3.1. Explicar por qué es necesario, en la representación de conjuntos mediante árboles trie, utilizar una marca de fin de palabra $ (puesto que podríamos hacer que las palabras del conjunto se correspondieran con las hojas del árbol, sin necesidad de utilizar marcas de fin).

3.2. En un árbol trie queremos representar palabras acentuadas, por lo que se propone añadir el conjunto de caracteres los siguientes {Á, á, É, é, Í, í, Ó, ó, Ú, ú}. Explicar cómo afectaría a la memoria y al tiempo de ejecución, con las representaciones de nodos con arrays y con listas enlazadas.

3.3. * Mostrar el resultado de insertar los siguientes elementos en un árbol TRIE: jurel, julepe, jueves, julepes, giles. Elegir un tipo de representación para el árbol y hacer una estimación aproximada de la memoria ocupada (para este ejemplo).

3.4. Escribir los procedimientos *Asigna*, *Valor_de*, *Anula* y *Toma_nuevo* para nodos de tries representados como listas de celdas. Calcular la memoria ocupada por esta estructura y el tiempo de ejecución de estas operaciones.

3.5. Un algoritmo de compresión dado se basa en la repetición de secuencias que suelen existir en los ficheros. El programa lee un fichero y devuelve un fichero de salida comprimido. Si se encuentra una secuencia larga que apareció con anterioridad, se sustituye la secuencia por una referencia *COPIA(X, Y)*, que indica que en ese lugar se deben colocar X caracteres empezando por los que aparecieron Y posiciones antes en ese mismo fichero. Por ejemplo, la cadena: "Modulador-Demodulador..." se comprimiría como: "Modulador-Dem*COPIA(8, 12)*...". Utilizando tries se facilitaría la búsqueda de secuencias que han aparecido con anterioridad. Describir las principales características de la estructura de tries en esta aplicación y (sin detallar excesivamente) cómo sería manejado el trie en el proceso de compresión.

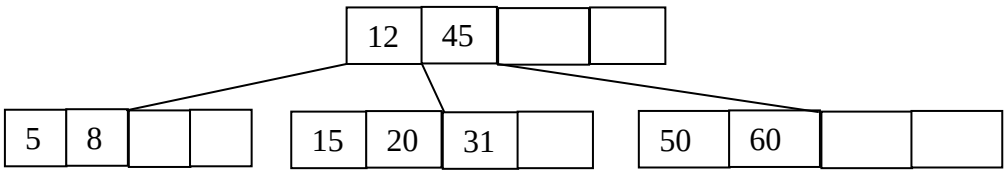
3.6. Tal y como hemos visto en clase, los tries se utilizan para representar conjuntos o diccionarios. Teniéndolo en cuenta, ¿es correcto definir un trie como un tipo de datos abstracto? Justificar la respuesta, en función de las propiedades de un TDA. En caso afirmativo, ¿en qué modelo matemático se basa? Resolver las mismas cuestiones para las tablas de dispersión y los árboles B.

3.7. Realizar una tabla comparativa en la que se muestre la eficiencia de la operación *Miembro* y la memoria total ocupada, para la representación de conjuntos con dispersión abierta, cerrada y tries con nodos representados con matrices y con listas. Expresarla en función de las variables: **n** palabras en el conjunto, **p** prefijos, ℓ longitud total de las palabras, $\mathbf{m}=\ell/\mathbf{n}$ longitud media de una palabra, **B** cubetas en la dispersión, $\mathbf{k_1}$ bytes/puntero, **d** caracteres en el alfabeto, $\mathbf{k_2}$ byte/carácter. Comentar los resultados obtenidos.

3.8. Un sistema multiusuario debe llevar el control de las personas que acceden al mismo. Para ello, se debe guardar para cada persona su nombre, apellidos y una clave de acceso. La operación básica consiste en dado un nombre y apellidos, obtener su clave. Puesto que se espera que el sistema tenga una cantidad muy grande de usuarios (muchos de ellos tendrán el nombre o algún apellido

comunes), se busca una representación eficiente en cuanto a tiempo y tamaño. ¿Cómo sería la representación mediante árboles trie? ¿Qué ventajas e inconvenientes tiene? ¿Qué otras estructuras pueden ser adecuadas para este problema?

