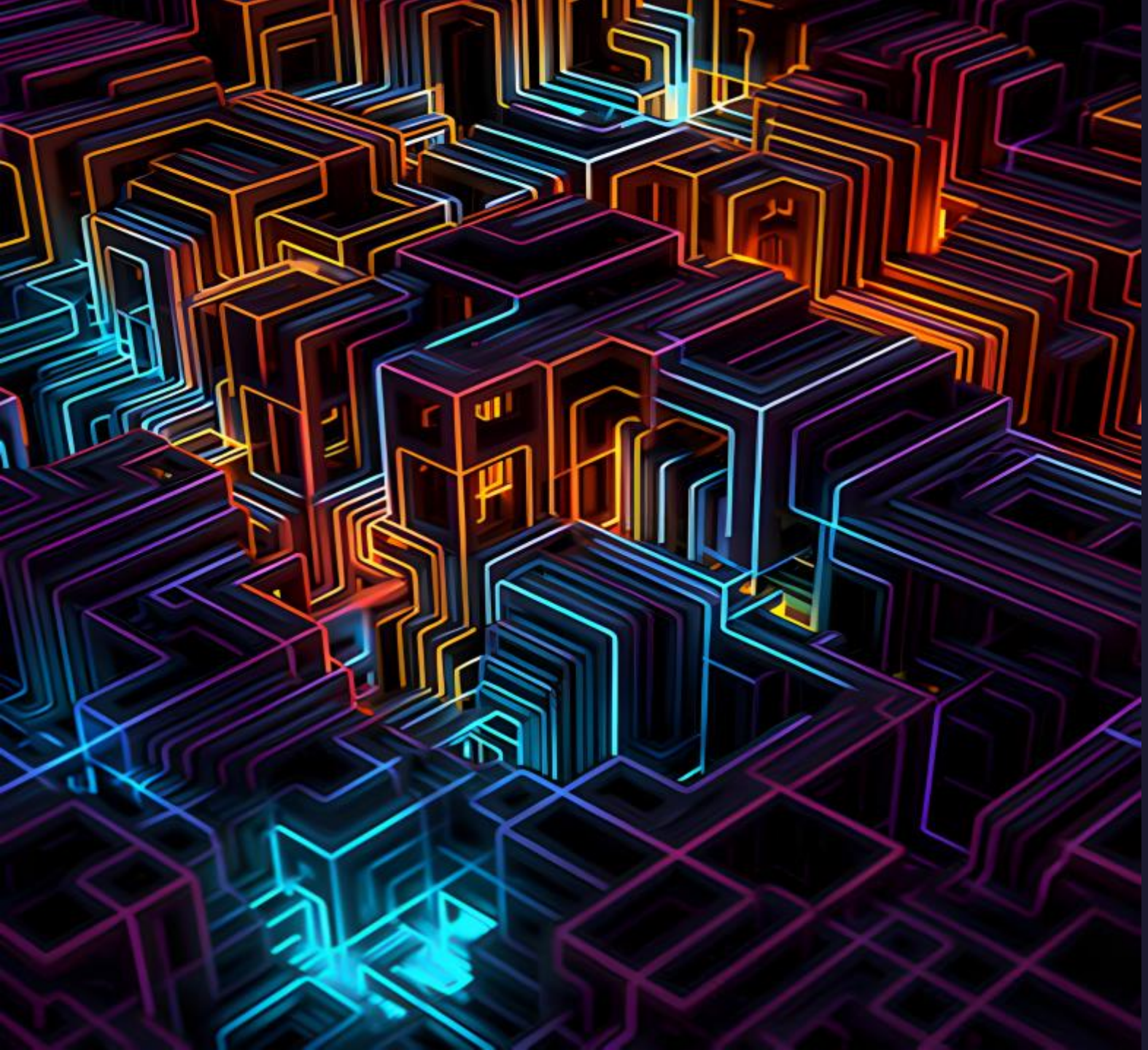


Arboles Trie

Análisis y diseño de algoritmos avanzados

Dra. Valentina Narváez Terán

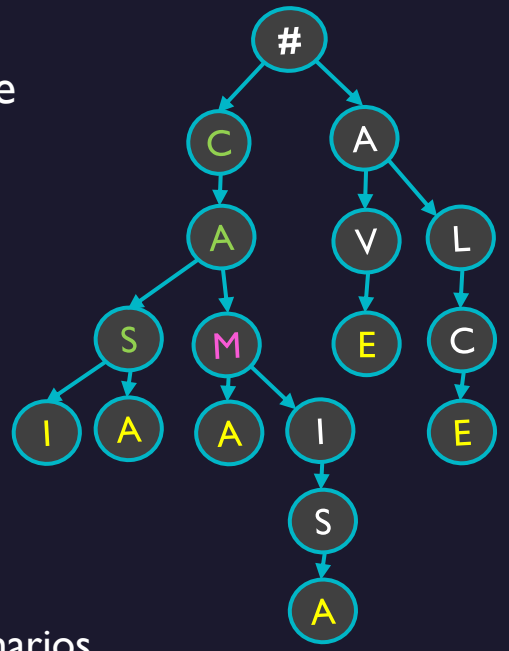
Tecnológico de Monterrey

Arboles trie

Son un tipo de árbol, para procesar cadenas eficientemente

Pueden crearse a partir de grupos de palabras

casi
casa
cama
camisa
ave
alce



Sirve para crear diccionarios, autocorrectores, o autocompletes

El # de la raíz no es importante, solo simboliza que es la raíz

Para buscar palabras validas. es útil que los nodos tengan un atributo que indique si son el final de una cadena o no.

