Ejercicios

3.1. Explica 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}. Explica cómo afectaría a la memoria y al tiempo de ejecución, con las representaciones de nodos con arrays y con listas enlazadas. ¿Qué otras alternativas existen para representar palabras con acento?

3.3. (EX) Muestra el resultado de insertar los siguientes elementos en un árbol trie: jurel, julepe, jueves, julepes, giles. Elige un tipo de representación para el árbol y haz una estimación aproximada de la memoria ocupada (para este ejemplo).

3.4. Escribir los procedimientos **Inserta**, **Consulta**, **Anula** y **TomaNuevo** 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. Describe las principales características de la estructura de tries en esta aplicación y (sin detallar excesivamente) como sería manejado el trie en el proceso de compresión.

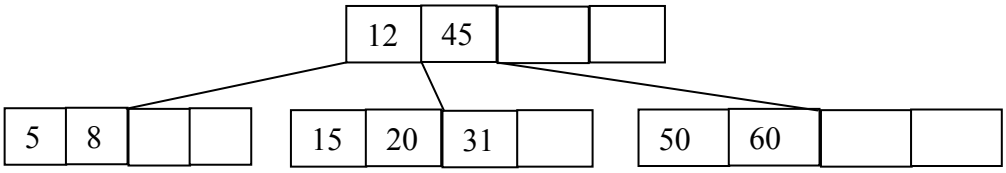
3.6. Tal y como hemos visto en clase, los tries se utilizan para representar conjuntos o diccionarios. Teniéndolo en cuenta, ¿es correcto definir un trie como un tipo de datos abstracto? Justifica la respuesta, en función de las propiedades de un TAD. En caso afirmativo, ¿en qué modelo matemático se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

3.7. (TG sec. 4.1.4) Realiza una tabla comparativa en la que se muestren 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 arrays y con listas. Exprésala 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. Comenta 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. (EX) 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. En clase hemos estudiado la representación de nodos trie mediante arrays y mediante listas. Teniendo en cuenta que los nodos tries son diccionarios (*clave*: carácter; *valor*: punteros a nodos), ¿qué otras estructuras de representación podrían usarse en los nodos, para eliminar los problemas de representación con arrays (desperdicio de memoria) y de las listas (búsquedas lentas dentro de la lista)? Justifica la respuesta y muestra cual 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, muestra 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).

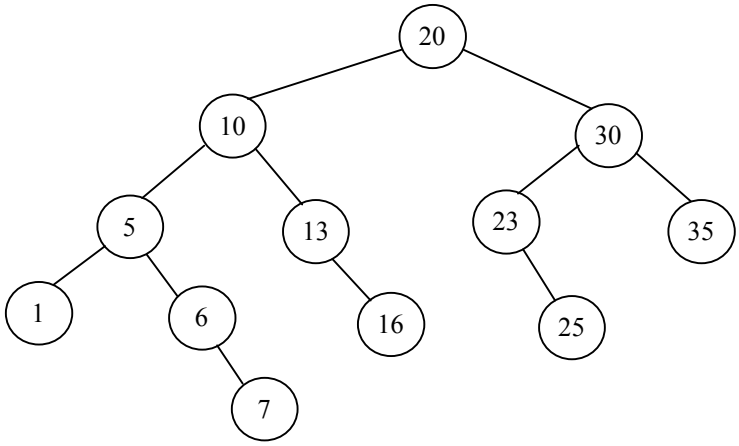
3.13. (EX) Muestra un ejemplo de árbol AVL donde al eliminar un elemento dado se requiera más de un rebalanceo para reequilibrar el árbol. Muestra el árbol antes y después de la eliminación. (Ojo, no se pide un ejemplo de rotación doble, sino de más de un rebalanceo).

