TEORIA-EDNL.pdf

vvss5

Estructuras de Datos no Lineales 2º

Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

TEORÍA EDNL

ÁRBOLES

1. ¿Por qué surgen los árboles?
   Surgen de la necesidad de representar elementos organizados en una jerarquía de diferentes capas o niveles, ya que con estructuras lineales no es posible representarlas. Además, permiten realizar búsquedas en orden logarítmico, lo cual no se consigue con las estructuras lineales que son de O(n).

2. ¿En qué situaciones es conveniente utilizar un vector de posiciones relativas?
   Es recomendable cuando se trabaje con árboles completos y sepamos el grado del árbol(número máximo de hijos), ya que si el árbol no es completo se desaprovecharía memoria en las posiciones del vector en las que no hubiese ningún nodo.
   Un ejemplo en el que esta representación es conveniente es para árboles APO.

3. ¿Cuántos tipos de recorrido de árboles en anchura existen?
   Solo existe un recorrido en anchura, el recorrido por niveles. Consiste en procesar la raíz, luego los hijos directos de izquierda a derecha, seguidamente los hijos directos de estos hijos y así sucesivamente.
   Existen dos versiones, una iterativa y otra recursiva.

4. ¿Cuántos tipos de recorridos de árboles en profundidad existen?
   Existen 3 tipos:
   - Preorden
   - Inorden
   - Postorden

5. ¿Qué condiciones tiene que cumplir los elementos de un árbol para poder realizar las búsquedas con un coste menos que O(n)?
   Es necesario que los elementos cumplan una relación de orden entre sí y que el árbol esté equilibrado.

6. ¿Puede reconstruirse un árbol unívocamente dado su inorden?
   No, para poder construir un árbol 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 qué nodos pertenecen al subárbol izquierdo y derecho de dicho nodo. Depende del árbol

7. ¿Por qué las operaciones de Insertar y Eliminar son de O(1) en la representación vectorial o de celdas enlazadas de árboles binarios?
   - Representación Vectorial:
     - Insertar: Porque siempre añadimos en la última posición del vector que esté vacía y el acceso al vector es de O(1).
     - Eliminar: Porque el puntero al nodo eliminado apuntaría a NODO_NULO, y el último nodo se mueve a donde estaba el nodo eliminado.

   - Representación de Celdas Enlazadas:
     - Insertar: Porque se cambia el puntero que tiene el padre al nuevo nodo insertado.
     - Eliminar: Porque libera la memoria de donde estaba el nodo y el puntero pasaría a apuntar a NODO_NULO.

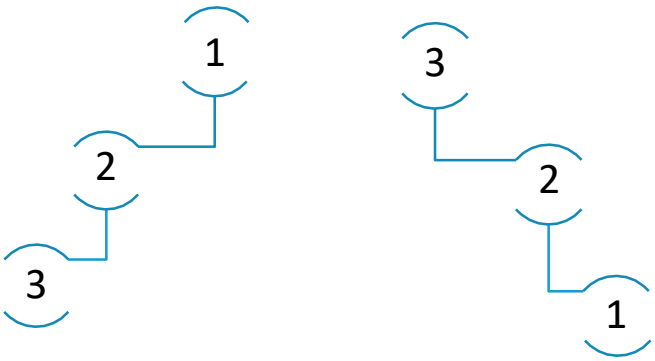
8. ¿La eliminación de nodos de un árbol binario se puede conseguir con O(1), cuando se utiliza una representación vectorial con índice al padre, hijo izquierdo e hijo derecho?
   En una representación vectorial, la eliminación de nodos es siempre de orden constante.
   (Si guardamos el número de nodos del árbol como Nnodos y utilizamos la técnica de compactación, que consiste en que al eliminar un nodo del árbol, este es sustituido en el vector por el nodo en la posición Nnodos – 1 del vector, y decrementamos en 1 Nnodos, y la inserción la hacemos añadiendo el nuevo nodo en la posición Nnodos del vector, ambas operaciones tendrán conste O(1).)

9. Ventajas de la representación de árboles binarios con celdas enlazadas frente a matrices.
   La representación mediante celdas enlazadas utiliza memoria dinámica por lo que es más eficiente en cuanto a espacio, mientras que la representación con matrices es pseudoestática. Con celdas enlazadas ocupamos exactamente el espacio que necesitamos, sin embargo con matrices debemos establecer a priori el tamaño.

