



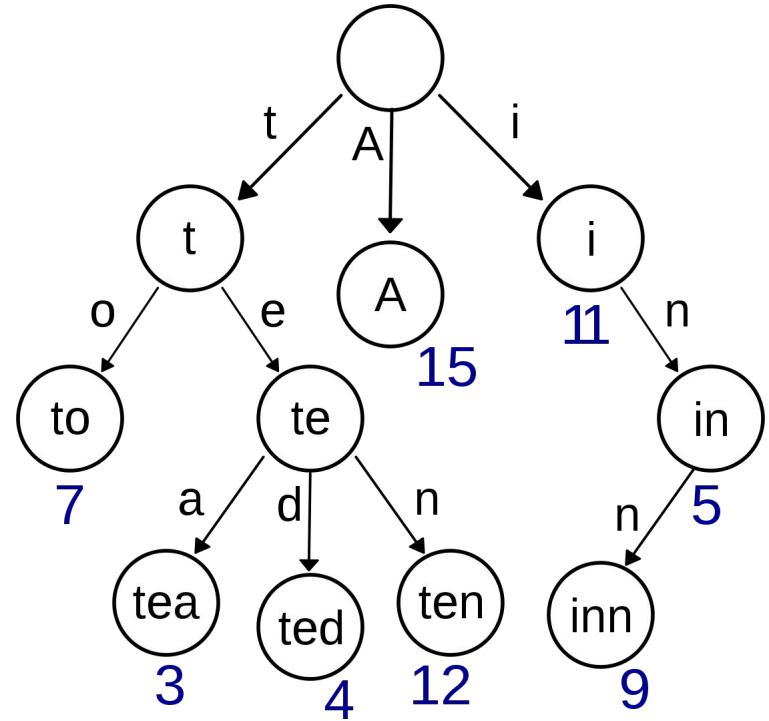
Dictionaries Allowing Errors in Queries

Maria Giraldo, Arturo Navarro, Anabel Paredes

Los Tries solo encuentran ocurrencias exactas.

Sería bueno tener una estructura de diccionario que encuentre cadenas que se diferencien de la búsqueda en d caracteres. (distancia d de Hamming)

Para $d=1$ existe solución





Dos posibles soluciones....

Generar todas las cadenas que difieren en d caracteres de un conjunto de palabras w_i , lo cual genera para cada palabra $\Theta(|A|^d \text{largo}(w_i)^d)$ variantes, donde A es el alfabeto.

Usar el trie para las w_i , pero por cada consulta q , generar Q con todas las palabras que difieren de q en d caracteres, y luego buscar cada Q_i en el trie. Esto toma $\Theta(|A|^d \text{largo}(q)^d)$ consultas, cada una de $\Theta(\text{largo}(q))$

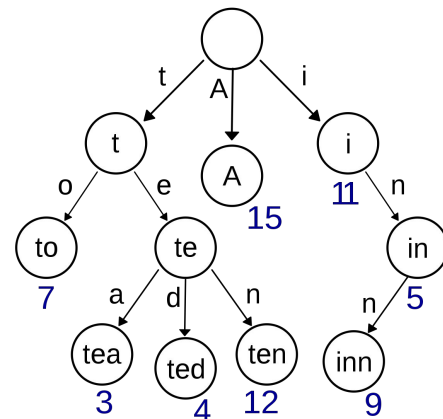
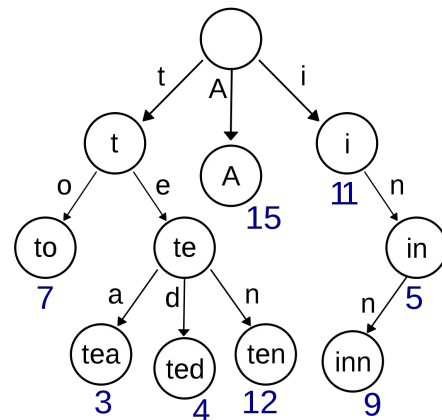
Ambos enfoques son **ineficientes**

Propuesta

Usar dos tries de tamaño $O(|A|\Sigma_w)$ donde $\Sigma_w = \sum_{i=1}^n \text{largo}(w_i)$

El primer trie T_w , guarda todas las w en el orden normal, mientras que el otro T'_w , almacena las palabras en el orden inverso

Para una consulta q , existe en T_w u que es un nodo que representa un prefijo π de q y de alguna w_i ; y existe en T'_w v para un sufijo σ de q y alguna palabra w_j