Por ejemplo sabemos que "CASA" es valida por que el ultimo nodo de la ultima "A" es un nodo final

En cambio, si buscamos "CAM" sabremos que no es una palabra valida porque M no es un nodo final

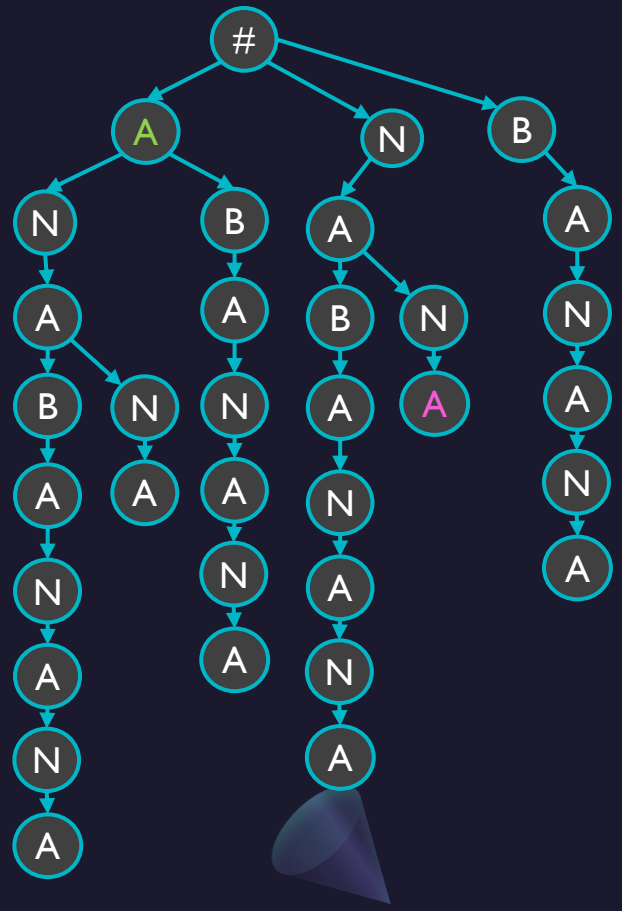
Arboles trie

Si el trie es creado a partir de los sufijos de una sola cadena, entonces es un suffix trie

Y será útil para… ¡string matching!

Sufijos de *anabanana*

anabanana
nabanana
abanana
banana
anana
nana
ana
na
a



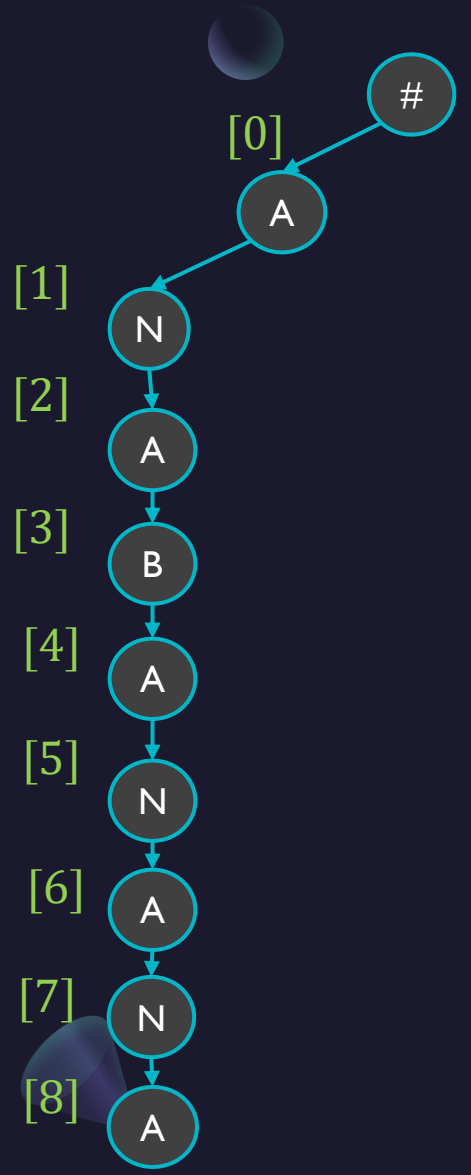
En este caso, los nodos necesitan un atributo donde almacenar los índices de las ocurrencias del carácter, con respecto a la cadena original "anabanana"

