Laboratorio A.E.D. Laboratorio 5

Guillermo Román
guillermo.roman@upm.es

Lars-Åke Fredlund
larsake.fredlund@upm.es

Manuel Carro
manuel.carro@upm.es

Marina Álvarez
marina.alvarez@upm.es

Julio García
juliomanuel.garcia@upm.es

Tonghong Li
tonghong.li@upm.es

Sergio Paraíso
sergio.paraiso@upm.es

Juan José Moreno
juanjose.moreno@upm.es

Luis Miguel Danielsson
lm.danielsson@alumnos.upm.es

Normas

- Fechas de entrega y penalización asociada:

Hasta el martes 23 de noviembre, 12:00 horas	0 %
Hasta el miércoles 24 de noviembre, 12:00 horas	20 %
Hasta el jueves 25 de noviembre, 12:00 horas	40 %
Hasta el viernes 26 de noviembre, 12:00 horas	60 %

  Después la puntuación máxima será 0
- Se comprobará plagio y se actuará sobre los detectados.
- Usad las horas de tutoría para preguntar sobre programación – son oportunidades excelentes para aprender.

Entrega

- Todos los ejercicios de laboratorio se deben entregar a través de

  `http://vps142.cesvima.upm.es`

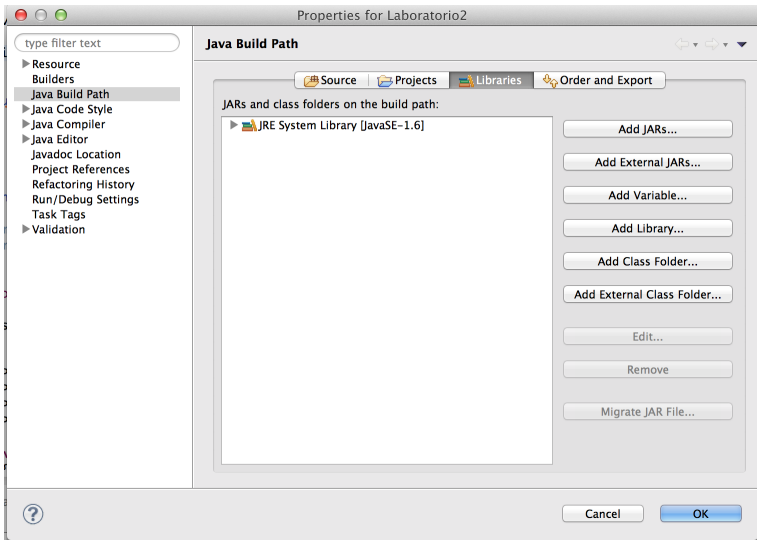
- Los ficheros que hay que subir es `DictImpl.java`.

Configuración previa

- Arrancad Eclipse
- Podéis utilizar cualquier versión reciente de Eclipse. Es suficiente con que instaléis la *Eclipse IDE for Java Developers*.
- Cambiad a "Java Perspective".
- Debéis tener instalado al menos Java JDK 8.
- Cread un proyecto Java llamado `aed`:
  - Seleccionad separación de directorios de fuentes y binarios.
  - **No debéis elegir la opción de crear el fichero** `module-info.java`
- Cread un *package* `es.upm.aedlib.tree` en el proyecto `aed`, dentro de `src`
- Aula Virtual → AED → Laboratorios → Laboratorio 5 → `Laboratorio5.zip`; descomprimidlo
- Contenido de Laboratorio5.zip:
  - `DictImpl.java, Dictionary.java, TesterLab5.java`

Configuración previa

- Importad al paquete `es.upm.aedlib.tree` los fuentes que habéis descargado (`DictImpl.java`, `Dictionary.java`, `TesterLab5.java`)
- Si no lo habéis hecho, añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios).

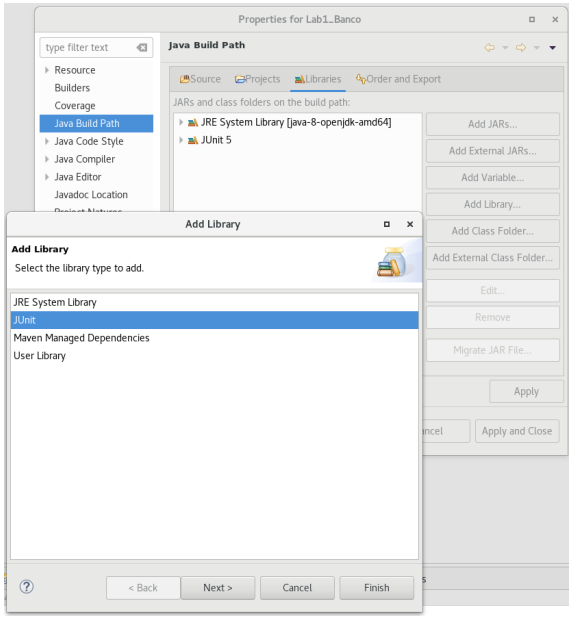


Para ello:

- Project → Properties → Java Build Path. Se abrirá una ventana como la de la izquierda
- Usad la opción "Add External JARs...".
- Si vuestra instalacion distingue `ModulePath` y `ClassPath`, instalad en `ClassPath`

Configuración previa

- Si no lo habéis hecho, añadid al proyecto `aed` la librería `JUnit 5`



Para ello:

- Project → Properties → Java Build Path. Se abrirá una ventana como la de la izquierda;
- Usad la opción "Add Library..." → Seleccionad "Junit" → Seleccionad "JUnit 5"
- Si vuestra instalación distingue `ModulePath` y `ClassPath`, instalad en `ClassPath`

- En la clase `TesterLab5` tenéis las pruebas, para ejecutarlas, abrid el fichero `TesterLab5`, pulsando el botón derecho sobre el editor, seleccionar "Run as..." → "JUnit Test"
- NOTA: Si al ejecutar no aparece la vista "JUnit", podéis incluirla en "Window" → "Show View" → "Java" → "JUnit"

Documentación de la librería aedlib.jar

- La documentación de la API de `aedlib.jar` esta disponible en `http://costa.ls.fi.upm.es/~entrega/aed/docs/aedlib/`
- También se puede añadir la documentación de la librería a Eclipse (*no es obligatorio*): en el "Package Explorer": "Referenced Libraries" → aedlib.jar y elige la opción "Properties". Se abre una ventana donde se puede elegir "Javadoc Location" y ahí se pone como "javadoc location path:"

  `http://costa.ls.fi.upm.es/~entrega/aed/docs/aedlib/`

  y presionar el buton "Apply and Close"