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

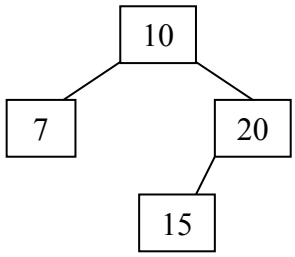
3.15. Representa gráficamente las modificaciones realizadas en un árbol binario de búsqueda cuando se le aplica una rotación doble a la derecha. Implementa esta operación utilizando las rotaciones simples y sin utilizarlas (accediendo directamente a los nodos). Suponiendo que se aplica esta operación a un árbol AVL

cualquiera, ¿el árbol resultante conserva siempre la propiedad de ser un árbol AVL? Compruébalo haciendo un cálculo de las alturas de los subárboles.

3.16. 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?



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



3.18. 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, modifica 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?

3.19. Sintetiza 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 indica la condición que se debe cumplir y la operación a aplicar. Expresa estas condiciones y las operaciones asociadas, con la estructura de representación vista en clase, suponiendo que el desequilibrio se detecta en un nodo A.

3.20. Suponiendo una estructura de árboles B de orden p = 5, muestra como 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.

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

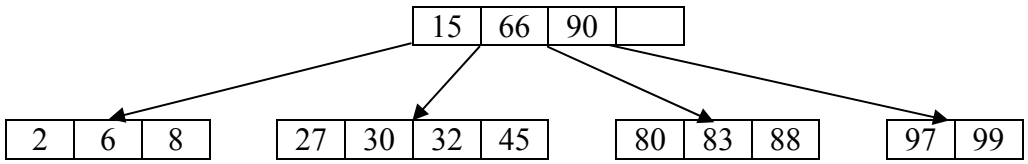
3.22. Propón una estructura de representación para un árbol B de orden **p**, con elementos de tipo entero. Para esta estructura escribe procedimientos para las operaciones **Miembro(**elemento, árbol_B**)** e **Inserta(**elemento, árbol_B**)**. ¿Qué se debe tener en cuenta si los nodos están almacenados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

3.23. Escribe 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.

3.24. (EX) 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. Muestra la estructura del árbol después de las inserciones de los elementos 7 y 10.

3.25. (EX) En una base de datos de personas, necesitamos acceder a los datos de una persona por su nombre ó 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. Señala las principales ventajas e inconvenientes de esta forma de representación.

3.26. (EX M01) Suponer el árbol B de orden **p** = 5, mostrado abajo. Elige varios elementos del árbol y elimínalos hasta que el número de entradas de la raíz disminuya en 1. Muestra el resultado después de cada eliminación.



3.27. (TG 4.5, EX S01) 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. Expón 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.

3.28. (EX M05) 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 desbalanceo 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.

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

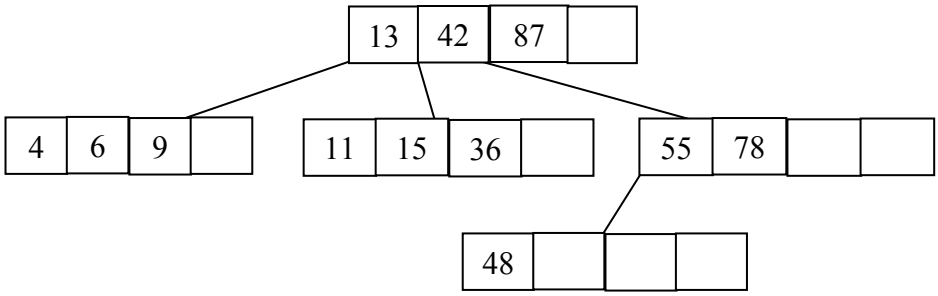
3.30. (EX) Comentar razonadamente cómo puede ser utilizado un árbol AVL para ordenar una secuencia de elementos en un tiempo O(n log n).

3.31. (EX D01) 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. Muestra 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.

3.32. (TG 4.1, EX D01) 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 pseudocódigo. Tener en cuenta que algunas palabras pueden tener significado en los dos idiomas (por ejemplo, can, mete, conductor, sin).

3.33. (EX M02) 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.

3.34. (TG 4.8, EX S02) 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, indica cuáles, por qué y arréglalos eliminando los elementos que los causan. Después, muestra el estado del árbol tras aplicar cada una de las siguientes operaciones: eliminar 13, eliminar 6, eliminar 55.



