**Arboles binarios**

1. **¿Por qué surgen los árboles binarios?**

Porque se necesita una estructura de datos que no esté organizada de manera secuencial, sino jerárquicamente, para así poder plasmar situaciones no representables con otros tipos de estructuras lineales; además de para obtener búsquedas de órdenes logarítmicos.

1. **¿Por qué la operación de insertar y eliminar es de O (1) en la representación vectorial o de celdas enlazadas de árboles binarios?**

Para insertar, porque en la vectorial siempre añadimos en la última posición del vector, que está vacía y el acceso al vector es de orden O (1). En la de celdas enlazadas, simplemente tenemos que apuntar al puntero que tiene el padre al nuevo nodo creado, lo cual también es de Orden O (1).

La operación eliminar es de O (1) en la representación de celdas enlazadas porque la acción que lleva a cabo es la de liberar memoria y apuntar a <<null>> el puntero al nodo eliminado. En la representación vectorial, la operación eliminar es de O (1) ya que el apuntador al nodo eliminado se queda en nodo nulo, y el nodo que estaba en la última posición mueve a donde estaba el nodo eliminado, como en estas operaciones no tenemos que buscar nada, la operación se hace en coste 1 aunque sean varias.

1. **¿Las operaciones insertar y eliminar del TAD Binario representado mediante un vector son de O (1)?**

Para insertar, porque en la vectorial siempre añadimos en la última posición del vector que esté vacía y el acceso al vector es de orden O (1). En la de celdas enlazadas, simplemente tenemos que apuntar el puntero que tiene el padre al nuevo nodo creado, lo cual también es de orden O (1).

La operación de eliminar es de O (1) en la representación de celdas enlazadas porque la acción que lleva a cabo es la de liberar memoria y apuntar a <<null>> el puntero al nodo eliminado. En la representación vectorial, la operación eliminar es de O (1) porque el apuntador al nodo eliminado se queda en nodo nulo, y el nodo que estaba en la última posición mueve a donde estaba el nodo eliminado. Como en estas operaciones no tenemos que buscar nada, la operación se hace en coste constante.

1. **¿La eliminación de nodos de un árbol binario se puede conseguir con O (1), cuando se utiliza una representación vectorial con índice al padre, hijo izquierdo e hijo derecho?**

En una representación vectorial, la eliminación de nodos es de orden constante.

1. **Ventaja de la representación de árboles binarios con celdas enlazadas frente a matrices**

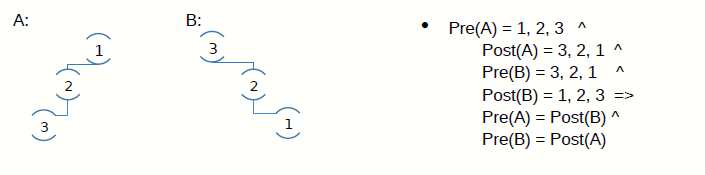
La principal ventaja es que la representación con celdas enlazadas es en memoria dinámica, mientras que con matrices es pseudoestática. Con las celdas enlazadas ocupamos el espacio que necesitamos; en la de matrices ocupamos siempre el espacio máximo del árbol, el cual debemos establecer a priori para esta representación.

1. **¿Puede construirse de forma única un árbol binario dado, conociendo su preorden y el peso de cada nodo? (Número de nodos descendientes suyos).**

No, para poder construir un árbol binario de forma unívoca, necesitamos conocer su inorden y su preorden o postorden. A partir del recorrido en preorden o postorden obtenemos el nodo raíz y, gracias al recorrido inorden, sabemos que nodos pertenecen al subárbol izquierdo y quienes al subárbol derecho de dicho nodo y así recursivamente con los subárboles.

1. **Sean A y B dos árboles binarios diferentes, ¿pueden ocurrir simultáneamente que: ¿Pre(A)=Post(B) y Pre(B)=Post(A)?**

Si, cuando los dos árboles binarios están vacíos, cuando solo tienen la raíz y cuando solo tiene la raíz, un hijo y el árbol sigue este esquema:



1. **¿En qué situaciones es conveniente utilizar un vector de posiciones relativas?**

Cuando no tengamos un apuntador al padre y a los hijos de cada nodo y así poder calcular la posición de los mismos, lo que nos permitiría modificar, eliminar e insertar un nodo en O (1).