Tarea de hoy: implementar un diccionario

Un diccionario permite:

- Añadir una palabra (`String`) al diccionario.
- Comprobar si un palabra está incluida en el diccionario.
- Borrar una palabra del diccionario.
- Devolver todas las palabras que empiezan por el mismo prefijo, en orden alfabetico.

```
public interface Dictionary {
  public void add(String word);
  public boolean isIncluded(String word);
  public void delete(String word);
  public PositionList<String>
           wordsBeginningWithPrefix(String prefix);
}
```

Ejemplo

```
Dictionary d = new DictImpl();

d.add("hola");
d.isIncluded("hola");    // => true
d.delete("hola");
d.isIncluded("hola");    // => false

d.add("hola");
d.add("buenas");
d.add("hola_todos");
d.add("hola_amigos");

d.wordsBeginningWithPrefix("hola");
                    // => ["hola", "hola_amigos", hola_todos"]
```

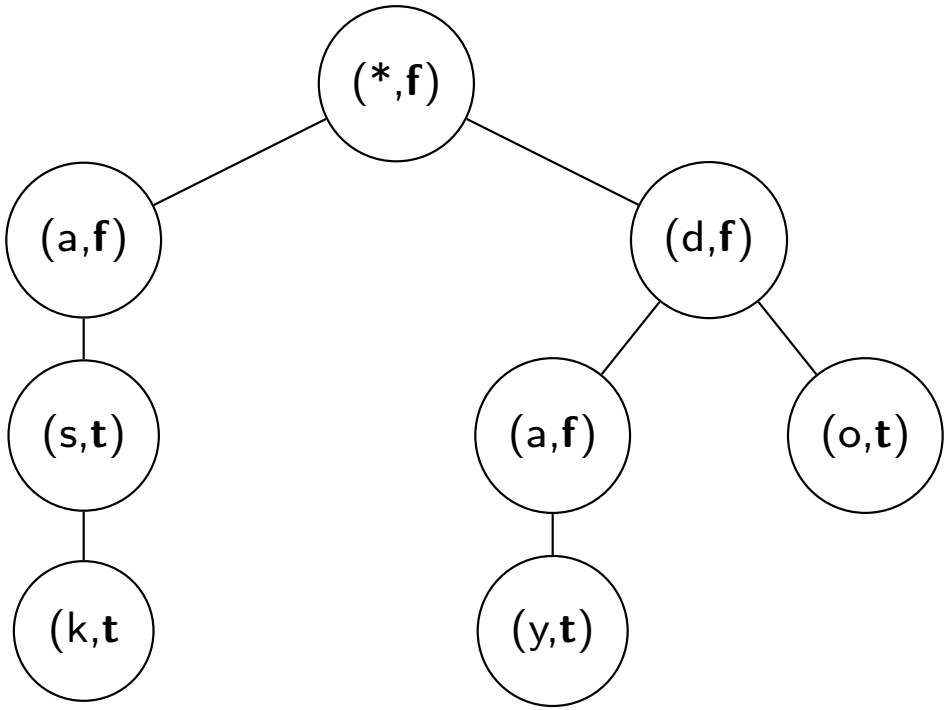
Implementación

- Implementación mediante árbol general.
- Elemento en cada nodo: `Pair<Character,Boolean>`.
- `DictImpl` ya tiene atributo para contener árbol:

```
GeneralTree<Pair<Character,Boolean>> tree;
```

Obligatorio: guardar elementos del diccionario en este atributo.
No se puede cambiar tipo o nombre.

Ejemplo de árbol para representar diccionario

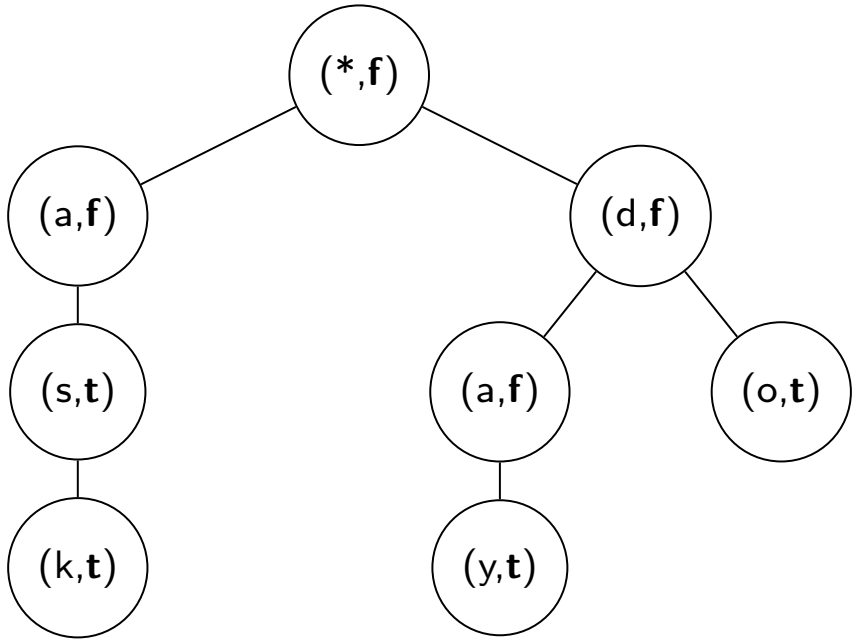


- Un nodo en el árbol tiene un elemento (*carácter*, *booleano*)
- Una palabra *p* está en el diccionario si se cumple que:
  - Hay una secuencia de nodos desde la raíz cuyos caracteres concatenados forman la palabra *p* y
  - El nodo final de esa secuencia tiene el valor booleano **t**.
- Nota: nodo raíz tiene asociado carácter 'ficticio' - no importa cuál es.

