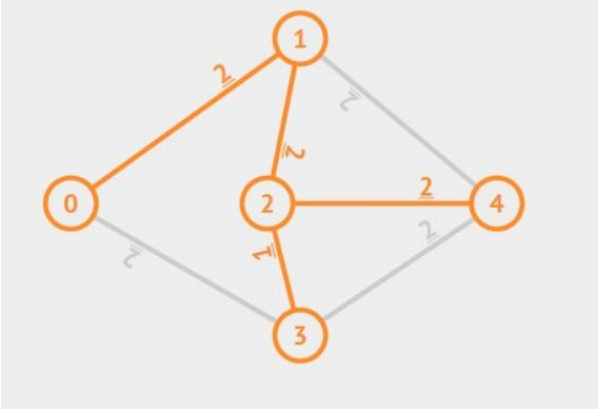
PREGUNTA: ¿La unión por tamaño genera un árbol resultante de altura equivalente a la altura del árbol que tenía mayor número de nodos más uno si los dos árboles a unir tienen el mismo tamaño?
RESPUESTA: Depende de cómo lo implementes. Vamos a suponer que guardas el tamaño de los árboles para unir el pequeño al grande, en ese caso el árbol resultante tiene como altura el mayor de estos dos números, la altura del pequeño + 1 (ya que quedaría como hijo), o la del árbol grande (que ya tenía esa altura)

PREGUNTA: ¿Podría utilizarse un APO para mejorar el Algoritmo de Dijkstra?
RESPUESTA: No ya que al mejorar un nodo habría que eliminarlo, lo que sería un problema ya que el APO solo nos permite eliminar el más pequeño.

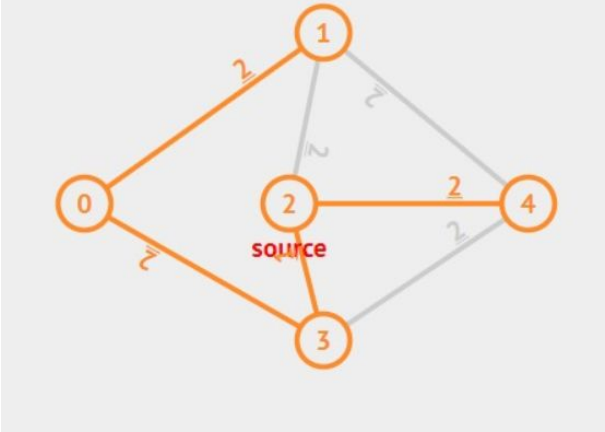
PREGUNTA: ¿Existen diferentes árboles de coste mínimo?
RESPUESTA: Sí, la solución al problema del árbol de expansión mínimo no tiene por qué ser única.

PREGUNTA: ¿Puede Prim y Kruskal dar soluciones diferentes?
RESPUESTA: Sí, pueden dar soluciones diferentes siendo ambas correctas

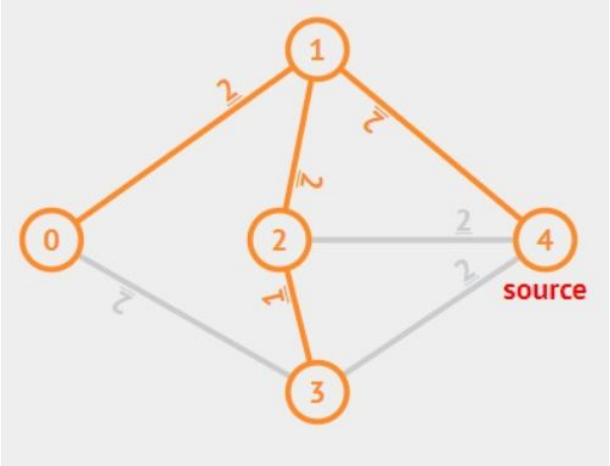
Árbol generador con Kruskal:



Árbol generador con Prim empezando por la arista 2:



Árbol generador con Prim empezando por la arista 4:



PREGUNTA: ¿Usamos un APO para tener las aristas ordenadas en Kruskal?
RESPUESTA: No, usamos el APO para tener un acceso rápido y eficiente a la arista más pequeña o de coste más pequeño.