3.9. * El dibujo de abajo representa un árbol B de orden p=5. Sobre el mismo se aplican las siguientes operaciones: Insertar 32, Insertar 25, Insertar 42, Insertar 44, Eliminar 15. Mostrar la estructura del árbol después de la inserción de 44, y después de eliminar 15.



3.10. Un nodo de un árbol trie se puede representar mediante un array que asocia un puntero a cada carácter, como una lista de pares (carácter, puntero) o, en general, mediante cualquier estructura que asocie valores de un tipo (en este caso caracteres) con valores de otro (punteros). Esto es lo que se llama una asociación o correspondencia (de caracteres a punteros a nodos). ¿Qué otras estructuras de representación podrían usarse en los nodos para eliminar los problemas de representar con arrays (desperdicio de memoria) y de las listas (búsquedas lentas dentro de la lista)? Justificar la respuesta y mostrar cuál sería la eficiencia de la operación Miembro aplicada sobre el trie.

3.11. Utilizando la estructura de representación para relaciones de equivalencia, con equilibrado de nodos y compresión de caminos, mostrar los árboles resultantes tras aplicar las siguientes operaciones, suponiendo que el conjunto universal contiene 9 elementos (de 1 a 9):

Unión(2, 3), Unión(4, 5), Unión(7, 6), Unión(4, 2), Unión(9, 4), Búsqueda(3), Unión(7, 4), Unión(1, 5).

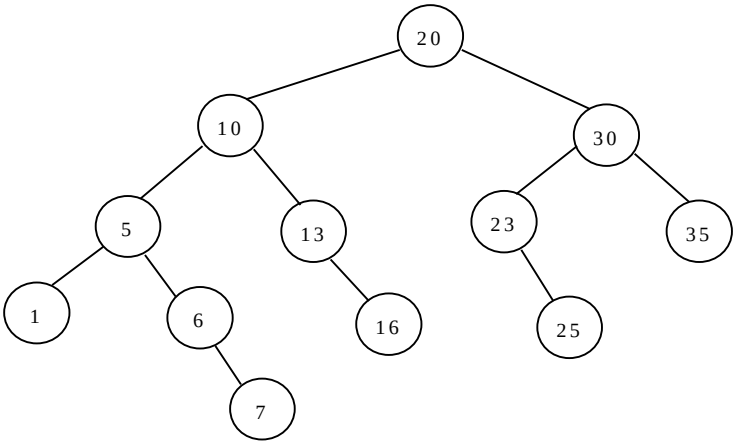
3.12. Mostrar los pasos de ejecución y la estructura de árbol resultante al aplicar las siguientes operaciones sobre un árbol AVL inicialmente vacío:

Inserta(8), Inserta(20), Inserta(12), Inserta(5), Inserta(35), Inserta(40), Inserta(7), Elimina(35), Inserta(10), Elimina(20), Elimina(5).

3.13. * Mostrar un ejemplo de árbol AVL donde al eliminar un elemento dado se requiera más de un rebalanceo para reequilibrar el árbol. Mostrar el árbol antes y después de la eliminación. (Ojo, no se pide un ejemplo de rotación doble, sino de más de un rebalanceo).

3.14. Definir el TDA **Relación de equivalencia**, de un tipo genérico, mediante alguno de los métodos de especificación formal vistos en clase. Añadir operaciones para quitar elementos de una clase de equivalencia, y ponerlos en una clase nueva o en una clase distinta. Implementar las operaciones anteriores, con la estructura de representación vista en clase.

3.15. Dada la siguiente estructura de árbol, comprobar si se trata de un árbol AVL. En caso afirmativo, ¿cómo se puede comprobar si se trata del peor caso de esta

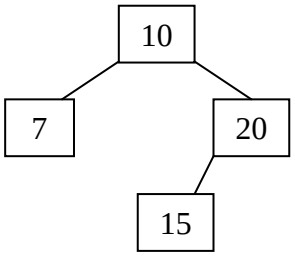


estructura (en cuanto a máximo desequilibrio)? En caso contrario, ¿cómo se puede reequilibrar para convertirlo en un árbol AVL?

3.16. Los árboles AVL son utilizados como un método para representar conjuntos ordenados. Proponer un esquema (en pseudocódigo) para implementar las operaciones *Unión*, *Intersección* y *Diferencia*. Comparar la eficiencia de estas operaciones con la conseguida en otras implementaciones de conjuntos ordenados.

