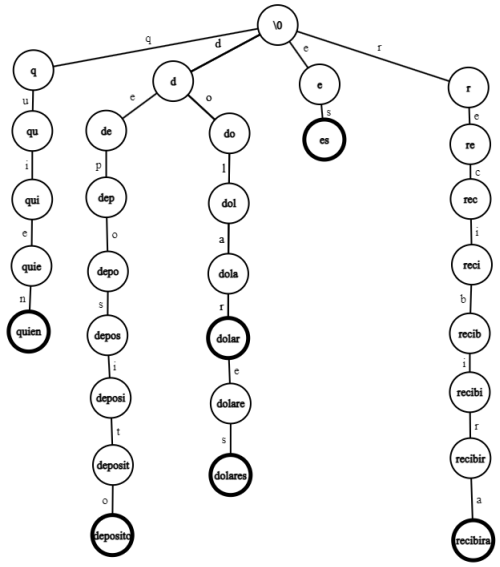
Explicacion del algoritmo

Idea inicial

El objetivo del programa es el de poder identificar las palabras a espaciar en una única lectura del texto de entrada. Para ello es necesario poder rastrear multiples palabras del diccionario al mismo tiempo.

Para lograr esto se guardan todas las palabras en un árbol general, cada nodo del árbol va a representar un posible prefijo de una palabra del diccionario. Las aristas del árbol se etiquetan con un carácter del alfabeto, de modo que si un nodo A con la palabra p esta conectado a otro nodo B mediante una arista con el carácter 'c', entonces la palabra que contiene el nodo B es p+c.

Por ejemplo el árbol para el diccionario ["quien", "deposito", "dolar", "dolares", "es", "recibira"] es:

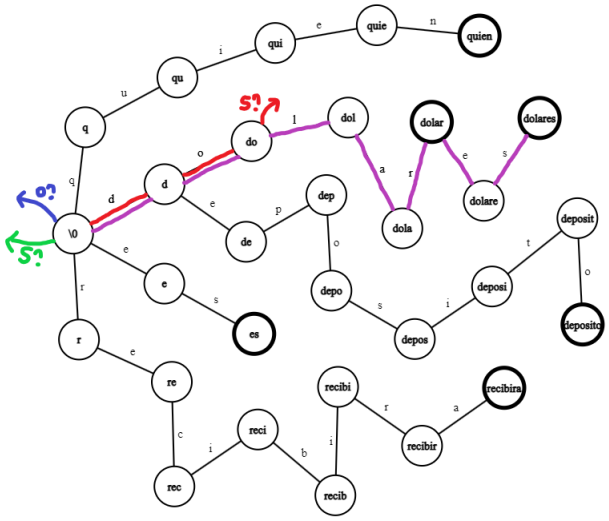


Los nodos a su vez tienen que indicar si el prefijo representa una palabra que esta en el diccionario, en el gráfico eso queda representado por un circulo de un color mas fuerte.

Usando este árbol, se puede analizar el string como sigue. Se inicializan dos iteradores i,j al inicio de la palabra a analizar, en cada paso se mueve el iterador j y se recorre el árbol en la arista correspondiente. Si en algún momento se llega a un nodo con una palabra del diccionario se guarda la posicion actual en una variable k(esta variable se sobrescribe si se encuentra otra palabra mas grande). Cuando se llega a un punto en el que no se puede seguir por el árbol, la palabra encontrada es el substring [i:k]. Despues se avanza el iterador i hasta k+1(hasta i+1 en el caso en que no se encontro una palabra) y se vuelve a la raíz del arbol para continuar el analisis.

Asi para el diccionario de ejemplo, la palabra "dosdolares" se espacia de esta

forma:



i,j
↓
1: dosdolares → dosdolares → dosdolares → **X**

El nodo "do" no tiene una arista con el carácter 's', como no se puede seguir vuelvo a la raíz y avanzo i un lugar para adelante.

i,j
↓
2: dosdolares → **X**

La raíz no tiene una arista con el carácter 'o', como no se puede seguir, paso al carácter siguiente.

i,j
↓
3: dosdolares → **X**

La raíz no tiene una arista con el carácter 's', como no se puede seguir, paso al carácter siguiente.

i,j → i j → i j → i j → i j
↓ ↓↓ ↓↓ ↓↓ ↓↓
4: dosdolares → dosdolares → dosdolares → dosdolares → dosdolares

Se encontro la palabra "dolar", se marca con una flecha k

i k j → i k j
↓ ↓↓ ↓ ↓↓
dosdolares → dosdolares → **X**

Se encontro la palabra "dolares", se marca con una flecha k:

i k
↓ ↓
dosdolares → **X**

No se puede seguir porque se termino de leer el string, la palabra mas large encontrada fue "dolares":

La salida final del programa es: "dolares"

Esta idea no es lo suficientemente optima ya que el indice j tiene que retroceder junto al indice i para analizar caracteres que ya fueron revisados. Esto hace que en algunos casos el algoritmo realiza n^2 operaciones, donde n es el largo del string. Es el caso de la palabra "cinco" con el diccionario ["cincos","incos","ncos","cos","os","s"]

Para poder evitar esto, es necesario agregar información adicional en el árbol.