PREGUNTA: ¿Realmente es más eficiente hacer Floyd que n veces Dijkstra?
RESPUESTA: la respuesta es depende. Depende de la densidad del grafo y también de la representación del mismo. En los algoritmos de las diapositivas los grafos están representados con matriz de costes. En un ejercicio de las prácticas os pedimos implementar Dijkstra con listas de adyacencia. Pues analizando esa implementación de Dijkstra y suponiendo que el grafo es muy poco denso (es decir, el número de aristas es mucho menor que el cuadrado de los vértices (n^2), se puede llegar a la conclusión (tiene su dificultad) de que usar n veces Dijkstra es más eficiente que Floyd.

PREGUNTA: ¿por qué el algoritmo de Floyd es tan sencillo y el de Dijkstra más complejo? En Dijkstra antes de recalcular los costes, el proceso de elegir el vértice que vamos a usar para mejorar es más minucioso y detallado, mientras que Floyd simplemente usa TODOS los nodos, da igual el coste del camino, si se ha usado ya o no, etc.
RESPUESTA: Son dos algoritmos en los que se usan técnicas de programación totalmente diferentes. El de Dijkstra es uno de los que se llaman algoritmos devoradores (de forma muy simple, pero no precisa, se basan en la idea de optimizar localmente para alcanzar el óptimo global); mientras que el de Floyd es un algoritmo de programación dinámica (otra vez con simplicidad, pero sin precisión, se basan en ir resolviendo un conjunto de subproblemas, inicialmente muy pequeños, cada vez más grandes, hasta resolver el problema). El de Floyd es más sencillo, pero la aplicación de la técnica de programación para llegar a él es más compleja.

PREGUNTA: ¿Cuenta la arista que de un vértice a sí mismo para aumentar el grado del vértice?
RESPUESTA: Sí, cuenta para el cálculo del grado de dicho nodo.

PREGUNTA: ¿Al finalizar el algoritmo de Kruskal implementado con el TAD partición implementado con bosque de árboles, el árbol que queda en la partición que usa no es el mismo que el árbol que devuelve verdad?
RESPUESTA: Por un lado sería irrelevante ya que es una variable local que desaparece, pero no, el árbol que queda no tiene nada que ver con la solución.

PREGUNTA: ¿Hay una forma de verificar que la operación unir del TAD partición nos va a dar como resultado un árbol igual que el que devuelve Kruskal (realmente la implementación dada devuelve un grafo pero todos sabemos que un árbol es un grafo conexo acíclico)?
RESPUESTA: No, no hay forma de verificar algo que ni es cierto, ni tiene por qué serlo.

PREGUNTA: ¿En un árbol general representado mediante listas de hijos, aunque la búsqueda en esas lista sea de orden n, la inserción y la eliminación de hijo izqdo no son de orden n verdad?
RESPUESTA: Correcto. El motivo es obvio, el hijo izquierdo es el primer elemento de la lista, luego encontrarlo es coste 1 y no depende del tamaño de dicha lista.

PREGUNTA: Las únicas operaciones de coste n de esta representación serían la de insertarHijoIzqdo e insertarhijoDrcho, pero porque hay que buscar la primera celda libre y eso se hace con un for tal y como vimos la implementación de dicha representación ¿Correcto?
RESPUESTA: De nuevo correcto.

PREGUNTA: ¿El orden de inserción de elementos en un APO influye en el rendimiento de las operaciones de insertar y eliminar no? Porque si vas, por ejemplo, insertando siempre elementos de manera que no haya que flotar el nodo con el elemento insertado o eliminando elementos de manera que no haya que hundir el nodo raíz entonces las operaciones de inserción y eliminación son casi de orden 1.
RESPUESTA: Sí, pero aquí las palabras clave son "casi" y "rendimiento". Puedes tener más o menos suerte y tener que flotar un poco más o un poco menos, pero el orden de esa operación es logarítmico.