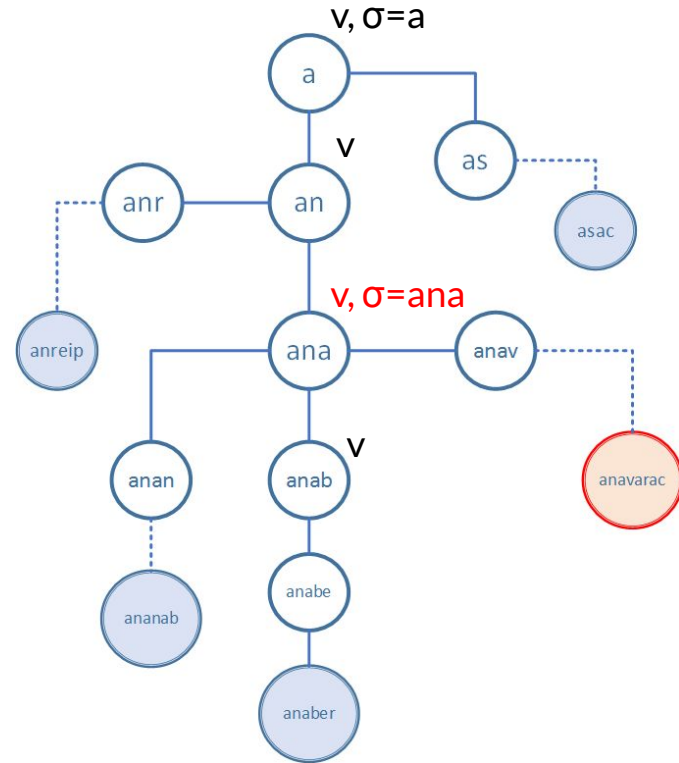
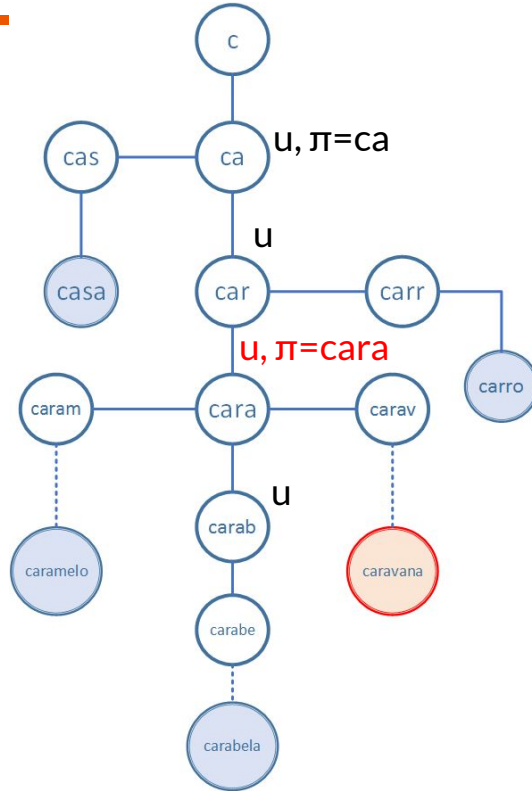


Par factible

El par de nodos (u,v) es un par factible si existe una cadena $w = \pi b \sigma$ tal que:

- π es prefijo de q con longitud l_q
- σ es sufijo de q con longitud r_q
- $b \neq q[l_q + 1]$
- $l_q + r_q + 1 = \text{largo}(q)$

Ejemplo: q=carabana (caravana)






Teorema

Estructura *double-trie* formada; listas de *trie* nodos y un árbol de búsqueda balanceado, almacena:

- conjunto de palabras de tamaño Σ_w , alfabeto A .
- Operación **find** en un string q , para encontrar todas las palabras que difieren en más de un lugar desde q en tiempo de salida $O(|A| \text{length}(q) \log \Sigma_w + k)$, k palabras.
- Operaciones **insert** y **delete** de una palabra w en tiempo $O(|A| \text{length}(w) \log \Sigma_w)$.

Tiempo para construir la estructura: $O(|A| \Sigma_w \log \Sigma_w)$.



Si el modelo computacional permite el uso de una tabla hash en lugar del árbol de búsqueda, , el factor $\log \Sigma_w$ desaparece, así que todas las operaciones se convierten en tiempo lineal en la longitud de la entrada, **ciertamente óptimo**.

Una solución diferente es ordenar todas las cadenas de entrada en orden lexicográfico en $O(\Sigma_w)$ y asignándoles su rango en ese orden como su número. En esta situación se puede aplicar la **cascada fraccional**.



Representación de cascada fraccional

- Al seguir una ruta los **subconjuntos** se vuelven **más dispersos**, por lo tanto:
 - se representan en listas y sublistas ordenadas, con punteros a sus vecinos
- Al comparar el primero con el segundo TRIE se obtiene una **secuencia de intervalos creciente** y una secuencia de **sublistas ordenadas decrecientes**.



Prueba de Etapa

Intersección entre el intervalo y la sublista

- Primero, se determina la **posición del intervalo** en la primera lista (se necesita árbol de Búsqueda)
- Luego, se ve la posición del intervalo anterior en la nueva sublista, posteriormente se extiende el intervalo y se verifica si contiene uno de los vecinos en la sublista.
 - Seguir el primer TRIE y colocar nodos en la pila para alcanzar la posición correspondiente en el segundo TRIE nos da una complejidad de **$O(\text{largo}(q))$** .
 - Determinar la posición inicial del intervalo en la lista una complejidad de **$O(\log n)$** .
 - Paso a la siguiente sublista e intervalo **$O(1)$**

Total: **$O(\text{largo}(q) + \log n)$**



Referencias

[1] Advanced data Structures pg 336-373

[2] Brodal and Gasieniec, Aproximate dictionary queries(1996)