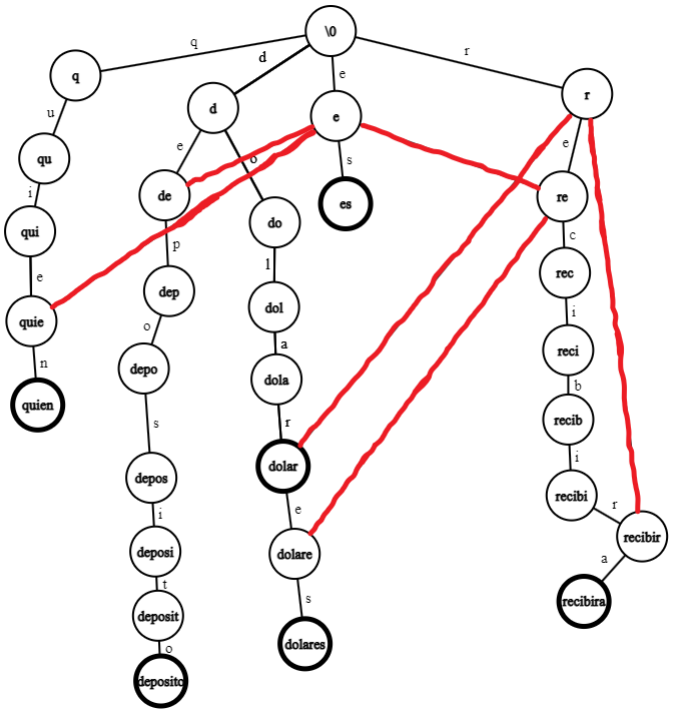
Optimización: transiciones de falla

La idea de la optimización es que si no se puede seguir recorriendo el arbol desde un nodo B, en vez de volver a la raíz y recorrer desde ahi hasta un cierto nodo A, se salte directamente del nodo B al nodo A. De esta manera se "reciclan" los caracteres que habia entre i y j en vez leerlos nuevamente.

Dado que es posible llegar desde el nodo raiz hasta el nodo A usando uno o mas de los caracteres finales de B. El prefijo que representa A es a su vez un sufijo propio de B. Solo hace falta añadir una restriccion adicional, el sufijo propio no puede contener parte de una palabra aceptada(sino podria pasar que se acepten dos palabras que se solapan), por eso tiene que ser un sufijo propio que no contenga parte de una palabra aceptada.

Así, se puede añadir a cada nodo del arbol que representa el prefijo X una arista que apunte a otro nodo del árbol que represente su sufijo propio más largo(transición de falla).

Estas serian algunas de las transiciones de falla para el arbol de ejemplo. Las aristas en rojo marcan las transiciones de falla. Dado que pueden ser muchas, los nodos que no tienen aristas rojas tienen su transición de falla apuntando a la raíz:



De esta forma, si en un cierto punto se quiere leer el carácter 'x' y el nodo

actual no posee una arista con dicho carácter, se sigue la transición de falla y se revisa si es posible seguir desde ahi con el carácter 'x', en caso contrario se sigue la transición de falla del nodo al que recién se llego y se repite el proceso.

Eventualmente las transiciones de falla terminan en la raíz, si se llega hasta la raíz y no es posible seguir con el caracter 'x', entonces se descarta el caracter y se sigue leyendo el string.

Con las transiciones de falla incluidas, el árbol ya puede usarse como si fuera un Automata de Estado Finito(de hecho, asi es como se lo llama en el código).

Calculo de las transiciones de falla

Las transiciones de falla solo se pueden calcular una vez se ingresaron todos las palabras del diccionario. Para calcularlas se realiza lo siguiente:

Se recorre el árbol por niveles: Si el nodo actual N es la raíz, su padre es la raíz o es un nodo de palabra aceptada, entonces su transición de falla apunta a la raíz. En caso contrario, N tiene un padre con la transición de falla ya calculada.

Si el nodo N esta unido a su padre P por una transición con el caracter 'x', entonces se sigue la transición de falla de P y de ahi se sigue la transición por el caracter 'x' si es que existe. Si no existe entonces se sigue la transición de falla del padre de P y se repite el proceso.

La transición de falla de N va a terminar apuntando al nodo al que se llego siguiendo las sucesivas transiciones de falla de los ancestros y la transición por 'x'. Si se llego hasta la raíz y la raíz no tiene una transición 'x', entonces la transición de falla de N apunta a la raíz.