10. ¿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 qué nodos pertenecen al subárbol izquierdo y derecho de dicho nodo.

11. ¿Tiene sentido el concepto de árbol terciario de búsqueda?
    Sí, teniendo en cuenta que la generalización de los ABB para árboles generales son los denominados árboles B, un árbol terciario de búsqueda sería un árbol B de orden m = 3 y k = 2, siendo m el número máximo de hijos de cada nodo y k el número de elementos o claves que tiene como máximo cada uno de los nodos. Este tipo de árboles nos permite almacenar varios elementos en un mismo nodo, lo que nos permite reducir el número de accesos a memoria secundaria, y por tanto, el tiempo de búsqueda.

12. Sean A y B dos árboles binarios diferentes, indique si puede ocurrir simultáneamente que: Pre(A) = Post(B) y Pre(B) = Post(A).
    Esto puede darse si ambos árboles vinarios son árboles vacíos, cuando sólo tienen un nodo raíz y cuando siguen un esquema similar al siguiente:



Siendo:

- Pre(A) = 1,2,3
- Post(A) = 3,2,1
- Pre(B) = 3,2,1
- Post(B) = 1,2,3

13. Las operaciones del TAD Árbol Binario permiten insertar y eliminar hojas pero no nodos internos. ✓

14. Para destruir un Árbol Binario completo implementado mediante una representación vectorial, no es necesario eliminar los nodos uno a uno en postorden, esto sólo se haría si se trata de un subárbol del mismo. F (Caso para la representación mediante estructura enlazadas)

15. La eficiencia espacial de la representación de un Árbol Binario mediante un vector de posiciones relativas será mejor cuantos menos nodos falten en el nivel inferior. ✓

16. Función contarNodos de un Árbol Binario siempre es de O(n). ✓

17. ¿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) y en la representación con posiciones relativas las relaciones entre los nodos (Padre/Hijos) van en relación con el mismo. Luego si no conocemos su grado no podemos prever la posición que deberían ocupar los hijos.

18. ¿Se podría usar las listas doblemente enlazadas en los árboles generales mediante listas de hijos?
    Sí, pero no tiene sentido ya que la especificación del TAD no contempla la posibilidad de acceder al hermano izquierdo de un nodo y de esta manera si sería posible.

19. ¿Puede determinarse un árbol general a partir del inorden y postorden?
    ~~No se puede, harían falta los tres tipos inorden, preorden y postorden, puesto que podría darse el caso de:~~
    ~~Post(A) = Post(B) ^ In(A) = In(B)~~
    Se puede hacer a partir del recorrido en inorden y otro (preorden o postorden) e información adicional para determinar el grado del arbol

20. ¿Por qué interesa que los árboles B tengan poca altura?
    Porque la búsqueda de elementos es proporcional al número de niveles que posee el árbol, de este modo reduciremos el número de accesos a memoria.

21. Dado que interesa reducir al máximo la altura de un árbol B, ¿por qué no aumentamos "indefinidamente" el número de hijos del árbol?
    No se puede hacer el tamaño indefinido, ya que el objetivo de hacer un árbol B es reducir el tiempo de acceso y si supera el número máximo de hijos que puede haber en un nodo del árbol B correspondiente, tendríamos que acceder más de una vez, y por tanto, se vuelve ineficiente.

22. ¿Existe alguna operación claramente ineficiente en los árboles generales representados mediante listas de hijos?
    Utilizando la implementación de árboles generales mediante listas de hijos, las operaciones de búsquedas de un nodo dentro de la lista de hijos de su padre son poco eficientes, ya que la búsqueda en una lista es de O(n). Por tanto, las operaciones de búsqueda, inserción y eliminación del hermano derecho de un nodo resultarán poco eficientes.