Tries

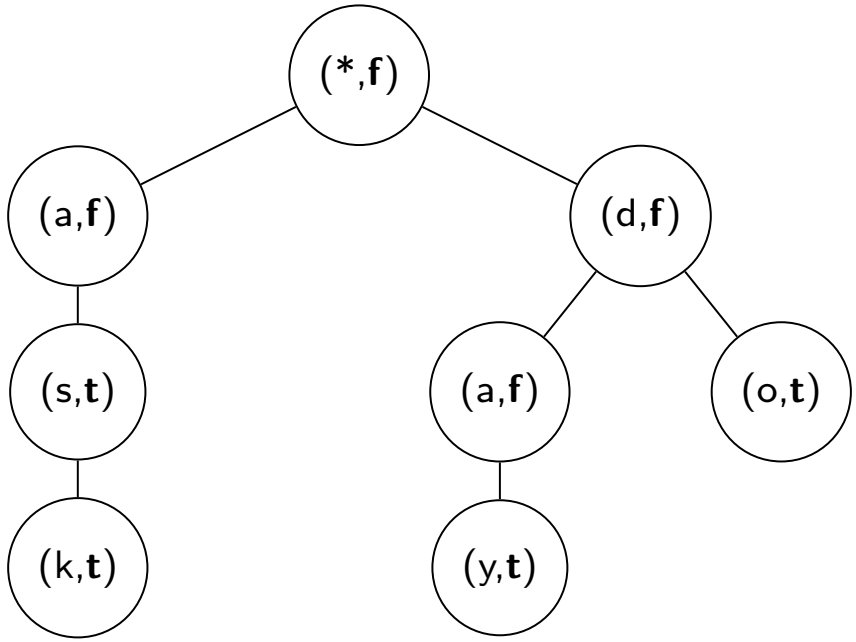
- Este tipo de arbol, que usa caminos para codificar claves, es conocido como un "Trie" o "Prefix Tree".
- Se puede usar "tries" para datos que no son `String`s. P.e., con `Integer`, el camino puede estar definido por los bits o las cifras del `Integer`.
- Se pueden asociar datos a cada final de palabra (clave) — como en un diccionario.
- La búsqueda en "tries" tiene mejor complejidad en el peor caso que las tablas de dispersion: $O(m)$, donde m es la longitud de la clave (en nuestro caso la longitud del `String`).

Ejemplo árbol



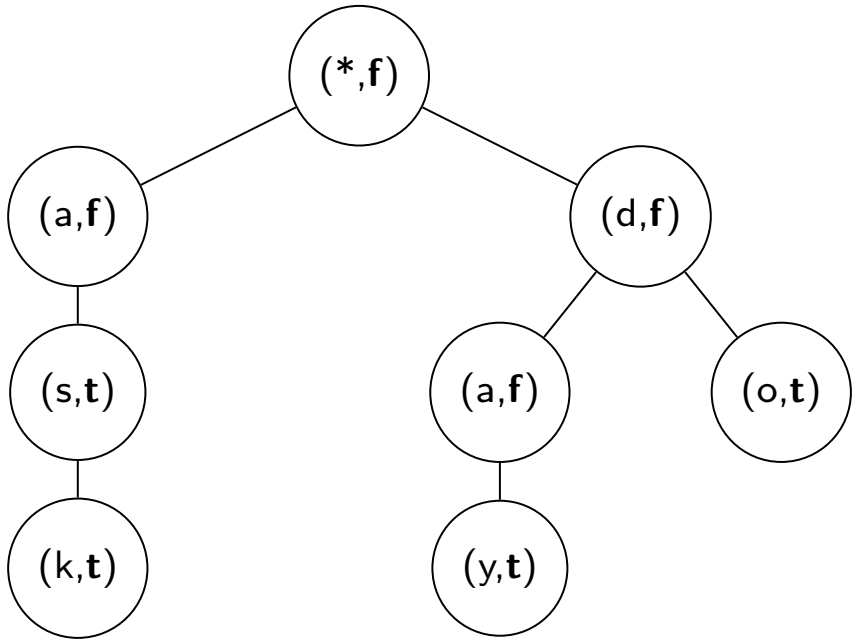
- ¿Qué palabras contiene el diccionario representado con el árbol anterior?

Ejemplo árbol



- ¿Qué palabras contiene el diccionario representado con el árbol anterior? $\equiv$ ¿Que caminos hay que terminan en nodo con valor **t**?

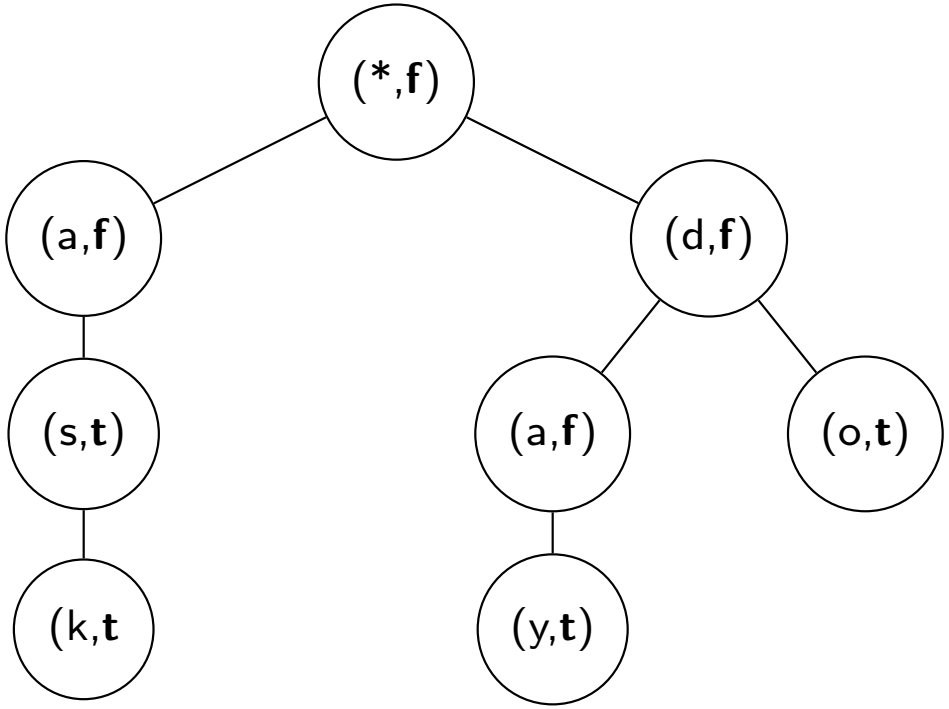
Ejemplo árbol



caminos que terminan en **t**	palabra
(*,**f**) − (a,**f**) − (s,**t**)	as
(*,**f**) − (a,**f**) − (s,**t**) − (k,**t**)	ask
(*,**f**) − (d,**f**) − (a,**f**) − (y,**t**)	day
(*,**f**) − (d,**f**) − (o,**t**)	do