1. **A partir de dos recorridos cualesquiera conocidos, ¿Podemos conocer el árbol?**

No, para conocer el árbol necesitamos dos recorridos, pero uno de ellos debe ser necesariamente inorden. A partir del recorrido en preorden o postorden obtenemos el nodo raíz y gracias al recorrido inorden sabemos quiénes pertenecen al subárbol izquierdo y quienes, al subárbol derecho de dicho nodo, y así recursivamente con los subárboles.

1. **¿Cuántos tipos de recorridos de árboles en anchura existen? Explica sus diferencias**

Solo existe un recorrido en anchura, que es el recorrido por niveles (se procesan todos los nodos de izquierda a derecha antes de pasar a otro nivel).

1. **¿Qué condición tienen que cumplir los elementos de un árbol para poder realizar las búsquedas con un coste menor que O(n)?**

Deben tener una relación de orden y su disposición en el árbol debe ser la adecuada para que dicho árbol este equilibrado.

**Arboles Generales, ABB, AVL, APO**

1. **¿Por qué no se puede implementar un árbol general con un vector de posiciones relativas?**

Porque no podemos conocer su grado (número máximo de hijos), luego no podemos prever la posición que deberían ocupar sus hijos.

1. **¿Se podrían usar las listas doblemente enlazadas en los árboles generales mediante listas de hijos?**

Sí, ya que una lista doblemente enlazada posee características mejoradas de una lista enlazada, usada en nuestra representación. Por lo anterior, si usamos una lista doblemente enlazada, vamos a poder cumplir con todas las especificaciones del TAD.

1. **¿Existe alguna operación claramente ineficiente en los árboles generales representados mediante listas de hijos?**

La operación constr, ya que tiene que recorrer todo el vector para poner los padres a null, y las de insertar, porque tiene que buscar que la primera posición esté a null, luego las dos en el peor caso serán de O(n).

1. **¿Puede determinarse un árbol general a partir a partir de los recorridos en inorden y postorden?**

No se puede, harían falta los 3 tipos de orden (PreOrden, InOrden y PostOrden), ya que podría darse el caso de:

Post(A)=Post(B) ^ In(A)=In(B)

1. **¿Puede construirse un ABB de forma unívoca dado su recorrido en inorden?**

Si. Si la lista no es vacía (árbol vacío), el elemento más a la izquierda de la lista será la raíz del árbol. Como es de búsqueda, los elementos en el hijo izquierdo serán menores. En el recorrido en preorden, el hijo izquierdo se recorre de forma recursiva después de la raíz, por lo que todos los elementos menores formarán el recorrido en preorden del hijo izquierdo. Por último, se recorre el hijo derecho de forma recursiva, por lo que el resto de elementos mayores formarán el recorrido en preorden del hijo derecho. Conociendo así los recorridos de ambos hijos, podemos construirlos con sendas llamadas recursivas.

1. **¿Qué consiguen los árboles binarios de búsqueda ABB?**

Consiguen que las operaciones de inserción y de búsqueda en el caso promedio serán de orden O (log n) cuando están equilibrados, aunque en el peor de los casos seguirá siendo de O(n) (árbol degenerado en una lista).

1. **¿Qué condición tienen que cumplir los elementos de un árbol para poder realizar búsquedas con un coste menor que O(n)?**

Que tengan establecida la relación de orden entre sí y que se posicionen en el árbol equilibrándolo.

1. **¿Qué condiciones debe cumplir un ABB para que la búsqueda sea menor que O(n)?**

Que el ABB esté lo más equilibrado posible. Destacar los AVL (ABB con autoequilibrado) donde la búsqueda, inserción y la eliminación son siempre de O (log n).

1. **¿Es siempre la eliminación de elementos en un ABB de orden mayor que O(n)?**

No, nunca es de orden mayor a O(n). sí tenemos un ABB equilibrado, la operación de eliminar nunca será mayor que de O (log n), y si el ABB ha degenerado en una lista, la operación será de orden O(n).

1. **¿En qué condiciones puede un árbol ser un APO y ABB simultáneamente?**

Aparte de cuando el árbol esté vacío o sólo tenga el nodo raíz, nunca.

1. **¿Qué aportan los AVL frente a los ABB?**

En un ABB, las operaciones de búsqueda, inserción y eliminación son, en el caso medio (cuando están equilibrados) de orden O (log n) y en el peor caso de orden O(n) (árbol degenerado en una lista). Un AVL, por definición, es un ABB equilibrado, por lo que elimina el peor de los casos anteriormente citado, luego garantiza que las operaciones sean de orden O (log n).