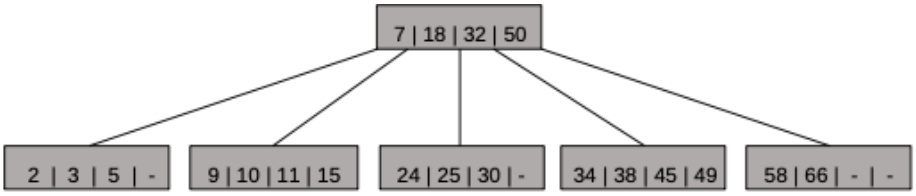
23. Las operaciones del TAD Árbol General no permiten insertar y eliminar nodos internos, solo nodos hoja. V

24. Para reconstruir un Árbol General solo necesitamos su recorrido en inorden y en postorden o preorden, pero no hace falta el grado del árbol ya que cada nodo puede tener un grado diferente. F

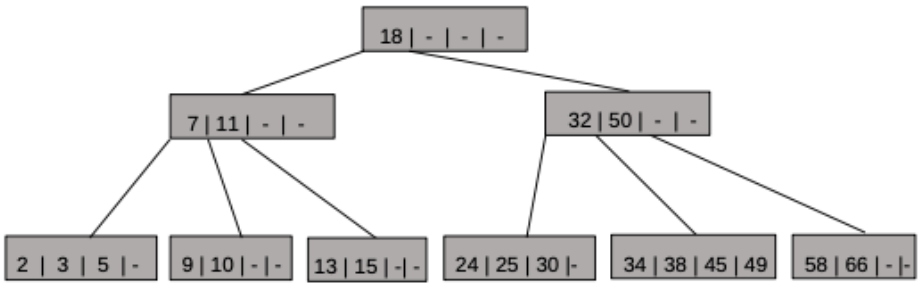
Eficientes en tiempo
(Cuanto mas alto mas cortas son las listas y al ser la busqueda de O(n) es más eficiente tener menos elementos en ellas)

25. La representación del TAD Árbol General mediante listas de hijos es más eficiente en espacio cuanto más bajo es el árbol para un número determinado de nodos. V (cuanto más bajo es el arbol menos listas tiene y es mas eficiente tener una lista grande que varias pequeñas).
26. Se define el desequilibrio de un Árbol General como la máxima diferencia entre las alturas de los subárboles más bajo y más alto de cada nivel. Esta definición y la diferencia de longitudes entre la rama más larga y más corta de dicho árbol son equivalentes. F es verdadero si las ramas cuelgan de la raiz pero como no tienen por que colgar de la raiz es F.
27. La función para calcular la profundidad de un Árbol General es similar a la del Árbol Binario y del mismo coste O(log n). F en el peor caso en Abin y Agen es O(n)

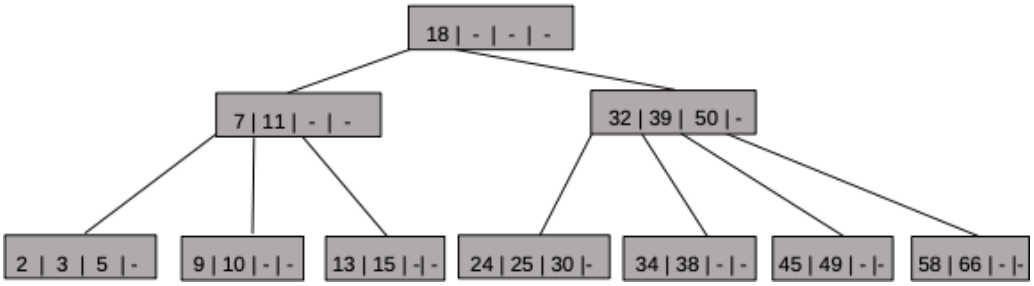
28. Insertar en el siguiente árbol B el elemento 13, seguido del 39.



1º- Hacemos división 9 | 10 | 11 13 | 15 y promocionamos 11.
Luego realizaremos división 7 | 11 18 | 32 | 50 y promocionamos 18.



2º- Hacemos división 34 | 38 | 39 45 | 49 y promocionamos 39.



29. ¿Qué consiguen los árboles binarios de búsqueda (ABB)?
    Consiguen que las operaciones de inserción y búsqueda en el caso promedio sean de orden O(log n) cuando están equilibrados, aunque en el peor caso seguirá siendo de O(n).