3.35. (TG 4.2, EX S02) Considera la siguiente definición de árboles trie, vista en clase.

Definición de los tipos:	Operaciones sobre el tipo:
dominio = ('A', 'B', ..., 'Z', '$'); **nodo** = array [dominio] of ^nodo; **trie** = ^nodo;	**Asigna** (n: nodo; caract: dominio; ptr: nodo); **Valor_de** (n: nodo; caract: dominio): nodo; **Toma_nuevo** (n: nodo; caract: dominio); **Anula** (n: nodo);

a) Escribe 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.

3.36. (TG 4.7, EX D02) Un árbol B de orden **p** = **n**+1 se usa para almacenar valores enteros. La definición del tipo de datos es la siguiente:

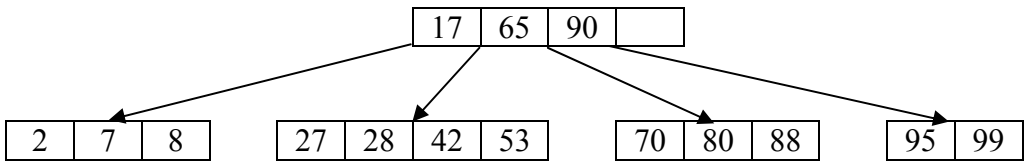
```
tipo
    nodo= registro
            valores: array [1..n] de entero
            punteros: array [0..n] de Puntero[nodo]
    finregistro
    ArbolB= Puntero[nodo]
```

Si una posición está vacía, entonces el valor correspondiente vale -1 y el puntero vale NULO. Para los nodos hoja, todos los punteros tienen valor NULO. Escribe un procedimiento para recorrer de forma ordenada (de menor a mayor) todos los elementos almacenados en un árbol B dado, usando esta definición.

3.37. (TG 4.4, EX M03) En un árbol AVL inicialmente vacío insertamos (por este orden) los siguientes elementos: 8, 13, 10, 5, 11, 6, 7; y después eliminamos el 13. Mostrar la estructura del árbol antes y después de cada operación que requiera un rebalanceo. Indicar también el caso que ocurre y cuál es el tipo de rebalanceo aplicado.

3.38. (EX S03) En una aplicación de base de datos representamos la información mediante árboles AVL, almacenados en disco. Cada sector de disco ocupa 512 bytes, en los cuales caben hasta 64 nodos del árbol. Estamos interesados en contar las operaciones de escritura en disco, que son las más costosas. Determinar, de manera razonada, cuántas escrituras (es decir, operaciones de escribir en bloques de disco distintos) se pueden realizar como máximo en el peor caso en una inserción de un elemento en el árbol AVL.

3.39. (EX) Sobre el árbol B de orden **p** = 5 mostrado abajo, insertar los elementos 50, 10 y 13. Después eliminar 53 y 8.



3.40. (EX) Un corrector ortográfico interactivo utiliza un trie para representar las palabras de su diccionario. Queremos añadir una función de **auto-completar** (al estilo de la tecla TAB en Linux): cuando estamos a medio de escribir una palabra, si sólo existe una forma correcta de continuarla entonces debemos indicarlo. Escribir una función **AutoCompletar** (**t**: trie; **pal**: cadena): cadena, que dado el árbol trie **t** y la palabra **pal** devuelve la forma de auto-completar la palabra. Por ejemplo, para la llamada **AutoCompletar**(t, “groen”) devolvería “land”, ya que podemos tener "groenlandia" o "groenlandés". Si hay varias formas, devolvería la cadena vacía. Por ejemplo, **AutoCompletar**(t, “ma”) devolvería “”.

Suponer que tenemos la operación: **Consulta**(**n**: trie; **c**: caracter): trie (que devuelve el valor de puntero asociado al carácter **c** en la raíz del trie **n**), y un iterador: **Para todo carácter c hijo del nodo n hacer** (que itera sobre todos los hijos de la raíz del trie **n**).