1. **Definición de AVL y factor de equilibrio**

Se define el factor de equilibrio de un nodo como la altura del subárbol derecho menos la altura del subárbol izquierdo de dicho nodo. Se define un árbol binario equilibrado como un árbol binario en el cual el factor de equilibrio de cada nodo es -1, 0 o 1. Se define un árbol AVL como un árbol binario de búsqueda equilibrado.

1. **¿Influye el orden de inserción de los datos en la altura de AVL?**

No, no influye. Esto es debido a que, por definición, un árbol AVL tiene la propiedad de auto equilibrado. Es decir, en todo momento para todos los nodos, la altura de la rama izquierda no difiera en más de una unidad a la altura de la rama derecha o viceversa, siendo esto independiente al orden de inserción de los datos.

1. **¿Por qué se exige un orden exacto en la inserción de elementos en el AVL?**

En los AVL no se exige un orden exacto en la inserción de elementos, ya que la propia operación de inserción se encarga de mantener el árbol equilibrado.

1. **¿Influye el orden de inserción de los elementos para el desequilibrio de un APO?**

En los APO no influye el orden de inserción de los elementos para el desequilibrio, ya que, por definición, es un árbol completo, los cuales siempre están equilibrados. El orden de inserción no sirve para nada.

1. **¿Puede reconstruirse un APO de forma unívoca dado su recorrido en preorden?**

Sí, dado que un APO es un árbol completo en el que el valor de cualquier nodo es menor que el de sus descendientes. Con el preorden obtenemos la raíz y el número de nodos y, por lo tanto, la altura del árbol. Con estos datos y a partir de la definición de APO podemos reconstruir el árbol.

1. **¿Es posible obtener coste O(n) en la eliminación de un nodo cualquiera de un APO?**

No es posible eliminar cualquier nodo de un APO, ya que solamente se puede eliminar la raíz del mismo.

1. **¿Es posible obtener coste O(n) en la inserción/eliminación de la raíz en un APO?**

No. Un APO por definición es un árbol completo parcialmente ordenado. Por tanto, cualquier de las dos operaciones toma un tiempo O (log2 n), puesto que en el árbol ningún camino tiene más de <<1+log2 n>> nodos. Estas operaciones son de orden O(n) cuando el árbol está degenerado en una lista y esta situación no puede darse en árboles completos.

1. **¿Influye el número de elementos de un APO en su desequilibrio?**

No, ya que un APO por definición es un árbol completo, es decir, que solamente pueden faltarle nodos en el último nivel y por la parte derecha, luego el factor de equilibrio va a ser -1 en el peor caso.

1. **¿Tiene sentido el concepto de un árbol terciario de búsqueda?**

Si, la generalización de los ABB para los árboles generales son los árboles B. Un árbol terciario de búsqueda sería un árbol B de orden m=3 (siendo m el número máximo de hijos de cada nodo) y k=2 (siendo k el número de claves que contiene como máximo cada uno de los nodos)

1. **Dado que interesa reducir al máximo la altura de un árbol B ¿por qué no aumentamos “indefinidamente” el número de hijos del árbol? Razona la respuesta.**

Al aumentar indefinidamente el número de hijos, aumentaríamos de forma indefinida el número de claves, aumentando el tamaño de cada nodo de forma indefinida, haciéndolos lentos de leer y procesar, superando los tamaños de los bloques de memoria.

1. **¿Por qué interesa que los árboles B tengan poca altura?**

El tener poca altura implica maximizar el número de nodos hijos de cada nodo interno, por lo que las operaciones para balancear el árbol se reducen, aumentando la eficiencia. Además, el tiempo de búsqueda en un árbol B es proporcional al número de niveles que posee.

1. **Se insertan un conjunto de elementos en un árbol B en un determinado orden. ¿La altura de ese árbol B resultado es independiente del orden de inserción?**

Si y no.

Si vamos a insertar un conjunto de elementos, y en el árbol hay claves libres donde pueden alojarse sin necesidad de promocionar, no hay problema, si no, la altura se vería afectada.