Arboles trie

Al ir creando los nodos, agregamos el índice del carácter respecto a la cadena original a lista de ocurrencias del nodo

a n a b a n a n a
0 1 2 3 4 5 6 7 8

0	anabanana
1	nabanana
2	abanana
3	banana
4	anana
5	nana
6	ana
7	na
8	a



Arboles trie

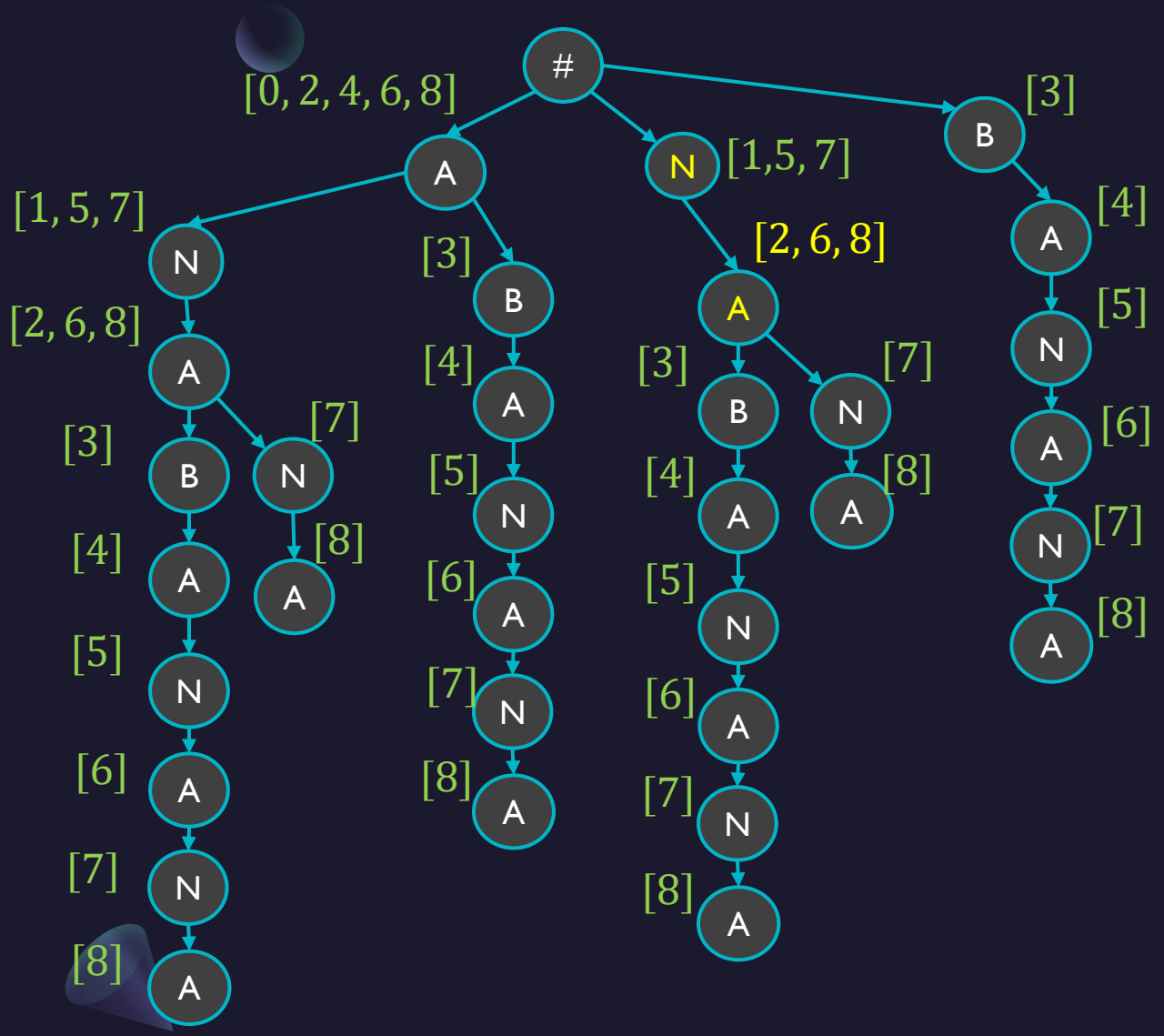
Para buscar un patrón, viajamos por el árbol, desde la raíz, siguiendo los caracteres.

