1. Tries

Sea S un conjunto de s strings del alfabeto α tal que ninguna es prefijo de otra (en el conjunto).

> 💬 **Definición**
>
> Un trie standard para **S** es un árbol ordenado **T** que cumple
> - Cada nodo de **T**, excepto la raíz , tiene por etiqueta un caracter de α.
> - Los hijos de cada nodo están ordenados según el alfabeto α.
> - Cada string del conjunto **S** corresponde a un camino único desde la raíz hasta un nodo hoja.
>   - La concatenación de los caracteres a lo largo de este camino reconstruye la string original.

Por lo tanto, un trie **T** representa las strings de **S** con caminos desde la raíz hasta los nodos externos de **T**

- Enfoque adecuado cuando se realiza una serie de consultas sobre un texto fijo.
- **Eficiencia en búsquedas:** Permite buscar strings en tiempo proporcional a la longitud del patrón buscado, no al tamaño del conjunto.
- **Compartición de prefijos:** Strings con prefijos comunes comparten los mismos nodos iniciales, ahorrando espacio.

Propiedades de un Trie standard

- La altura del trie es igual al largo de la cadena más larga de **S**
- Cada nodo iterno tiene, a lo mucho, la misma cantidad de nodos que letras del alfabeto.
- Tiene la misma cantidad de hojas que de Strings.
- El número de nodos del trie es, a lo mucho, el largo de todas las cadenas en **S** juntas.

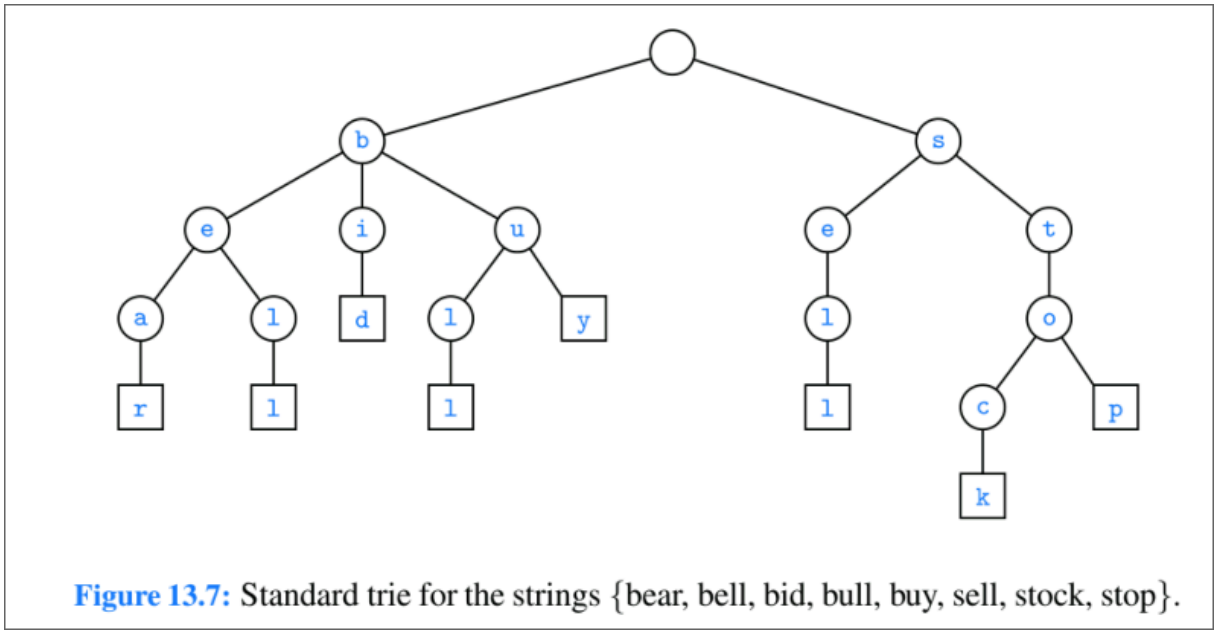


Figure 13.7: Standard trie for the strings {bear, bell, bid, bull, buy, sell, stock, stop}.

1.1. Operaciones primarias

1.1.1. Comparación de patrones (búsqueda exacta)

- **Objetivo:** Determinar si una string **X** está presente en el conjunto **S** almacenado en el trie.
- **Complejidad: O(m)**, donde **m** es la longitud de **X** (independiente del tamaño de **S**).