1. **Explica el por qué se exige un equilibrio perfecto en los AVL.**

En los AVL no se exige un equilibrio perfecto; el factor de desequilibrio puede ser -1, 0 o 1. Gracias a esta forma de equilibrio, la complejidad de una búsqueda en estos árboles es de orden logarítmico.

1. **¿Puede reconstruirse un árbol unívocamente dado su inorden?**

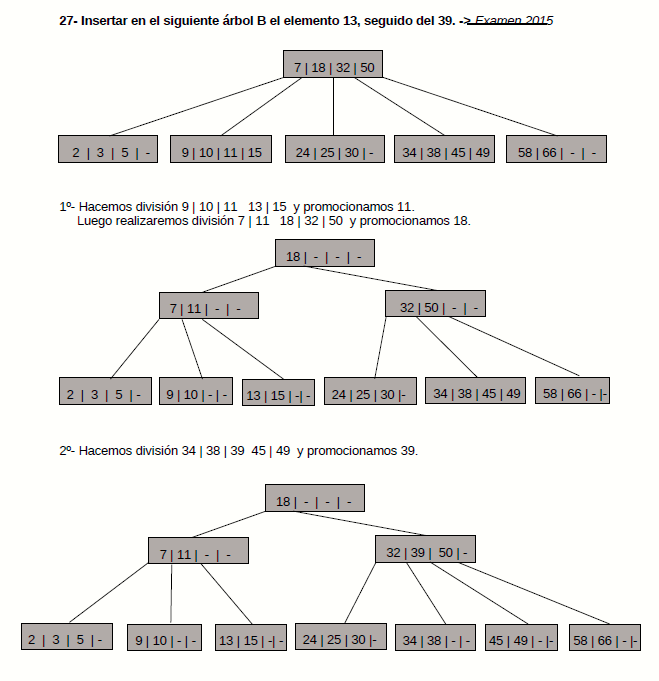
No, ya que no tenemos forma de saber cuál es el nodo raíz ni que elementos pertenecen al subárbol izquierdo ni cuales al derecho. Para reconstruir un árbol a partir del inorden, será necesario otro recorrido cualquiera (preorden o postorden)

1. **¿Puede reconstruirse un árbol unívocamente dado su recorrido en postorden y su recorrido en preorden?**

No se puede, harían falta los 3 tipos de orden.

1. **¿Por qué interesa reducir al máximo la altura de un árbol B?**

Porque al maximizar el número de nodos hijo de cada nodo, haciendo disminuir la altura del árbol, las operaciones para balancearlo se reducen, y aumenta la eficiencia. Indicar que lo ideal es que un nodo ocupe un bloque de memoria y el número de hijos sea consecuente a este tamaño.



<http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb_B.htm>

<https://es.wikipedia.org/wiki/%C3%81rbol-B#Inserci%C3%B3n>

<https://www.youtube.com/watch?v=EbiFlTGh0rI>

**Grafos**

1. **¿Por qué surgen los grafos?**

Porque se hacía necesaria una estructura que nos permitiese representar relaciones binarias entre elementos de un conjunto. Los grafos permiten estudiar las interrelaciones entre unidades que interactúan unas con otras.

1. **Explica las razones por las que no es necesario marcar los nodos visitados al realizar el recorrido de un grafo**

Si es necesario marcar los citados nodos. Existen dos razones. La primera es para no procesar un nodo más de una vez en grafos cíclicos (cuando llegue a un nodo visitado, no lo vuelvo a procesar). La segunda razón es para poder procesarlos todos si existe más de una componente conexa en el grafo.

1. **¿Puede recuperarse un grafo no dirigido a partir de sus recorridos en anchura y profundidad?**

No, ya que ambos recorridos nos devuelven una lista con los nodos que contiene el grafo, pero no proporciona información sobre las aristas (existentes o no) entre los nodos.

1. **¿Se podría mejorar el algoritmo de Dijkstra utilizando un APO?**

Si, es posible mejorarlo siempre y cuando los nodos estén formados por una estructura que contenga el coste del arco y el vértice que une.

1. **El algoritmo de Dijkstra, ¿Funciona correctamente con valores negativos?**

No, porque no garantizaría los mínimos locales. Al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista de coste negativo.