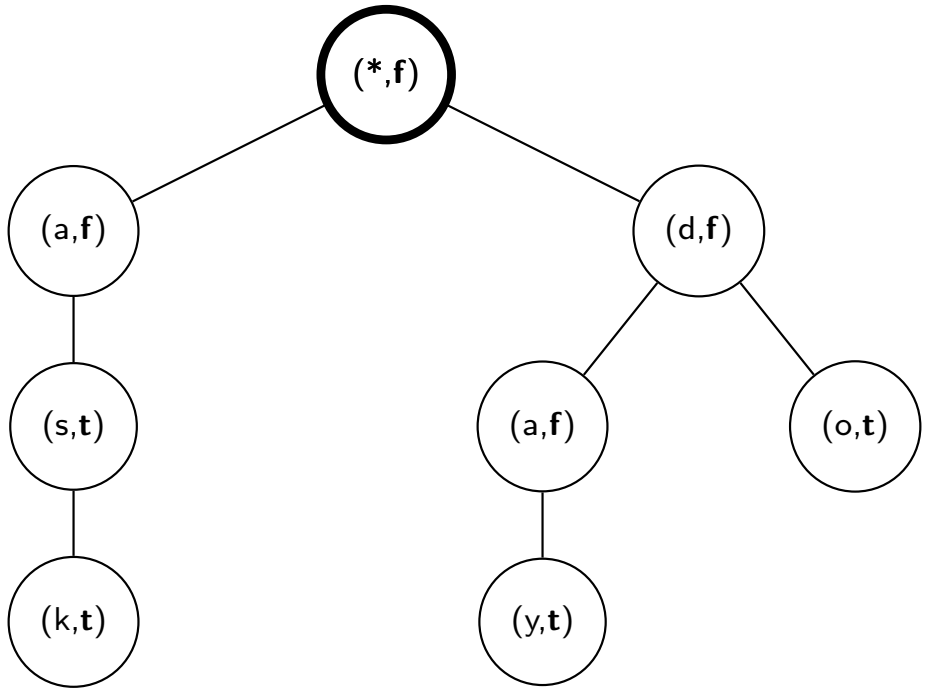
El diccionario contiene las palabras “as”, “ask”, “day” y “do”.

Reglas de construcción de tries



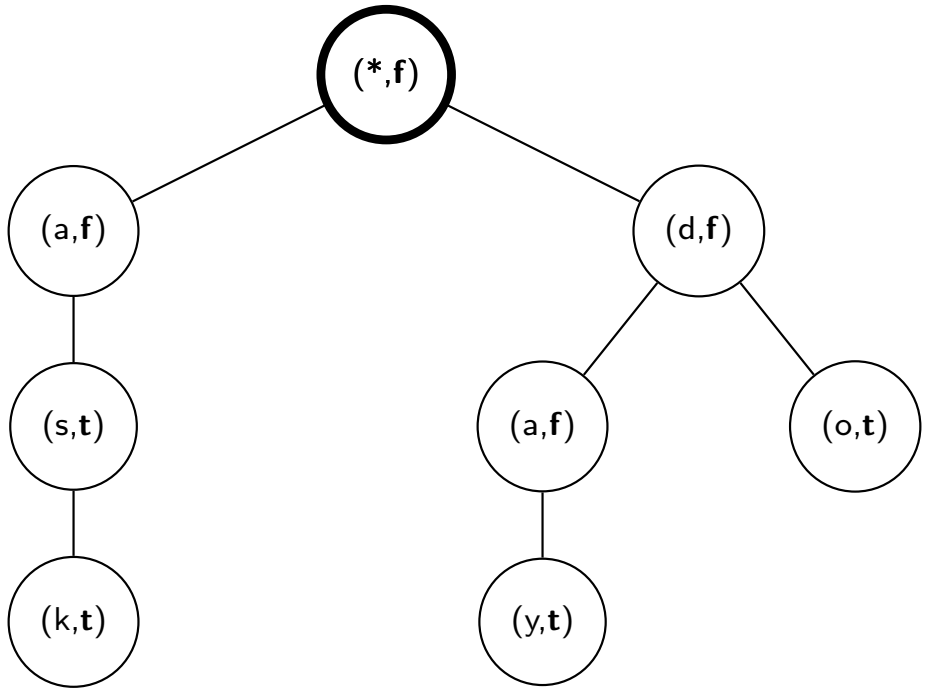
- Un diccionario siempre tiene un nodo raíz especial:
  - Añadido por el constructor de la clase (no hace falta hacerlo).
  - Contiene un elemento con carácter `null` y boolean **f**.
- Un nodo nunca tiene dos hijos con el mismo carácter
- Hijos: ordenados alfabéticamente. Un hijo con carácter "a" aparece antes que un hijo con "b" (acelerar búsquedas).

Insertar palabras en un árbol



Inserción de “done”:

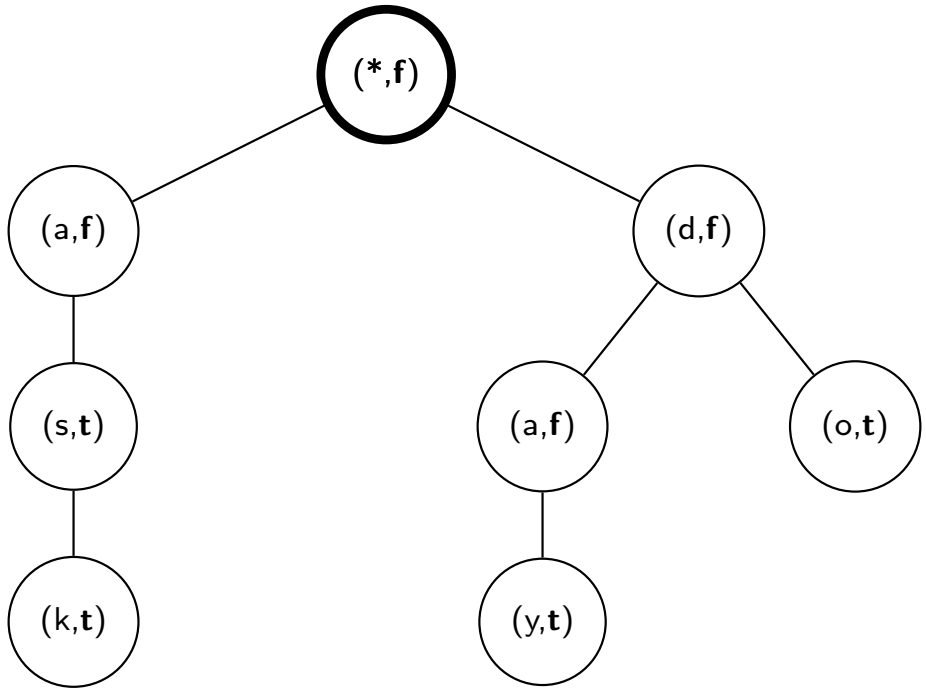
Insertar palabras en un árbol



Inserción de “done”:

- Empezamos en la raiz.