3.41. (EX J04) En una tabla de dispersión abierta hacemos que las cubetas de la tabla en lugar de ser listas de elementos sean árboles AVL.

a) Haz una estimación de cuál sería el orden de complejidad de esta estructura para la operación de consulta en los casos mejor, peor y promedio.
b) Señala las principales ventajas e inconvenientes de esta estructura respecto a la dispersión abierta normal y respecto a los árboles AVL.
c) Suponer que usamos la anterior estructura para insertar los elementos: 16, 72, 826, 1016, 12, 42, 623, 22, 32, siendo **B**= 10 cubetas y **h(x)** = **x** mod 10. Mostrar el resultado obtenido, señalando los pasos más destacados.

3.42. (EX J04) En la estructura de relaciones de equivalencia mediante punteros al padre, mejorada con balanceo de árboles y compresión de caminos, vimos que en las raíces se almacena la altura del árbol correspondiente, en valores negativos. Pero la altura no se recalcula al producirse una compresión del árbol, por lo que podría contener valores incorrectos. Escribir un algoritmo que compruebe si las alturas de los árboles son correctas, actualizándolas en caso necesario. ¿Cuál es el orden de complejidad de este algoritmo?

3.43. (EX S04) En un árbol B de orden **p** = 5 se insertan las claves 1, 2, 3, ..., **n**, en ese mismo orden. ¿Qué claves originan la división de un nodo? A la vista del resultado, ¿se puede generalizar el comportamiento para un árbol B de orden **p** cualquiera? ¿Qué claves hacen que la altura del árbol crezca, para el caso de **p** = 5? Obtener fórmulas generales, y justificar las respuestas mostrando varios ejemplos de árboles B con diferentes alturas.

3.44. (EX S04) Suponer que estamos trabajando con los tipos de datos abstractos **ArbolTrie** y **NodoTrie**, con operaciones como las vistas en clase:

Inserta (var n: NodoTrie; c: carácter; ptr: NodoTrie)
Consulta (n: NodoTrie; c: carácter): NodoTrie
Anula (var n: NodoTrie)
TomaNuevo (var n: NodoTrie; c: carácter)

Escribir una operación que dado un árbol trie **t**, una palabra **p** y un entero **n**, escriba todas las palabras del árbol que empiezan por **p** y sean de longitud **n**. No suponer ninguna estructura de datos concreta para los nodos del trie, usar sólo las operaciones abstractas anteriores. La cabecera de la función debe ser de la forma:

MuestraPalabras (t: NodoTrie; p: cadena; n: entero)

Usa las operaciones sobre cadenas que se consideres necesarias.

3.45. (EX D04) En un árbol AVL inicialmente vacío queremos insertar los siete siguientes elementos: 25, 10, 5, 20, 35, 15, 30. Encontrar un orden de inserción de los elementos que requiera el máximo número de operaciones de rotación. Encontrar

también un orden que necesite el mínimo posible de rotaciones. En ambos casos, mostrar los árboles antes y después de cada operación de rotación, e indicar de qué tipo es. En general, en un AVL cualquiera con *n* nodos, la diferencia entre el caso mejor y peor, ¿tiene influencia en el orden de complejidad de las operaciones sobre el árbol? Justificar brevemente.

3.46. (EX M05) Programar una operación que encuentre todas la palabras que aparecen en un árbol trie y que coincidan con una cadena de caracteres pasada como parámetro. Esta cadena puede incluir un carácter especial "?" que significa que en esa posición puede haber cualquier carácter. Por ejemplo, la cadena "c?s??" puede corresponder a "casas", "costa", "caspa", "cosas", etc. La función debe ser lo más eficiente posible, evitando moverse por ramas del árbol donde no pueda haber solución. Suponer que existe sobre el tipo **trie** una operación **Consulta (n: trie, c: carácter): trie** y un iterador del tipo **para cada carácter c hijo del nodo n hacer**.