1. **¿Por qué se coloca el infinito en la diagonal principal de la matriz de costes en el algoritmo de Floyd?**

En el algoritmo de Floyd la diagonal principal no se establece a infinito, sino a cero, representando de esta manera que el coste de ir de un vértice a sí mismo es nulo. Si suponemos que entre un vértice ‘A’ y sí mismo existe un camino cíclico y hemos colocado la diagonal principal a infinito, en este caso nos daría como resultado que tiene menor coste realizar dicho camino que no movernos de A.

1. **¿Por qué no se permiten los costes negativos en el algoritmo de Floyd?**

Para que haya coherencia numérica.

Al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el coste general del camino al pasar por una arista de coste negativo.

1. **¿Por qué en Floyd se inicializa a cero la diagonal de la matriz resultado?**

Para representar que el coste de ir de un vértice a sí mismo es nulo.

1. **¿Por qué en Prim no existen ciclos? (en el resultado se sobreentiende)**

Prim asegura que en su algoritmo no se producen ciclos utilizando la siguiente técnica: partiendo de un nodo inicial, va construyendo el árbol generador de coste mínimo marcando los nodos que va visitando. De esta forma, nunca seleccionará una arista que conecte un nodo visitado con un nodo que también esté visitado, evitando así que se produzcan ciclos.

1. **¿Qué condición debe de cumplir un grafo no dirigido para poder utilizar el algoritmo de Prim?**

La condición es que el grafo debe ser ponderado y conexo, ya que, si no fuera conexo, sería imposible añadir los nodos inconexos al árbol generador y, si no fuera ponderado, no podríamos obtener un árbol de coste mínimo.

1. **Comente la siguiente afirmación: “Prim y Kruskal resuelven el mismo problema y dan la misma solución”**

Ambos resuelven el mismo problema que sería el encontrar un árbol generador de coste mínimo, pero no tienen por qué devolver el mismo resultado.

Este resultado no depende de que los algoritmos sean diferentes, sino de que el árbol generador de coste mínimo no tiene por qué ser único; depende de la existencia de distintas aristas con el mismo peso. Esto puede implicar que existan varias soluciones factibles que dan lugar a árboles generadores de coste mínimo distintos, si las condiciones iniciales en un algoritmo cambian.

El anterior hecho provoca que el algoritmo pueda tomar diferentes caminos para llegar a soluciones distintas. Por supuesto, si no existen aristas con el mismo peso, la solución será la misma para ambos algoritmos. Indicar además que en todos los casos la suma total de los pesos coincidirá siempre.

1. **¿Por qué el algoritmo de Kruskal asegura que no se producen ciclos?**

Porque el algoritmo trabaja con una partición que contiene todos los nodos, usando esta para comprobar constantemente que componentes del grafo se encuentran conexas (es decir, en el mismo subconjunto), de forma que nunca se va a coger una arista que tenga sus dos nodos dentro de un mismo subconjunto.

Si esto llega a suceder, se produciría un ciclo, puesto que se está insertando una arista a una componente ya conexa, cerrando de esta forma un ciclo.

1. **¿Es necesario ordenar las aristas en el algoritmo de Kruskal?**

El algoritmo de Kruskal es del orden de O (log n) en donde n es el número de vértices. Para poder conseguir esta complejidad, es necesario que las aristas sean ordenadas por su peso usando una ordenación por comparación del orden de O (m log m), en donde m es el número de aristas, lo que permitiría eliminar la arista de peso mínimo en tiempo constante. En el algoritmo de Kruskal que se nos ha propuesto, se puede conseguir mediante la utilización de un APO.

1. **¿Cuál es la condición que debe cumplir un grafo no dirigido para que Kruskal obtenga un resultado?**

La condición es que el grafo debe ser ponderado y conexo, ya que, si no fuera conexo, sería imposible añadir los nodos inconexos al árbol generador y, si no fuera ponderado, no podríamos obtener un árbol de coste mínimo.

1. **¿Por qué Kruskal no devuelve un grafo?**

En nuestra representación, Kruskal devuelve un grafo, en el que su interior se encuentra el árbol generador.

Se puede optar por cualquier estructura de datos que nos permita representar el árbol generador que Kruskal desarrolla, como pueden ser una matriz, un grafo, un árbol general, etc.

Indicar que en la implementación del algoritmo que se nos ha facilitado si se devuelve un grafo.