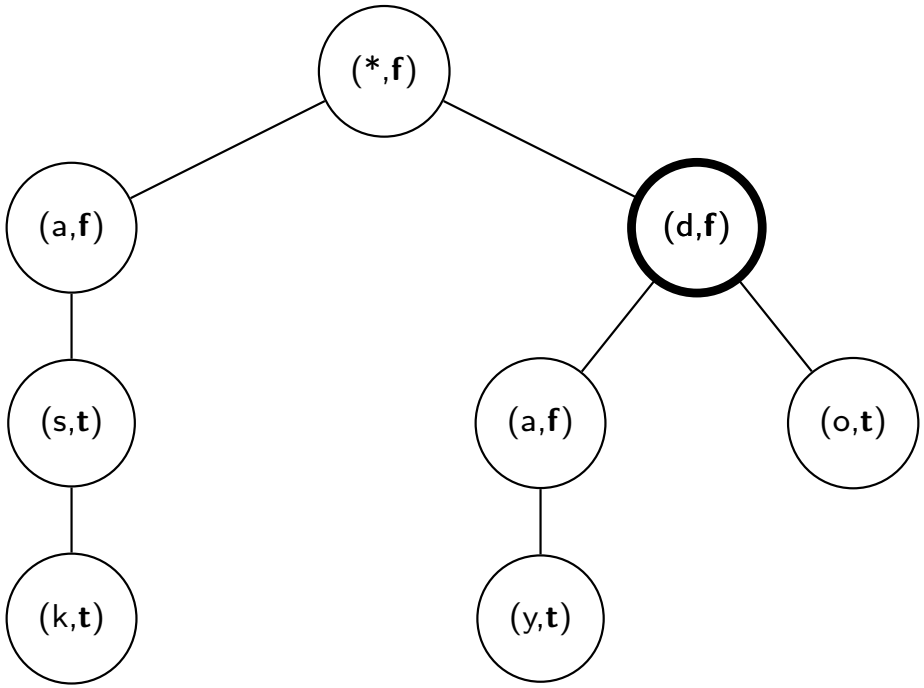
Insertar palabras en un árbol



Inserción de “done”:

- Empezamos en la raiz.
- Buscamos un hijo de la raiz con carácter “d”.

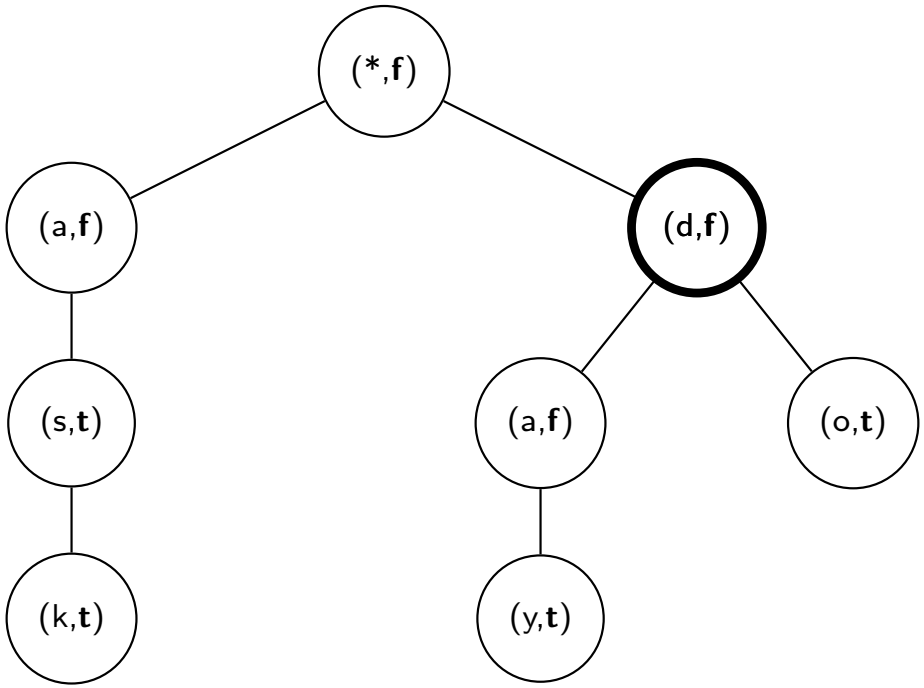
Insertar palabras en un árbol



Inserción de “done”:

- Lo tenemos. Continuamos con la segunda letra “o”.

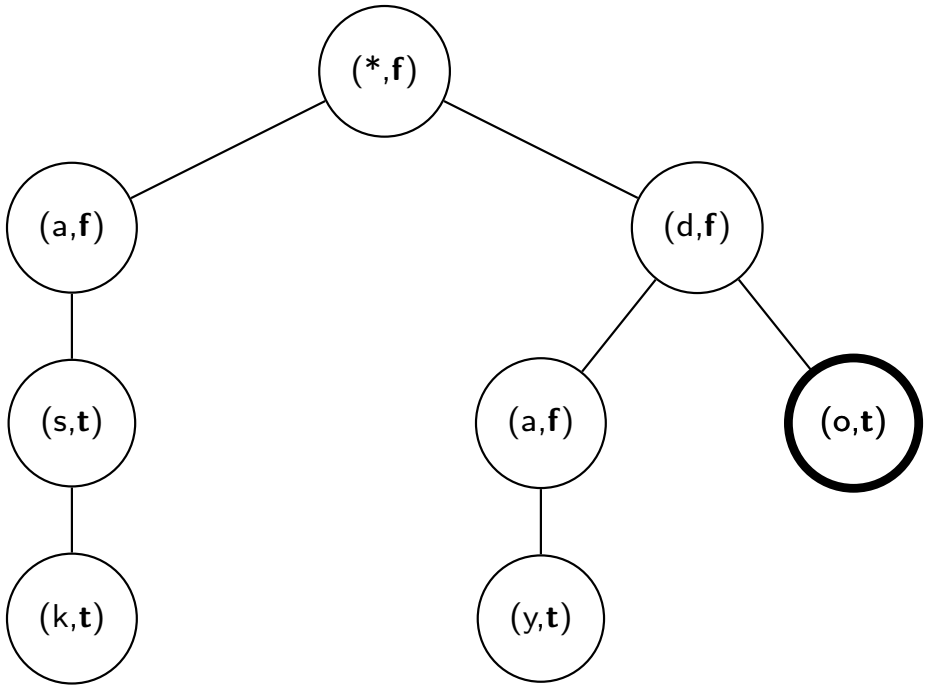
Insertar palabras en un árbol



Inserción de “done”:

- Lo tenemos. Continuamos con la segunda letra “o”.
- Buscamos un hijo con carácter “o”.