3.47. (EX J05) Dada la siguiente secuencia de claves: 100, 29, 71, 82, 48, 39, 101, 22, 46, 17, 3, 20, 25, 10,

a) dibujar el AVL correspondiente a la inserción de estos elementos, indicando en los pasos en que sea necesario el tipo de rotación que se aplica y cómo se detecta el desequilibrio correspondiente;
b) eliminando consecutivamente diferentes claves del árbol anterior, buscar, mostrar y resolver al menos un desequilibrio que requiera una rotación simple y otro que requiera una doble.

3.48. (EX J05) Un poeta joven tiene dificultad para encontrar rimas. Como es un poeta-informático decide hacer una operación que le facilite la búsqueda de rimas. Tiene una serie de palabras almacenadas en un árbol trie, y quiere escribir un algoritmo que reciba el trie, una palabra con la que quiere rimar, y un entero que indica la letra de la palabra a partir de la que se rima (la rima será consonante). El algoritmo debe escribir las palabras que riman con la palabra de entrada. Programar esta operación. Habrá que hacerla sobre el trie original (no se podrá formar un trie invirtiendo las palabras en el trie original). Suponer que existe sobre el tipo **trie** una operación **Consulta (n: trie, c: carácter): trie** y un iterador del tipo **para cada carácter c hijo del nodo n hacer**. No se debe suponer ninguna estructura de datos concreta para el trie, usar sólo las operaciones anteriores.

3.49. (EX S05) Considera la estructura de relaciones de equivalencia vista en clase, mediante árboles de punteros al padre, usando las técnicas de balanceo de árboles y compresión de caminos. Mostrar un ejemplo, con sucesivas operaciones de unión sobre una relación inicialmente vacía, donde uno de los árboles crezca hasta profundidad 3. Se deben mostrar en cada paso los valores de la tabla y la estructura de árboles asociada.

3.50. (EX S05) Un árbol B de orden **p** = **n**+1 se usa para almacenar valores enteros. La definición del tipo de datos es la siguiente:

```
tipo
    nodo= registro
            valores: array [1..n] de entero
```

```
        punteros: array [0..n] de Puntero[nodo]
    finregistro
    ArbolB= Puntero[nodo]
```

Si una entrada del nodo está vacía, el valor correspondiente vale -1. Para los nodos hoja, todos los punteros tienen valor NULO. Escribir un procedimiento para recorrer de forma ordenada (de menor a mayor) todos los elementos de un árbol B que estén entre dos valores dados **v1** y **v2** (con **v1**<**v2**), ambos inclusive, usando los tipos definidos. La cabecera del procedimiento será:
operación Recorrido (raíz: ArbolB; v1, v2: entero)

3.51. (EX F06) Programar una operación de unión entre dos árboles trie. Dados dos árboles trie, la operación debe añadir a uno de ellos todas las palabras que aparecen en el otro. Se deben utilizar las operaciones genéricas sobre nodos trie: **Consulta (n: trie, c: carácter): trie** (devuelve el hijo del nodo trie **n** para el carácter **c**, o NULO si no existe), **Inserta (n: trie, c: carácter, m: trie)** (añadir al nodo trie **n** el hijo **m** asociado al carácter **c**), **NuevoTrie: trie** (devuelve un nuevo nodo vacío) y un iterador del tipo **para cada carácter c hijo del nodo n hacer**.

3.52. (EX) Suponer una estructura de relaciones de equivalencia mediante punteros al padre. Mostrar un ejemplo de operaciones donde se vea la diferencia entre usar la estrategia de balanceo de árboles y no usarla. Mostrar otro ejemplo que muestre la diferencia entre usar compresión de caminos o no. En ambos casos, mostrar la evolución de las estructuras para cada operación, explicarlo y razonar cuál es mejor.

3.53. (EX) Un árbol trie contiene la conjugación en tiempo presente del verbo ver (“veo”, “ves”, “ve”, etc.). Mostrar la estructura del árbol resultante. Indicar cómo quedaría esa estructura implementando los nodos del trie con las dos técnicas vistas en clase: con listas de nodos y con arrays de punteros. Hacer una estimación de la memoria ocupada en cada caso, introduciendo las variables oportunas y explicando claramente la obtención del resultado. ¿Cuál es más adecuada para este ejemplo concreto?