1. **¿Qué pasaría si Prim y Kruskal operaran sobre un grafo dirigido?**

Que no funcionaría correctamente, ya que ambos están diseñados para trabajar con aristas de grafos no dirigidos, no cumplimos dicha precondición. Tengamos en cuenta que, por definición, la matriz de un grafo dirigido no puede ser simétrica. En el caso del algoritmo de Kruskal que se nos ha facilitado en la asignatura, para copiar las aristas del grafo en el APO, solo se recorre la diagonal superior de la matriz, ya que se supone simétrica, luego ya estamos presuponiendo que el peso entre dos vértices es el mismo en ambas direcciones. En el caso del algoritmo de Prim que se nos ha facilitado en la asignatura, el coste de la arista se incorpora de forma simétrica en el árbol, ocurriendo lo mismo que en el Kruskal.

1. **Dado el algoritmo de Kruskal implementado mediante el TAD Partición, ¿Son los mismos arboles los de la partición y los del algoritmo?**

No son los mismos ya que cuando acaba Kruskall, la estructura del árbol se almacena en el objeto partición, depende del orden de inserción de las aristas (por peso) y la raíz de ese árbol no tiene por qué coincidir con la del árbol devuelto por el algoritmo.

1. **¿Qué se consigue con la técnica de unión por tamaño? ¿Y con la unión por atura?**

En la unión por tamaño, el árbol con menos nodos se convierte en subárbol del que tiene mayor número de nodos. En la unión por altura, el árbol menos alto se convierte en subárbol del otro.

En ambas técnicas se controla la altura del árbol resultante, consiguiendo que la operación unir () sea del orden de O (1) y que la operación encontrar sea del orden de O (log n)

1. **En la representación mediante bosques de árboles con unión por altura, ¿por qué las raíces de los árboles se representan con números negativos?**

Para distinguir si el dato almacenado en el vector (cuyo índice se corresponde con el nodo) se refiere a la altura (número negativo y por lo tanto se trata de un nodo raíz) o a su padre (número no negativo y por tanto no se trata de un nodo raíz). Para saber la altura, al dato almacenado como número negativo se le debe sumar 1 y posteriormente obtener su valor absoluto.

1. **Dado el TAD Partición, decir que combinación entre estructura de datos y estrategia es necesaria para que tanto la búsqueda como la unión sean de O (1)**

Es imposible conseguir que ambas operaciones sean de O (1).

Utilizando el control de altura podremos conseguir para la búsqueda un orden de O (log n) y para la unión un orden constante. Si además unimos la estrategia de comprensión de caminos, podemos lograr acercarnos a un orden constante en la búsqueda, pero nunca llegar a O (1)

1. **¿Qué beneficio aporta la compresión de caminos en la implementación del TAD partición?**

En la implementación del TAD partición mediante bosques de árboles y mediante unión por tamaño o por altura, la compresión de caminos se utiliza para poder acercarnos a órdenes constantes en la función encontrar. Esto lo conseguimos haciendo que los nodos por los que pasamos sean hijos del nodo raíz. Indicar que esta técnica permite acercarnos a coste 1, aunque no se garantiza.

Nota: La unión por tamaño o por altura garantiza el orden logarítmico en la búsqueda. La compresión de caminos permite acercarnos a un coste constante.

1. **Diferencias y similitudes entre Prim y Kruskal.**

Ambos algoritmos resuelven el mismo problema, que sería encontrar en un grafo conexo, no dirigido y ponderado; un árbol generador de coste mínimo.

Kruskal y Prim se diferencian en la metodología usada para la obtención del resultado y evitar la generación de ciclos. Esta diferenciación radica en que Prim parte de un nodo cualquiera y construye el árbol a partir de él marcando los nodos que va visitando, mientras que Kruskal trabaja con particiones para asegurar que no se producen ciclos.

También difieren en su orden de complejidad computacional temporal. Mientras que el algoritmo de Prim es de O (n2); el de Kruskal es más eficiente, siendo su orden de O (a log n).

1. **¿Qué ventajas e inconvenientes plantea el uso de matrices de adyacencia y costes para la representación de grafos?**

Como ventaja, son muy eficientes para comprobar si existe una arista entre un vértice y otro. Sin embargo, tienen el inconveniente de que pueden desaprovechar gran cantidad de memoria si el grafo no es completo. Además, si la matriz no es dinámica, no se pueden añadir o eliminar vértices del grafo. La matriz de coste nos permite expresar el valor de las aristas de un grafo ponderado.