30. ¿Es siempre la eliminación de elementos en un ABB de orden mayor que O(n)?
    No, ya que las operaciones de búsqueda, inserción y eliminación dependen del grado de desequilibrio del árbol, y para un ABB de n elementos, el coste oscila entre O(log2 n) (para árboles AVL, en los que el desequilibrio de un nodo es -1, 0 o 1) y O(n) (para árboles degenerados en listas). Debido a que en el peor caso el coste de la eliminación de elementos es de O(n), el coste nunca va a ser de un orden superior a éste.

    (segunda opción)
    No, nunca supera O(n) ya que este es el peor caso, en donde el árbol tiene forma de lista.
    Puede ser:
    -NODO HOJA: Lo eliminamos sin problema en O(1)
    -SOLO 1 HIJO: Eliminamos el nodo y subimos un nivel todo lo que tiene abajo.O(1)
    - DOS HIJOS O MAS: Tomamos el elemento más pequeño de los más grandes, los cambia, y suben los otros niveles. O(n)

31. ¿Qué condiciones debe cumplir un ABB para que la búsqueda sea menor que O(n)?
    Dependiendo del orden de inserción de elementos en un ABB, este, en el peor caso, puede degenerarse en una lista, haciendo que las operaciones de búsqueda, inserción y eliminación sean de O(n).
    En cualquier otro caso, estas operaciones serán de orden menor. Para asegurar que esto no ocurra, el ABB debe estar equilibrado.

32. 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, teniendo en cuenta el mínimo permitido según el orden del árbol. ✓

33. 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 la mediana al nodo padre. No se permite que existan nodos con menos elementos de la mitad (por defecto) de la capacidad de un nodo. F (la raiz si)

34. 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. ✓

35. Para conseguir que la anchura del árbol B sea menor, me interesa crear nodos con el mayor tamaño posible. V se refiere a nº de hijos (cuanto más grandes sean los nodos, menos hijos hacen falta.

36. La propiedad de búsqueda de un ABB permite encontrar un elemento en un tiempo de O(n) en el caso peor. V

37. La eliminación de elemento en un ABB puede llegar a tener un coste O(n). V

38. El recorrido en preorden de un ABB determina unívocamente el ABB del que procede. V

39. Los nuevos elementos de un árbol B se insertan en las hojas y, si es necesario, se reorganiza el árbol. V

40. En un árbol B de orden m, todos los nodos contienen un mínimo de [m-1/2] claves, y un máximo de m-1. F la raíz no.

ÁRBOL BINARIO PARCIALMENTE ORDENADO (APO)

41. ¿En qué condiciones puede un árbol ser un APO y ABB simultáneamente?
Sólo cuando se trate de un árbol vacío o que solo tenga el nodo raíz.

42. ¿Influye el orden de inserción de los elementos para el desequilibrio de un APO?
No, porque por definición de APO, es un árbol binario completo, en el que sus niveles por tanto, serán completos exceptuando el último que puede tener huecos a la derecha. Por tanto, se puede asegurar que sea cual sea el orden de inserción de los elementos, la estructura del árbol resultante es el misma y por lo tanto el desequilibrio.

43. ¿Influye el número de elementos de un APO a su desequilibrio?
Sí, ya que un APO es un árbol binario completo. Un árbol binario completo de altura h tiene entre 2^h y 2^(h+1) nodos. En el caso de que el numero de nodos del APO fuese 2^(h+1) significaría que el desequilibrio de todos los nodos del árbol es 0, y por tanto el del árbol. En caso contrario, el desequilibrio del árbol sería 1.

44. ¿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.

cualquier recorrido en profundidad vale para reconstruirlo debido a su propiedad de completitud

la propiedad de completitud implica que la altura menor posible de un APO de n nodos es $h = \log_2 n$ y en el peor caso insertar y eliminar son de $O(h)$ por lo cual $O(\log n)$

45. ¿Es posible obtener coste O(n) en la eliminación de un nodo cualquiera de un APO?
    No. Los APO se utilizan para representar colas con prioridad, y se caracteriza por ser un árbol binario completo y parcialmente ordenado.
    La operación suprimir únicamente nos permite eliminar el elemento que se encuentra en la cima o raíz del árbol. Esta operación, elimina el nodo del último nivel del árbol, más a la derecha, sustituyendo el elemento que había en el nodo raíz por el del nodo eliminado, y el elemento que ahora se encuentra en la raíz, es recolocado en su nodo correspondiente gracias al criterio de ordenación que sigue el árbol y para que se siga cumpliendo. Este mismo criterio permite que se recorra únicamente una rama del árbol desde la raíz hasta el nodo que corresponde al elemento que se encontraba en el nodo eliminado, lo que hace la operación de eliminación o de supresión O(log2 n).