Las ocurrencias asociadas al nodo del ultimo carácter sirven para calcular donde se encuentra

Por ejemplo, "na" tiene ocurrencias [2, 6 8] en su ultimo nodo

Así que "na" aparece en:
$2 - (x - 1) = 1$
$6 - (x - 1) = 5$
$8 - (x - 1) = 7$

$x = 2$ es la longitud de "na"



Arboles trie: crear el árbol

Versión recursiva

$insert\_recursivo(nodo, C, i, index)$

Si $i < m$
  Por cada $hijo$ de nodo
    Si $hijo$ coincide con $C[i]$
      $insert\_recursivo(hijo, C, i + 1, index)$
      return

  // Si ningún hijo coincide con $C[i]$
  $hijo$ = un nuevo nodo con $C[i]$
  Agregar hijo a $nodo$
  $insert\_recursivo(hijo, C, i + 1, index)$

La raíz de el árbol no es parte de ninguna cadena

Al agregar "anabanana" el nodo de la ultima "a" queda marcado como fin de cadena

Esto se puede conseguir con un atributo del nodo

Importante: al pasar por nodos que coincidan, o agregar hijos hay que mantener los atributos actualizados según el tipo de trie

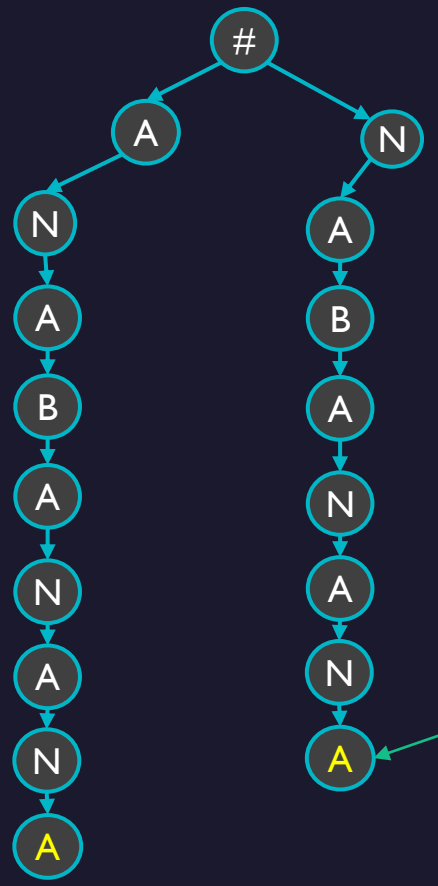
Arboles trie

Versión recursiva

$insert\_recursivo(nodo, C, i, index)$

Si $i < m$
  Por cada $hijo$ de nodo
    Si $hijo$ coincide con $C[i]$
      $insert\_recursivo(hijo, C, i + 1, index)$
      return

  // Si ningún hijo coincide con $C[i]$
  $hijo$ = un nuevo nodo con $C[i]$
  Agregar hijo a $nodo$
  $insert\_recursivo(hijo, C, i + 1, index)$