1. **¿Qué ventajas e inconvenientes plantea el uso de listas de adyacencia para la representación de grafos?**

Esta representación aprovecha el espacio de memoria, pues solo se representan los arcos existentes en el grafo. Sin embargo, son pocos eficientes para determinar si existe una arista entre dos vértices del grafo, pues esta operación implica recorrer una lista.

1. **¿Existe alguna estructura para la representación de grafos que permita añadir y suprimir vértices?**

Si. Estas operaciones son factibles, si se usan como estructuras, matrices y vectores dinámicos, o una lista de listas de adyacencia.

**Extras**

1. **¿Existe una estructura de datos que permita ejecutar simultáneamente las operaciones de unión y encontrar en un tiempo constante?**

No, pero utilizando la representación del TAD Partición mediante bosque de árboles, que consiste en representar cada partición como un árbol cuya raíz es el representante de dicha partición, conseguiremos que la operación de unión sea de coste constante, utilizando la técnica de unión por altura o bien la de unión por tamaño, estableciendo la raíz del árbol de menor altura o con menos nodos, respectivamente, como hijo de la raíz del árbol con mayor altura o más nodos.

Esta solución nos seguiría dejando el problema de que la operación encontrar es de coste n. utilizando la técnica de compresión de caminos a corto plazo no obtendríamos ventaja, pero a medio-largo plazo, obtendríamos un árbol de altura próxima a 1.

1. **Aparte de una mejora en el problema de la búsqueda, ¿realizan alguna aportación adicional los árboles?**

Sí, aparte de mejorar la complejidad de la búsqueda (O log n), aportan la posibilidad de

tener estructuras de datos cuyos elementos se organicen de forma jerárquica, para así

representar situaciones no representables mediante estructuras de datos lineales (pilas, colas,

listas, etc.).

1. **Explicar las ventajas e inconvenientes que plantean las colas con prioridad representadas como una lista ordenada por prioridad frente a un APO**

La ventaja es que, usando una lista ordenada, la extracción y eliminación del primer elemento de la cola (o frente) se hace en O (1), mientras que en el APO la eliminación toma un tiempo de O(log) puesto que hay que reorganizar el árbol (empujar o flotar los nodos).

Sin embargo, la inserción en una lista ordenada toma un tiempo de O(n), mientras que en el APO es de O(log).

1. **¿Qué estrategia de unión (por altura o tamaño) combina mejor con la técnica de compresión de caminos en el TAD partición?**

La unión por tamaño, puesto que la compresión de caminos puede cambiar la altura de los

árboles, y esto puede dar problemas si lo combinamos con la unión por altura (donde el árbol

más bajo se convierte en subárbol del más alto).

1. **Todos los subgrafos de n-1 aristas de un grafo de n nodos ¿son árboles generadores? ¿O** **solo los subgrafos de coste mínimo?**

Serán árboles generadores si están formados por los n nodos (vértices) del grafo y n - 1

aristas de éste, y dichos nodos están conectados en el subgrafo.

Los subgrafos de coste mínimo son árboles generadores, puesto que están formados por los n

vértices de un grafo (que debe ser no dirigido, conexo y ponderado) y n - 1 aristas (cuya suma

total, del peso de las aristas es el mínimo).

1. **Dado un conjunto de elementos insertados en el mismo orden en un ABB y en un AVL, ¿Daría como resultado árboles de la misma altura?**

Depende de si se mantuviese el equilibrado en el ABB ya que, por definición, un AVL es un ABB con auto equilibrado donde su factor de equilibrio es -1, 0 o 1.

1. **Un AVL es un ABB, pero el recíproco no es cierto. ¿Estás de acuerdo?**

Un AVL es por definición un ABB equilibrado (el factor de equilibrio de cada nodo del árbol

debe ser 0, -1 o 1). Sin embargo, un ABB no tiene por qué estar siempre equilibrado (árbol

degenerado en una lista, o bien el factor de desequilibrio de sus nodos es diferente a los

valores antes mencionados), por lo que será AVL SÓLO cuando se cumpla la condición del

factor de equilibrio antes mencionada (es decir, cuando esté equilibrado).

1. **Dado un árbol conexo y acíclico, ¿es necesario marcar los nodos visitados al recorrerlo?**

Si es necesario, ya que, aunque no sea necesario marcar los nodos visitados para evitar los ciclos, la segunda razón de marcar los nodos es procesar si hay mas de una componente conexa al grafo.