→ Proceso:

1. Se recorre el trie desde la raíz, siguiendo los caracteres de X.
2. Si se llega a un nodo externo (hoja), X está en S.
3. Si en algún paso falta un hijo, X no existe en el trie.

1.1.2. Comparación de prefijos (búsqueda por prefijo)

- **Objetivo:** Encontrar todas las strings en S que tienen a X como prefijo.
- **Complejidad:** O(m + k), donde:
  - m = longitud de X (tiempo de búsqueda del prefijo).
  - k = número de strings con prefijo X (tiempo de recuperación).

→ Proceso:

1. Se busca el prefijo X en el trie.
2. Si existe, se recorren todos los nodos hijos desde el último nodo de X para listar todas las strings con ese prefijo.

1.2. Inserción en Tries

Insertar una string X en un trie T de manera que:

- Se mantenga la propiedad de que ninguna string en el trie sea prefijo de otra.
- La estructura del trie permita búsquedas eficientes posteriores.

Para garantizar esta condición:

- Se agrega un carácter especial (ej: *, $, #) al final de cada string.
  - Ejemplo: Si X = «casa», se almacena como «casa*». Esto evita que una string sea prefijo de otra (ej: «cas» no será prefijo de «casa» porque esta termina con *).

1.2.1. Algoritmo propiamente dicho

1. **Inicio:** Comenzamos en la raíz del trie.
2. **Recorrido:**
   - Por cada carácter c de X, avanzamos al hijo correspondiente a c.
   - Si un hijo no existe, lo creamos.
3. **Punto de divergencia:**
   - Si el recorrido se detiene en un nodo interno v (porque no hay más caracteres coincidentes), creamos una nueva rama desde v con los caracteres restantes de X.
4. **Nodo final:**
   - Al llegar al final de X, agregamos el carácter especial (*).

Tiempo por Inserción: $O(d.m)$

- Donde **m** es la longitud de la string X y **d** el tamaño del alfabeto.
  - En el peor caso, se deben crear **m** nodos, y cada uno puede tener hasta **d** hijos (aunque en la práctica, solo se crean los necesarios).

Construcción del trie completo (n strings): $O(d \cdot n)$

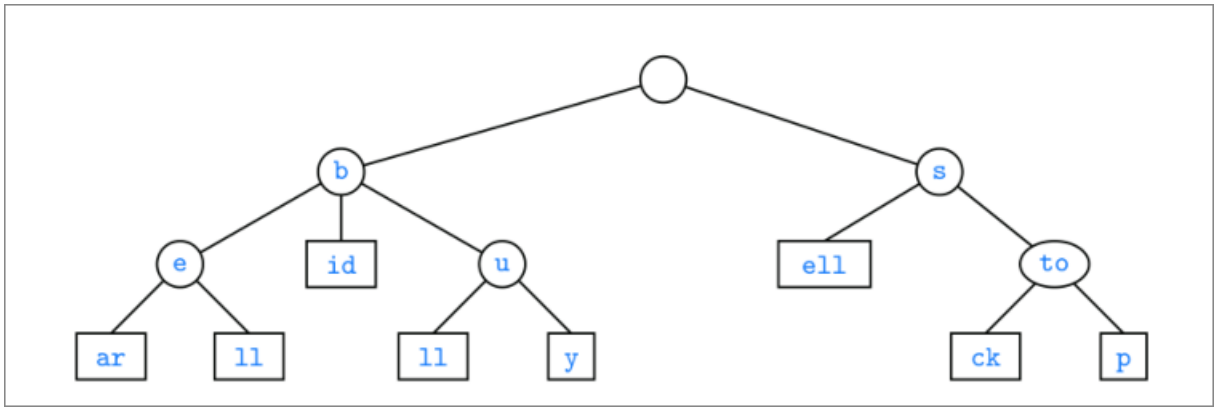
- **n** = suma de las longitudes de todas las strings en S.

1.3. Tries comprimidos (Patricia)

El temita con los Trie standard es que, potencialmente, hay muchos nodos que tienen sólo un hijo. La existencia de estos nodos es desperdicio de memoria.

- Un trie comprimido es similar a un a uno standard, pero se asegura de que cada nodo interno tenga al menos dos hijos.
- Sustituye los nodos de 1 sólo hijo por la cadena correspondiente
- Comprime las cadenas de nodos de un sólo hijo en aristas individuales.
- Se dice que un nodo interno es **redundante** si tiene un sólo hijo y no es el nodo raíz.