3.17. Representar gráficamente las modificaciones realizadas en un árbol binario de búsqueda cuando se le aplica una rotación doble a la derecha. Implementar esta operación utilizando las rotaciones simples y sin utilizarlas (accediendo directamente a los nodos). Suponiendo que se aplica esta operación a un árbol AVL cualquiera, ¿el árbol resultante conserva siempre la propiedad de ser un árbol AVL? Comprobarlo haciendo un cálculo de las alturas de los subárboles.

3.18. * Para el árbol AVL mostrado abajo, aplicar las siguientes inserciones de elementos: Insertar 27, Insertar 22, Insertar 25. En caso de desequilibrio, decir de qué tipo es y cuál es la operación que se aplica para solucionarlo.



3.19. En la operación *Inserta*, para añadir un elemento nuevo a un árbol AVL, sabemos que como máximo se realizará un rebalanceo a una altura determinada. Aprovechando esta propiedad, modificar el procedimiento de inserción para que no se compruebe la condición de equilibrio después de haber realizado una reestructuración. ¿Cuál es la mejora que se obtiene?

3.20. Sintetizar en una tabla los posibles casos de desequilibrio (6 en total) que se pueden dar al eliminar un elemento de un árbol AVL. Para cada caso indicar la condición que se debe cumplir y la operación a aplicar. Expresar estas condiciones y las operaciones asociadas, con la estructura de representación vista en clase, suponiendo que el desequilibrio se detecta en un nodo A.

3.21. Suponiendo una estructura de árboles B de orden p = 5, mostrar cómo sería el resultado de aplicar las siguientes inserciones sobre un árbol inicialmente vacío:
9, 4, 15, 1, 3, 20, 24, 35, 17, 18, 19, 40, 41, 50, 36, 22, 21, 10, 0.

3.22. Dado el árbol B resultante del ejercicio anterior, mostrar el resultado de aplicar las siguientes eliminaciones:
19, 40, 21, 4.

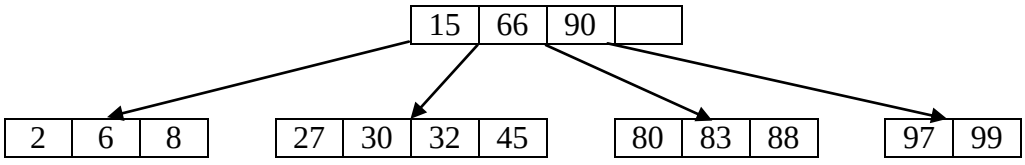
3.23. Proponer una estructura de representación para un árbol B de orden p, con elementos de tipo entero. Para esta estructura escribir procedimientos para las operaciones *Miembro(elemento, árbol_B)* e *Inserta(elemento, árbol_B)*. ¿Qué se debe tener en cuenta si los nodos están almacenados en disco y no en memoria?

3.24. Escribir un procedimiento en pseudocódigo para la operación de eliminación de un elemento en un árbol B. El procedimiento debe tratar todos los casos posibles de eliminación en un nodo hoja o interno.

3.25. * En un árbol B de orden p = 5, inicialmente vacío, introducimos los siguientes elementos: 5, 2, 12, 28, 4, 1, 3, 7, 3, 9, 10. Mostrar la estructura del árbol después de las inserciones de los elementos 7 y 10.

3.26. * En una base de datos de personas, necesitamos acceder a los datos de una persona por su nombre o por su número de DNI. Para ello nos creamos dos estructuras de representación que referencian a los mismos datos: un árbol B para los nombres y una tabla hash donde las claves son los números de DNI. Señalar las principales ventajas e inconvenientes de esta forma de representación.

3.27. * Suponer el árbol B de orden **p** = 5, mostrado abajo. Elegir varios elementos del árbol y eliminarlos hasta que el número de entradas de la raíz disminuya en 1. Mostrar el resultado después de cada eliminación.



3.28. * Queremos construir una variante intermedia entre los árboles AVL y los árboles binarios de búsqueda no balanceados. Para ello, definimos los árboles BWM igual que los árboles AVL, pero la condición de balanceo es que la altura de los subárboles de cada nodo puede diferir como máximo en 2. Exponer las principales ventajas e inconvenientes de los árboles BWM respecto a los AVL. Mostrar el peor caso (número mínimo de nodos) de árbol BWM para altura 5.