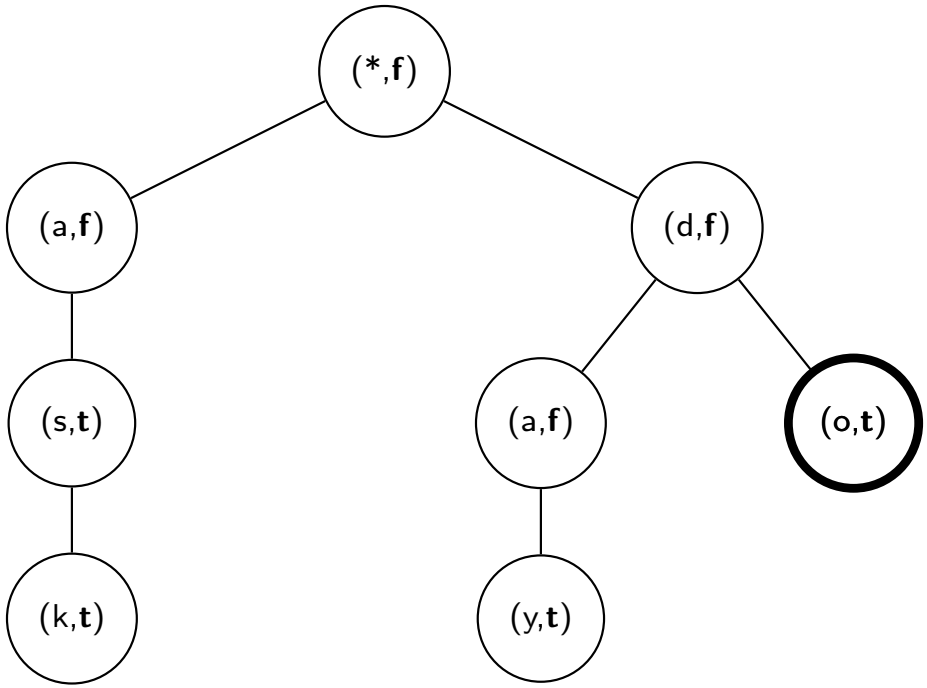
Insertar palabras en un árbol



Inserción de “done”:

- Lo tenemos. Continuamos con la tercera letra “n”.

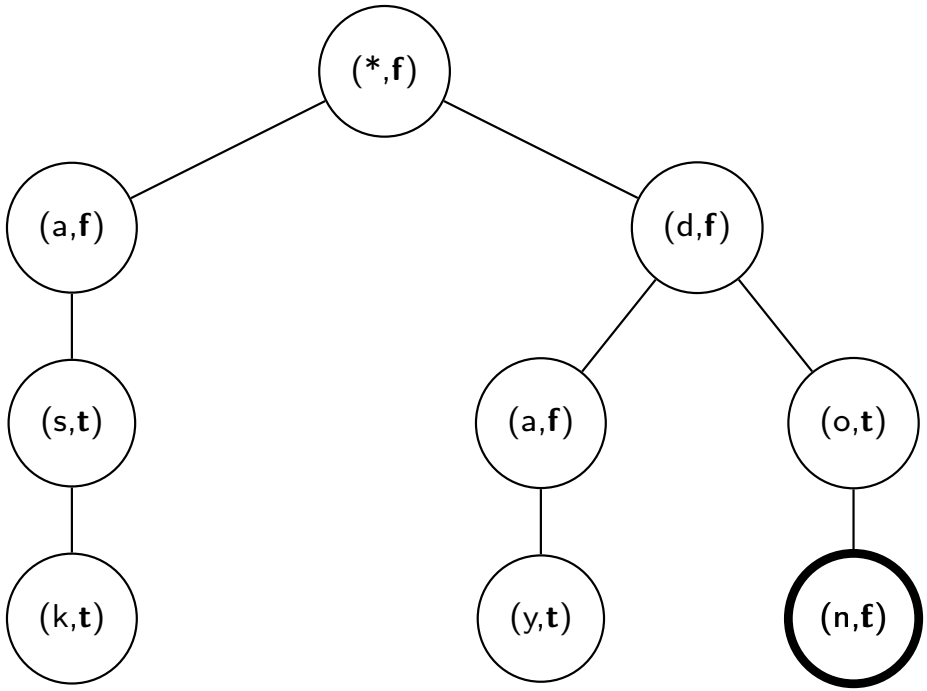
Insertar palabras en un árbol



Inserción de “done”:

- Lo tenemos. Continuamos con la tercera letra “n”.
- Buscamos un hijo con carácter “n”.

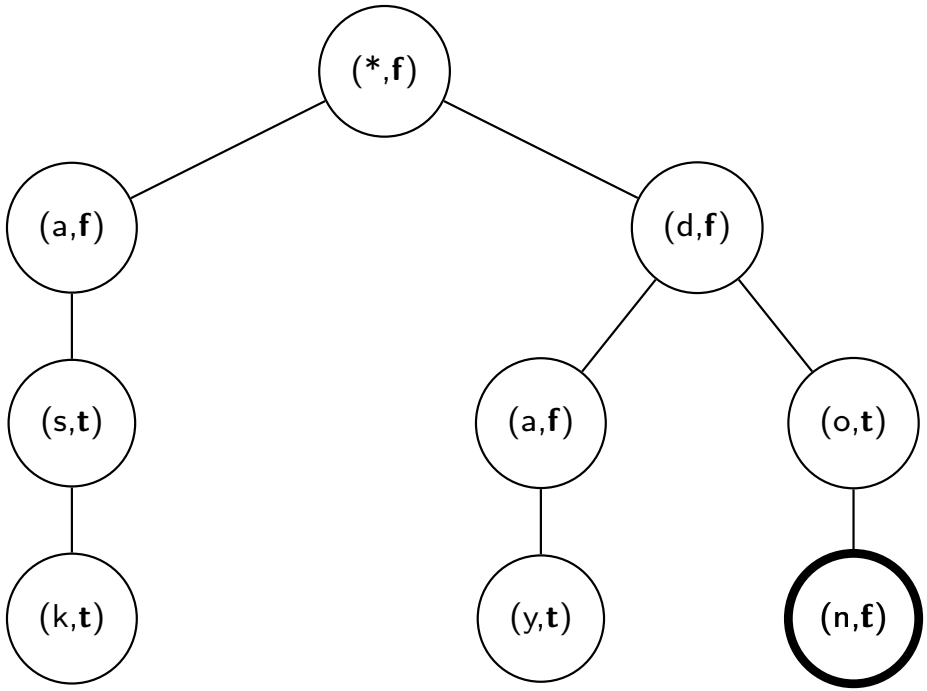
Insertar palabras en un árbol



Inserción de "done":

- No hay. Creamos un hijo (*respetando el orden alfabético*) y con boolean **f**, ya que la palabra no ha terminado.