No sabemos cuantos hijos tendrá cada nodo

Se pueden definir como una lista/arreglo

Al agregar "nabanana" el nodo de la ultima "a" queda marcado como fin de cadena

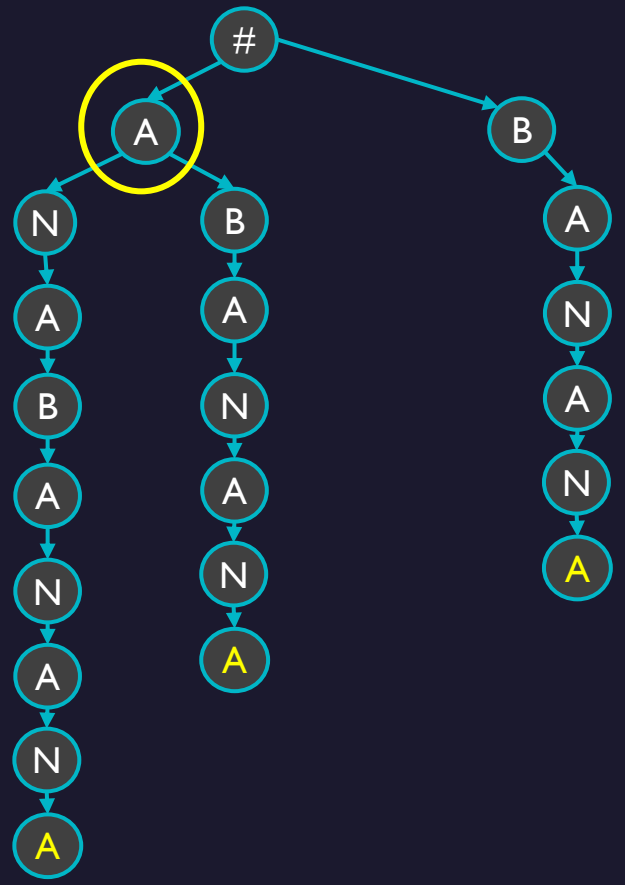
Arboles trie

Versión recursiva

$insert\_recursivo(nodo, C, i, index)$

Si $i < m$
  Por cada $hijo$ de nodo
    Si $hijo$ coincide con $C[i]$
      $insert\_recursivo(hijo, C, i + 1, index)$
      return

  // Si ningún hijo coincide con $C[i]$
  $hijo$ = un nuevo nodo con $C[i]$
  Agregar hijo a $nodo$
  $insert\_recursivo(hijo, C, i + 1, index)$



Al agregar "abanana", no es necesario crear un hijo para la primer "a"

Arboles trie

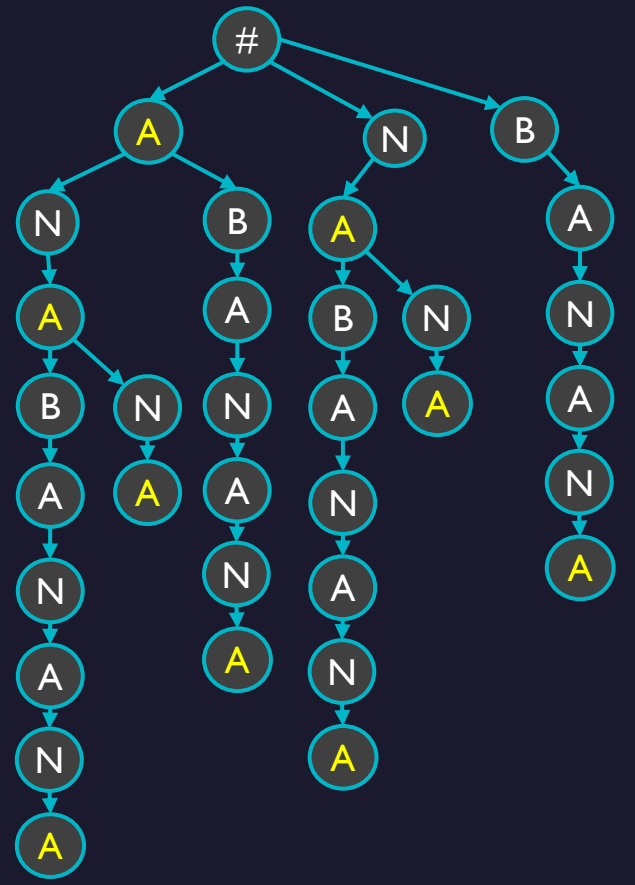
Versión recursiva

$insert\_recursivo(nodo, C, i, index)$

Si $i < m$
  Por cada $hijo$ de nodo
    Si $hijo$ coincide con $C[i]$
      $insert\_recursivo(hijo, C, i + 1, index)$
      return

  // Si ningún hijo coincide con $C[i]$
  $hijo$ = un nuevo nodo con $C[i]$
  Agregar hijo a $nodo$
  $insert\_recursivo(hijo, C, i + 1, index)$

Eventualmente, tendremos el árbol completo



Arboles trie: actividad 3.1

Usando el código base (en Drive) crea dos trie y úsalos para hacer búsquedas.

1. Trie diccionario creado a partir de una lista de palabras.
   Crea una función de búsqueda que determine si una palabra existe o no en el árbol

2. Suffix trie a partir de sufijos una cadena.
   Crea una función de búsqueda que pueda encontrar las ocurrencias de un patrón en el árbol

C/C++/Python
En equipo, para el día de la próxima clase

Utiliza los siguientes casos de prueba

casi
casa
cama
camisa
ave
alce

anabanana
nabanana
abanana
banana
anana
nana
ana
na
a