46. ¿Es posible obtener coste O(n) en la inserción/eliminación de la raíz de un APO?
    No. Los APO se utilizan para representar colas con prioridad, y se caracteriza por ser un árbol binario completo y parcialmente ordenado. La operación suprimir únicamente nos permite eliminar el elemento que se encuentra en la cima o raíz del árbol. Esta operación, elimina el nodo del último nivel del árbol, más a la derecha, sustituyendo el elemento que había en el nodo raíz por el del nodo eliminado, y el elemento que ahora se encuentra en la raíz, es recolocado en su nodo correspondiente gracias al criterio de ordenación que sigue el árbol y para que se siga cumpliendo. Este mismo criterio permite que se recorra únicamente una rama del árbol, desde la raíz hasta el nodo que corresponde al elemento que se encontraba en el nodo eliminado, lo que hace la operación de eliminación o de supresión O(log2 n). De ese mismo modo, la inserción se realiza añadiendo el nuevo nodo en la posición libre del árbol más a la derecha del último nivel. Una vez añadido este nodo, el elemento de dicho árbol es "flotado" hacía arriba buscando su posición correspondiente para seguir cumpliendo la propiedad de ordenación del APO. Por tanto, se recorre únicamente una rama del árbol, desde el nodo añadido hasta el nodo que corresponde al elemento que se añade, lo que hace la operación de inserción sea también O(log2 n).

47. Los elementos de un APO se obtienen en orden mediante la extracción sucesiva de estos. ✓ Se sacan elementos de la raíz y estos se van reordenando por lo que en la raíz siempre estará el más pequeño.
48. Todo APO min-max cumple estrictamente con las condiciones que hemos definido para un APO, pero no ocurre al contrario. F todas las condiciones no se cumplen
49. La propiedad de orden parcial de un APO no implica que siempre va a estar equilibrado. ✓
50. Si un Árbol es un APO, tiene un desequilibrio en valor absoluto menor o igual que 1, pero no todo desequilibrio menor o igual que 1 indica que sea un APO. ✓

51. El recorrido en anchura de un APO no proporciona el acceso en orden a los elementos almacenados. O se pueden obtener los elementos en orden de un APO. F

ÁRBOL DE BÚSQUEDA EQUILIBRADRO (AVL)

52. ¿Qué aportan los AVL frente a los ABB?
    En un ABB de n elementos las operaciones de búsqueda, inserción y eliminación dependen del grado de desequilibrio del árbol y puede llegar a ser O(n) en el caso más desfavorable (árbol degenerado en una lista). Los AVL son árboles binarios de búsqueda en el cuál el factor de equilibrio de cada nodo es -1, 0 o 1, es decir, el árbol está equilibrado. Por ello se evita que el árbol se degenere en una lista y se asegura que tanto las búsquedas de elementos como las inserciones y eliminaciones de nodos se pueden efectuar en O(log2 n).

53. Explica 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 de 1, 0 o -1.
    La razón de que se exija este desequilibrio es para garantizar que las búsquedas se realicen en O(log n)

54. ¿Influye el orden de inserción de los datos en la altura de un AVL?
    No, debido a que por definición, un AVL tiene la propiedad de autoequilibrado. Es decir, en todo momento para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa, siendo esto independiente al orden de inserción de los datos.