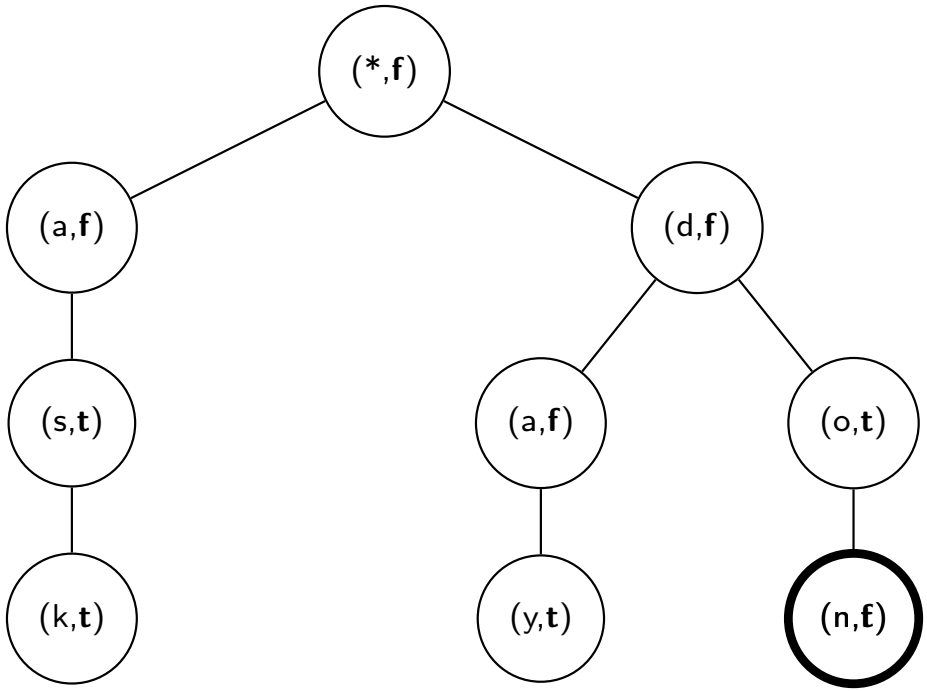
Insertar palabras en un árbol



Inserción de “done”:

- Continuamos con la cuarta letra “e”.

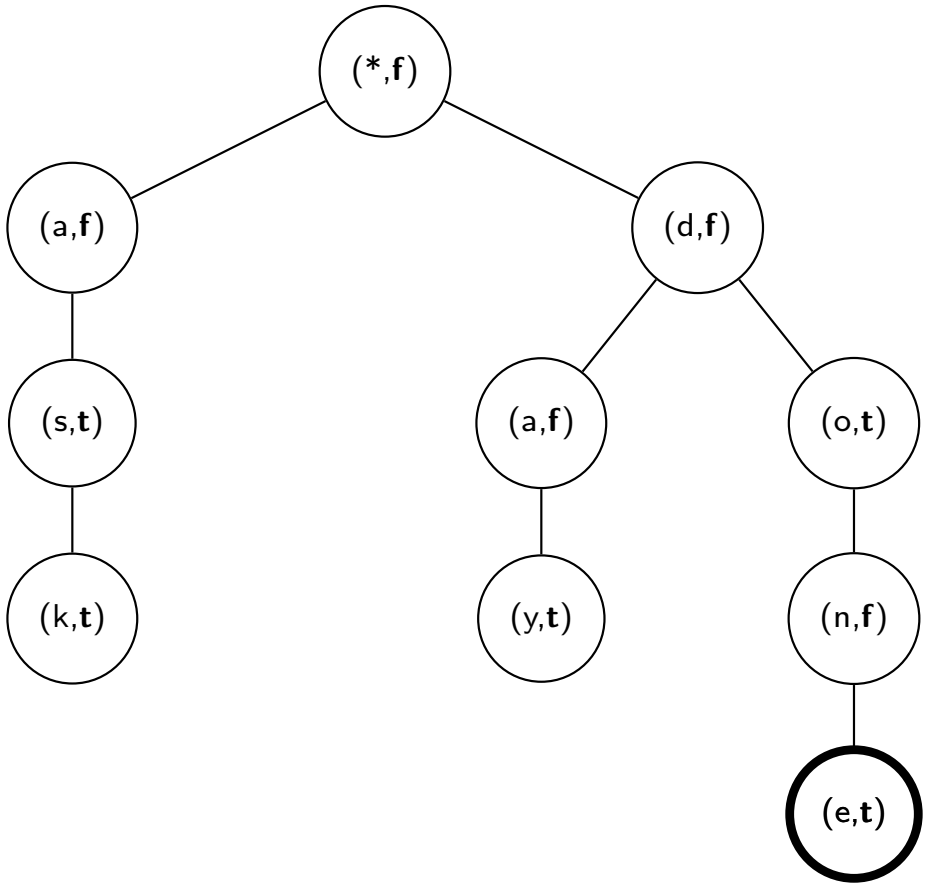
Insertar palabras en un árbol



Inserción de “done”:

- Continuamos con la cuarta letra “e”.
- Buscamos un hijo con carácter “e”.