PREGUNTA: ¿La condición que debe tener un ABB para que la búsqueda sea de un coste menor que n es que su desequilibrio no sea tal que el árbol sea una lista degenerada? ¿Esa situación es la única en la que un ABB tiene una búsqueda de coste n?
RESPUESTA: Bueno, todo es una cuestión de grados. La inserción en un ABB en EL CASO PEOR, que es como tú planteas cuando el árbol es en realidad una lista. En este caso el árbol degenerado sería un lista, no es cuestión de que la lista esté degenerada. En ese CASO PEOR es n. En el caso mejor, cuando el nivel de equilibrio es razonable

(típicamente la condición que representa esto es ser un AVL) el orden es logarítmico. Y luego claro, en medio de ambos casos, el orden va empeorando desde logarítmico hasta n, en función del aumento del desequilibrio del árbol, digamos que es algo gradual.

PREGUNTA: Por qué para construir un Abin de forma única necesitamos conocer su inorden y luego o su preorden o su postorden? ¿No valdría solo con el inorden?
RESPUESTA: No, para identificar totalmente un árbol necesitas dos recorridos, no uno.

PREGUNTA: ¿por qué para construir un Abin de forma única necesitamos conocer su inorden y luego su preorden o su postorden?
RESPUESTA: El Inorden nos sirve para saber delimitar qué forma parte del subárbol izquierdo y qué forma parte del subárbol derecho, pero para poder delimitarlo, necesito saber quién es el nodo raíz, y ese es un dato que solo se puede sacar del preorden o el postorden.

PREGUNTA: ¿No se podría mejorar un Agen representado por lista de hijos?
RESPUESTA: Representación del TAD Árbol general mediante listas de hijos.
El coste temporal de añadir y quitar nodos del árbol proviene de dos factores a considerar: a) coste de añadir y quitar un nodo en el vector de nodos; b) coste de añadir y quitar un nodo en la lista de hijos de su padre.

Si nos fijamos en a), tenemos que distinguir cómo gestionamos el vector de nodos, para lo cual hemos discutido en clase dos estrategias: 1) marcamos las celdas libres del vector; 2) agrupamos las celdas ocupadas al principio del vector. Con 1) la inserción de cualquier nodo (hijo izquierdo o hermano derecho) es de O(n); por contra la eliminación es O(1). Con 2) tenemos la situación inversa, la inserción es O(1) y la eliminación O(n).

¿Por qué en este último caso la eliminación es O(n)? Porque no basta con mover el último nodo del vector a la posición liberada, sino que hay que buscar dicho nodo en la lista de hijos del padre para actualizar su posición y esa búsqueda es O(n) en el peor caso —en general, O(g), siendo g el grado del árbol, por tanto el peor caso se dará cuando g = n-1, es decir, para un árbol de altura 1.

Analicemos el factor b). Ahora tenemos que considerar si la operación es con el hijo izquierdo o con el hermano derecho. Insertar o eliminar el hijo izquierdo estará en O(1), pues habrá que insertar o eliminar en la primera posición de la lista de hijos. Sin embargo, al insertar o eliminar el hermano derecho hay que buscar en la lista de hijos del padre, lo cual implica que estás operaciones sean O(n) en el peor caso — otra vez y

en general, O(g), siendo g el grado del árbol, por tanto el peor caso se dará cuando g = n-1, es decir, para un árbol de altura 1.

En resumen:

- Insertar un hijo izquierdo está en O(1) si usamos la opción 2) para gestionar el vector de nodos, de lo contrario está en O(n).
- Eliminar un hijo izquierdo está en O(1) si usamos la opción 1) para gestionar el vector de nodos, de lo contrario está en O(n).
- Insertar y eliminar un hermano derecho están en O(n).

PREGUNTA: Si veo un árbol binario, que cumpla ambas condiciones en el momento de la imagen ¿Puedo identificar en ese momento a ese árbol como un APO? ¿O para que un árbol sea un APO debe serlo desde su creación? Nota: esta misma pregunta podría extenderse también a Abb o AVL, que cumplen una serie de condiciones.
RESPUESTA: Yo diría que no. Sigue siendo un árbol binario que momentáneamente ha adoptado la forma de APO, ABB o AVL y posiblemente la pierda más adelante.

PREGUNTA: ¿qué aplicación práctica podría tener un APO min-max?
RESPUESTA: Son útiles cuando necesites acceder en coste O(1) a los dos extremos a la vez, al máximo y al mínimo.