55. El factor de equilibrio en los AVL.
    El factor de equilibrio o balance de un nodo se define como la altura del subárbol derecho menos la altura del subárbol izquierdo correspondiente. El factor de equilibrio puede ser:
    - -0 -> está equilibrado
    - -1 -> Falta hijo Izquierdo
    - - -1 -> Falta hijo Derecho

56. 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. F Si un arbol esta equilibrado es O(log n)

57. La propiedad de equilibrio de un AVL permite encontrar un elemento en un tiempo de O(log n) en el caso peor. ✓

El AVL tiene autoequilibrado, el Abb no

58. La inserción en el mismo orden de un conjunto de elementos en un ABB y un AVL no tendría porque dar como resultado árboles de la misma altura. ✓

59. Un AVL es un ABB y el recíproco no es cierto. ✓

60. Es cierto que todos los AVL son ABB y es cierto que algunos ABB no sean AVL ✓

61. La propiedad de equilibrio de un AVL no implica que su altura sea la mínima posible. ✓ Las hojas no estan todas en el mismo nivel (no es completo)

62. 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. F

63. La propiedad de búsqueda de un ABB permite encontrar un elemento en un tiempo de O(n) en el caso peor. V

64. La condición de equilibrio no perfecto de un AVL asegura que la inserción de un elemento se pueda hacer a un coste de O(log n) en el peor caso. ✓

TAD PARTICIÓN

65. Dado el TAD Partición, decir qué combinación entre estructuras de datos y estrategia es necesaria para que tanto la búsqueda como la unión sean de O(1).
No existe ninguna estructura de datos que haga posible que ambas operaciones sean de orden O(1) simultáneamente. Podemos optar por hacer que únicamente la búsqueda sea de O(1) utilizando para ello, por ejemplo, un vector de pertenencia como estructura de datos o hacer que únicamente la unión sea de O(1) usando para ello un bosque de árboles como estructura de datos y una estrategia de unión por tamaño o unión por altura, con lo que se conseguirá que la operación de búsqueda sea de orden O(log n) en el peor caso.
En todo caso, utilizando unión por altura o por tamaño y búsqueda por compresión de caminos, después de muchas ejecuciones podría ser que ambas hubieran conseguido tener un tiempo de O(1).

66. ¿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. He de 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.

67. En el TAD Partición, ¿es posible emplear la unión en altura y la compresión de caminos a la vez?
    Sí. Son dos técnicas que hacen cosas distintas. La unión por altura realiza la operación Unión() de manera que el árbol con menos altura siempre será hijo del árbol con más altura, mientras que la compresión de caminos, lo que hace es que en cada búsqueda, al subir de nivel hacemos que el nodo por el que pasamos sea hijo de la raíz y así nos aproximamos a la solución ideal de dos niveles. Por tanto, se emplean las dos cosas a la vez, de hecho, el uso de estas dos técnicas de manera simultánea se utiliza para conseguir después de muchas ejecuciones mejores tiempos para las operaciones Encontrar() y Unión().

68. ¿Qué se consigue con la técnica de unión por tamaño? ¿y con la técnica de unión por altura?
    Ambas técnicas aseguran un coste, en el peor caso, de log n para la operación encontrar.
    - Por tamaño: El árbol con menos nodos se convierte en subárbol del que tiene mayor número de nodos.
    - 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).

69. ¿Existe una estructura de datos que permita ejecutar simultáneamente las operaciones de unión y encontrar en 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 con 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, que consiste en que cada vez que se llama a la operación encontrar, los nodos de la rama que une la raíz con el nodo a encontrar pasen a ser hijos de la raíz, a corto plazo no obtendríamos ventaja, pero a medio largo plazo obtendríamos un árbol de altura próxima a 1, lo que reduciría el coste de la operación encontrar casi a constante.

GRAFOS

70. ¿Por qué surgen los grafos?
    Surgen de la necesidad de tener una estructura de datos cuyos nodos no estén organizados de manera secuencia (como en las listas) ni tampoco estén

relacionados de manera jerárquica (como en los árboles). En este sentido, lo único que diremos es que es una estructura que conecta los nodos de una red mediante aristas.

71. Explica las razones por las que no es necesario marcar los nodos visitados al realizar el recorrido de un grafo.
Dado que no existe secuencialidad entre los nodos de un grafo ni hay una jerarquía definida, nada nos impide que podamos entrar en un ciclo. Por tanto, es necesario marcar los nodos recorridos para evitar recorrerlo más de una vez.

ALGORITMO DE PRIM, DIJSTRA, KRUSKAL Y FLOYD

72. 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 únicamente de que los algoritmos sean diferentes, sino además 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 (el vértice inicial) 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.