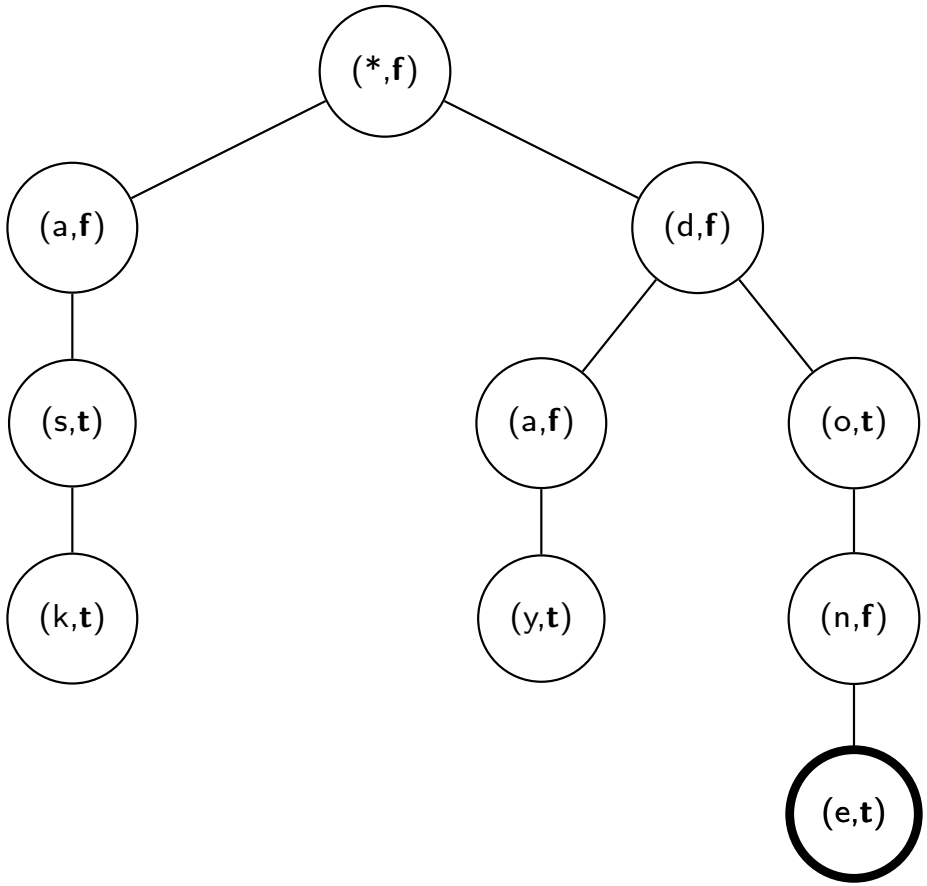
Insertar palabras en un árbol



Inserción de "done":

- No hay. Creamos un hijo nuevo (*respetando el orden alfabetico*) y con boolean **t** ya que la palabra termina.

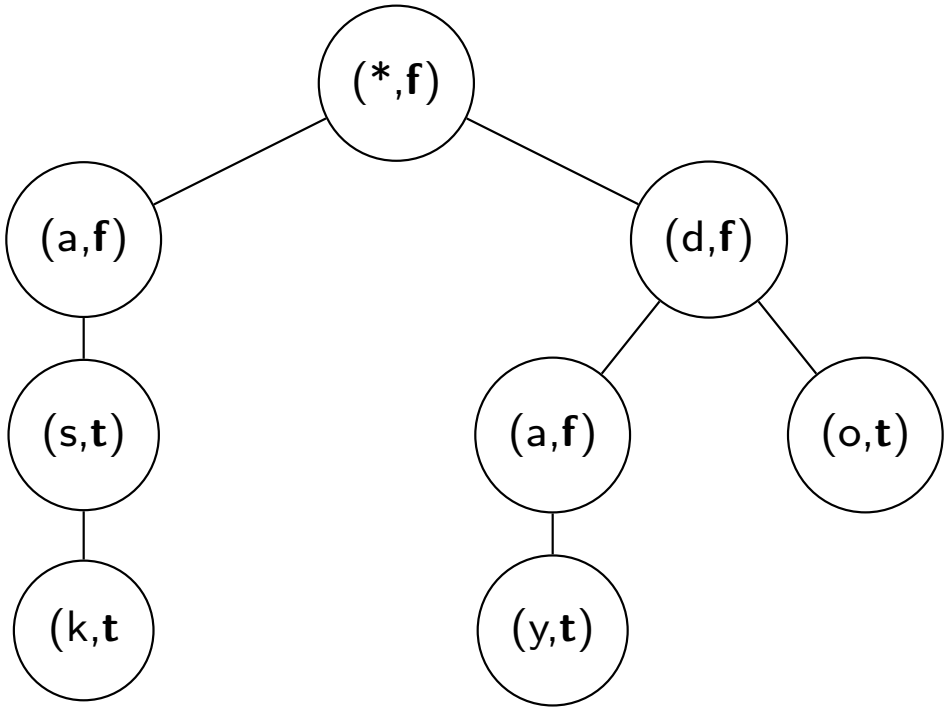
Insertar palabras en un árbol



Inserción de “done”:

- No queda mas letras; terminamos.

Buscar y borrar



- Buscar una palabra en un trie $\Rightarrow$ buscar un camino conteniendo los caracteres de la palabra.
  - Si falta algún carácter, palabra no incluida.
  - Si llegamos a final de la palabra **y** nodo tiene boolean **f**, palabra no incluida.
- Borrar: buscar la palabra en el árbol (punto anterior). Si encontrada, cambiar el boolean del último nodo del camino a **f**.

Todas las palabras con el mismo prefijo

Para implementar el método

```
PositionList<String> wordsBeginningWithPrefix(String prefix);
```

se puede:

- Crea una lista de posiciones *result*.
- Busca el nodo P que representa el prefijo.
  - Si no existe, terminar.
  - Si existe: visitar todos los nodos descendientes de P en *pre-orden*, recordando el camino hasta cada nodo.
    - ★ Si un nodo descendiente n tiene el boolean **t**, se añade a *result* el `String` definido por el camino hasta n.
- Devolver *result*.

Acceder a los caracteres de un `String`

- Dado un `String` `s`, el método `s.length()` devuelve su longitud.
- Se puede acceder al carácter que ocupa la posición `i` dentro de `s` usando `s.charAt(i)`.
- Como en los `array`, el índice inicial de un `String` es 0.

Métodos auxiliares útiles **no obligatorios**

Puede resultar útil definir algunos métodos auxiliares:

```
// Devuelve el nodo cuyo camino desde la raiz contiene
// la palabra prefix. Si no existe el metodo devuelve null.
Position<Pair<Character,Boolean>> findPos(String prefix) { ... }

// Devuelve el hijo del nodo pos que contiene el caracter ch.
Position<Pair<Character,Boolean>>
  searchChildLabelledBy(char ch,
                        Position<Pair<Character,Boolean>> pos) {
  ...
}

// Anade un hijo al nodo pos conteniendo el elemento pair,
// respetando el orden alfabetico de los hijos.
Position<Pair<Character,Boolean>>
  addChildAlphabetically(Pair<Character,Boolean> pair,
                         Position<Pair<Character,Boolean>> pos) {
  ...
}
```

Notas

- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar, y debe ejecutar `TesterLab5` correctamente sin mensajes de error
- Nota: una ejecución sin mensajes de error no significa que el método sea correcto (es decir, que funcione bien para cada posible entrada)
- Todos los ejercicios se comprueban manualmente