> 💬 **Propiedades de un trie comprimido**
>
> Un trie comprimido almacenando una colección **S** de **s** strings de un alfabeto de tamaño **d**, tiene las siguientes propiedades:
> - Cada nodo interno de **T** tiene al menos dos hijos y, a lo mucho, **d** hijos.
> - **T** tiene **s** nodos hoja. Es decir que tiene tantas hojas como cantidad de strings.
> - La cantidad de nodos de **T** es **O(s)**.



Ventajas de un trie comprimido

Los tries comprimidos son especialmente útiles cuando:

- La colección de strings ya está almacenada en otra estructura (ej: un array, una base de datos o un archivo).
- No es necesario almacenar todos los caracteres explícitamente en el trie, sino que este actúa como un índice para acceder a los datos originales.
- El espacio en memoria es crítico, y se quiere evitar redundancia.

Ejemplo

- Tenemos un array **S** = [«algoritmo», «algebra», «arbol»] almacenado en memoria.
  - El trie comprimido no guarda las strings completas, sino referencias a substrings de **S**.

En lugar de almacenar caracteres directamente, los nodos del trie comprimido usan tries (i, j, k) para representar substrings:

- i: Índice del string en el array S (ej: S[0] = «algoritmo»).
- j: Posición inicial de la substring en S[i].

- k: Posición final de la substring en S[i].
- La etiqueta del nodo es la substring X = S[i][j..k].

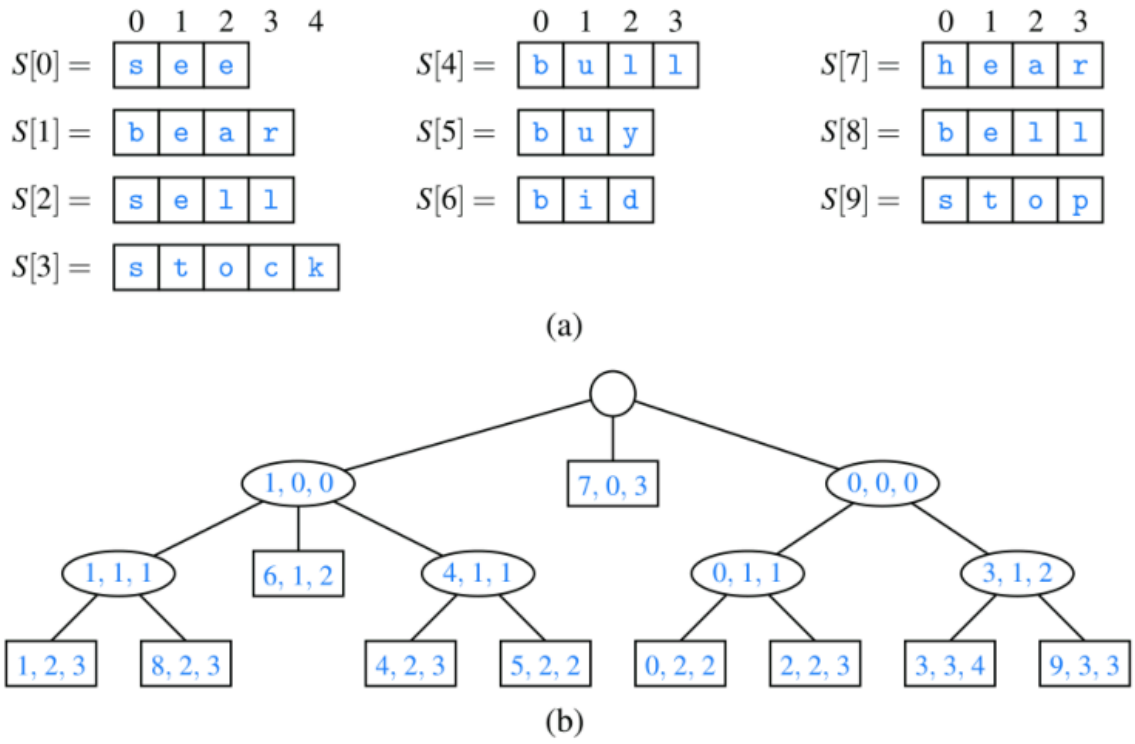


Figure 13.10: (a) Collection S of strings stored in an array. (b) Compact representation of the compressed trie for S.

Este mecanismo nos permite reducir el espacio total para el trie mismo desde $O(n)$ en un trie standard a $O(s)$ en un trie comprimido. Donde n es el largo total de los strings en **S** y s es el número total de los strings en **S**.

La búsqueda en un trie comprimido no es necesariamente más rápida que en un trie standard.