73. ¿Por qué el algoritmo de Kruskal asegura que no se producen ciclos?
El resultado de aplicar el algoritmo de Kruskal es un árbol generador (o de extensión) de coste mínimo y, por definición, un árbol es un grafo no dirigido conexo y acíclico. Para asegurar esto, este algoritmo utiliza el TAD partición, separando cada nodo del grafo en una partición. Cuando se encuentra el camino de coste mínimo entre 2 nodos, se utiliza la operación unión de este TAD para unir las particiones a las que pertenecen ambos nodos. Antes de hacer esto, el algoritmo comprueba que el representante de cada partición no sean el mismo, en cuyo caso significaría que ya se había encontrado un camino de coste mínimo entre esos nodos.

74. ¿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

ES INTERESANTE

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.

75. ¿Cuál es la condición que debe cumplir un grafo no dirigido para que Kruskal obtenga un resultado?
    Debe ser un grado ponderado, no dirigido y conexo.

76. ¿Por qué Kruskal no devuelve un grafo?
    En nuestra representacion, Kruskal devuelve un grafo ponderado, acíclico y conexo, 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...

77. ¿Qué pasaría si Prim y Kruskal operaran sobre un grafo dirigido?
    Que no funcionarían correctamente y retornaría un resultado erróneo. El motivo es que ambos están diseñados para trabajar con aristas de grafos no dirigidos, no cumplimos dicha precondición. En el caso del algoritmo de Kruskal que se nos ha facilitado en la asignatura, para copiar las aristas del grafo en el APO, sólo 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 Kruskal.

78. Dado el algoritmo de Kruskal implementado mediante el TAD Partición, ¿son los mismos árboles los de la partición y los del algoritmo?
    No son los mismos (o es muy improbable) 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 porque coincidir con la del árbol devuelto por el algoritmo. El TAD Particion es un algoritmo aux que se utiliza para resolver Kruskal pero no contiene la solucion

79. ¿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.

80. 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).

81. ¿Qué ventajas e inconvenientes plantea el uso de matrices de adyacencia y costes para la representación de grafos?
    - Ventajas: Son muy eficientes para comprobar si existe una arista entre un vértice y otro.
    - Inconvenientes: Desaprovechan gran cantidad de memoria si el grafo no es completo. Además, si la matriz no es dinámica, no se pueden añadir o eliminar vértices.

82. ¿Qué ventajas e inconvenientes plantea el uso de listas de adyacencia para la representación de grafos?
    - Ventajas: Aprovecha el espacio de memoria, pues solo se representan los arcos existentes en el grafo. Además, permite añadir y suprimir vértices.
    - Inconvenientes: Son poco eficientes para determinar si existe una arista entre dos vértices del grafo, pues esta operación implica recorrer la lista.

83. ¿Existe alguna estructura para la representación de grafos que permita añadir y suprimir vértices?
    Sí, estas operaciones son factibles si se usan como estructuras matrices y vectores dinámicos o una lista de adyacencia.

84. ¿Por qué no se permiten los costes negativos en Floyd?
    Porque podríamos encontrar un camino de un nodo hacia sí mismo con coste menor, lo cual no tendría sentido.

85. El algoritmo de Dijkstra, ¿funciona correctamente con valores negativos?
    El algoritmo de Dijsktra, es un algoritmo voraz, es decir, se va quedando con la mejor solución hasta el momento. Si se permitiesen los costes negativos, podría ocurrir, que llegados a un punto de la traza del algoritmo, en el que tendríamos una solución concreta (se supone que la mejor hasta el momento), nos encontrásemos con un camino más corto, originado por estos costes negativos, lo cual sería incoherente ya que hemos dicho que se va quedando con la mejor solución, y si este camino fuese realmente mejor (más corto), ya lo habríamos cogido en un punto anterior de la traza del algoritmo. Además, con los costes negativos, podrían originarse ciclos.

86. ¿Por qué se coloca infinito en la diagonal principal de la matriz de costes en el algoritmo de Floyd?
    La diagonal principal de Floyd se inicializa a 0 indicando así que la longitud de ir desde un vértice a sí mismo es igual a 0.