3.29. Dada la definición de árboles BWM del ejercicio anterior, comprobar si podemos aplicar el mismo análisis de casos que para los árboles AVL (pero ahora con la condición de diferencia de altura 2), en el caso de ocurrir un desequilibrio del árbol en una inserción. En caso afirmativo, mostrar la estructura del árbol BWM tras la inserción de los siguientes elementos: 11, 10, 4, 5, 2, 6, 7, 9, 8.

3.30. Mostrar el árbol AVL resultante después de insertar los mismos elementos del ejercicios anterior, es decir: 11, 10, 4, 5, 2, 6, 7, 9, 8. Comparar el número de rebalanceos necesarios con el número de rebalanceos requerido para el caso de árboles BWM. ¿Era previsible el resultado esperado? ¿Por qué?

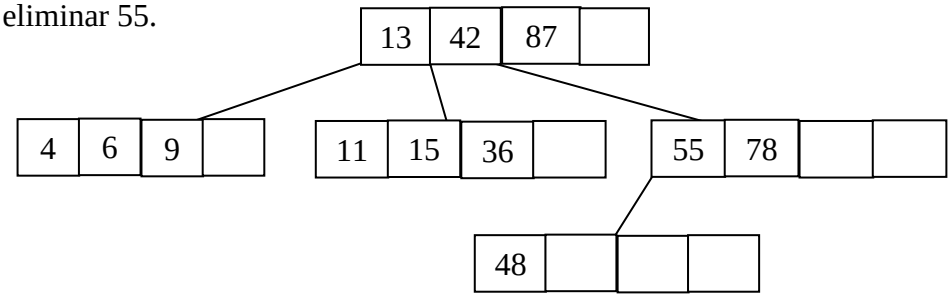
3.31. * Comentar razonadamente cómo puede ser utilizado un árbol AVL para ordenar una secuencia de elementos en un tiempo O(n log n).

3.32. * Mostrar el árbol B de orden p=5, resultante de insertar en un árbol inicialmente vacío los siguientes elementos: 23, 12, 82, 14, 20, 9, 50, 32, 43. Mostrar los pasos intermedios en los que se modifique la estructura del árbol. El resultado obtenido ¿depende del orden en que han sido insertados los elementos? En caso afirmativo, mostrar un orden que dé lugar a otro árbol y mostrar ese árbol.

3.33. * En un árbol TRIE representamos palabras en dos idiomas (por ejemplo español e inglés). Queremos añadir a cada palabra una definición de su significado y la traducción al otro idioma. Describir la estructura de datos, mostrando las definiciones de los tipos en notación Pascal. Tener en cuenta que algunas palabras pueden tener significado en los dos idiomas (por ejemplo, can, mete, sin).

3.34. * En un árbol B de orden **p** = 3 inicialmente vacío, insertamos los elementos: 10, 8, 24, 12, 17, 15 y 13. Mostrar el árbol B después de cada inserción que modifique la estructura del árbol y el resultado final. Mostrar también un árbol binario de búsqueda perfectamente balanceado que contenga los mismos elementos.

3.35. * El siguiente dibujo representa un árbol B de orden **p** = 5. ¿Es correcto el estado del árbol o hay algún error? En caso de haber errores, indicar cuáles, por qué y arreglarlos eliminando los elementos que los causan. Después, mostrar el estado del árbol tras aplicar cada una de las siguientes operaciones: eliminar 13, eliminar 6, eliminar 55.



3.36. * Consideremos la siguiente definición de árboles trie, vista en clase.

Definición de los tipos:	Operaciones sobre el tipo:
dominio = (‘A’, ‘B’, ..., ‘Z’, ‘$’);	**Asigna** (n: nodo; caract: dominio; ptr: nodo); **Valor_de** (n: nodo; caract: dominio): nodo;

nodo = array [dominio] of ^nodo; **trie** = ^nodo;	**Toma_nuevo** (n: nodo; caract: dominio); **Anula** (n: nodo);

a) Escribir un algoritmo en Pascal para resolver el siguiente problema: dada una cadena **prefijo**, listar todas las palabras del árbol que empiecen por **prefijo**. Por ejemplo, si **prefijo** = "res", podrían aparecer por pantalla: "res, resaber, resabiar...". Suponer que tenemos la función **longitud(cadena)** para conocer la longitud de una cadena y accedemos a las letras con **cadena[1]**, **cadena[2]**, **cadena[3]**, ...

b) ¿Qué ocurre si en lugar de buscar por prefijos queremos buscar por sufijos? Dar una idea de cómo resolver el problema en ese caso.