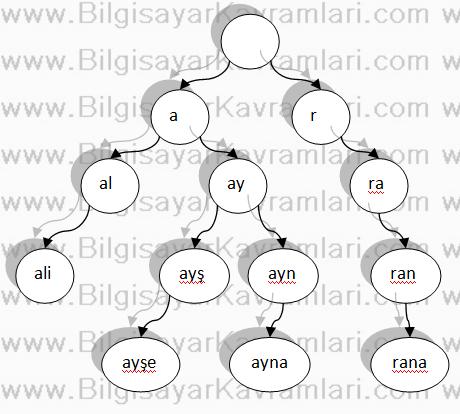
# METİN AĞAÇLARI(trie)

Metin ağaçları(trie), her düğümün kendisinden sonra gelen harfi işaret ettiği multi-way [ağaçlardır](http://bilgisayarkavramlari.com/2008/05/07/agaclar-tree/). Ayrıca bilgisyar bilimlerinde digital,[radix tree](http://en.wikipedia.org/wiki/Radix_tree),prefix tree isimleriyle de anılmaktadır.Basitçe ağacın üzerine bir metin kodlanabilir ve bu metni veren ağacın üzerinde tek bir yol izlenebilir (deterministic). Veri yapılarıdaki klasik ikili arama ağçlarından farklı olarak(binary search tree) metin ağacındaki düğümler kendilerine has bir anahtar değere sahip değillerdir. Ağaçtaki bütün çocuk düğümler, yapraklar geldikleri ebeveyn düğümdeki eki kendilerinde önek olarak bulundururlar. Bütün yapraklar ve çocuk düğümler ortak kökten yani boşluk karakterinden gelmektedirler. Ağaçtaki yaprak haricindeki çocuk düğümlerinin aranan kelimeye anahtar teşkil etmesi ve yol göstermesi haricinde bir anlamı yoktur. Yani kökten çıkarak bir yaprağa varılmadığı sürece anlamlı bir string elde edilemez. Durum aşağıdaki örnek üzerinde daha rahat anlaşılabilir:



http://www.bilgisayarkavramlari.com/wp-content/uploads/trieagaci.jpg

Yukarıdaki ağaçta dikkat edilirse kök düğüm her zaman boş metni (string) ifade etmektedir. Bu boş metin hangi harf ile devam edilirse ilgili kolu takip eder ve gitmiş olduğu düğüm o ana kadar geçmiş olduğu kollardaki harflerin birleştirilmiş halidir. Bir düğümden bir harf taşıyan sadece bir kol çıkabilir.

Yukarıdaki ağaçta dikkat edilirse kök düğüm her zaman boş metni (string) ifade etmektedir. Kökten başlayarak kelimenin içindeki harflerin sırasını teker teker takip ederek ağaçta kökten yapraklara doğru ilerlenir. Eğer sıradaki harf ile bulunulan düğümden bir alt dala gidilemez ise kelime bulunamamış demektir. Sorunsuz bir şekilde kökten başlayarak kelimedeki harflarin sırasıyla herhangi bir yaprağa ulaşabiliyorsak aradığımız kelime ağaçta yer alıyor demektir.

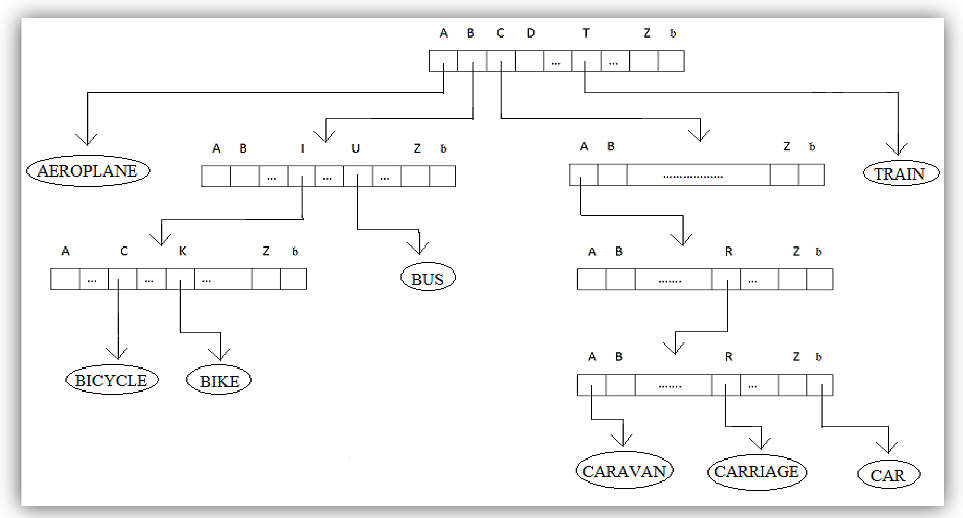
Trie ağacının ismi re**trie**val kelimesininin ortasındaki 4 harften gelmektedir.

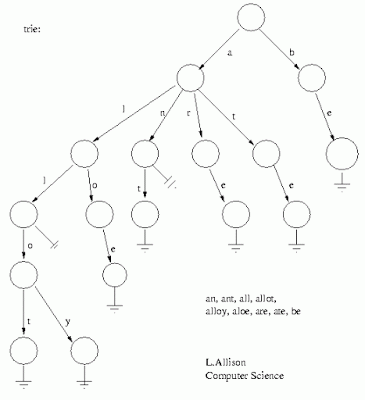
Metin ağaçlarının (trie), [ikili arama ağaçlarına](http://bilgisayarkavramlari.com/2008/05/07/ikili-arama-agaci-binary-search-tree/) göre en önemli avantajları bir metni aramanın, metin boyutu kadar işlem gerektirmesidir. [İkili arama ağaçlarında](http://bilgisayarkavramlari.com/2008/05/07/ikili-arama-agaci-binary-search-tree/) ise bu süre log n kadar varkit almaktadır. Buradaki n, ağaçtaki düğüm sayısıdır dolayısıyla [ikili arama ağaçları](http://bilgisayarkavramlari.com/2008/05/07/ikili-arama-agaci-binary-search-tree/), ağaçtaki bilgiye göre hızlı veya yavaş çalışırken, metin ağaçları, ağaçta ne kadar bilgi bulunduğundan bağımsız olarak çalışırlar.

Metin ağaçları [hafızayı](http://www.bilgisayarkavramlari.com/2008/11/07/rastgele-erisilebilir-bellek-random-access-memory-ram/) da verimli kullanırlar çünkü bir metin ağacının en derin noktası, ağaç üzerindeki en uzun metin kadardır. [İkili ağaçlar](http://bilgisayarkavramlari.com/2008/05/07/ikili-arama-agaci-binary-search-tree/)da ise bu derinlik eklenen düğüm sayısına göre en kötü ihtimalle düğüm sayısı kadar olabilmektedir.

Ayrıca metin ağaçları [en uzun önek eşlemesi (longest prefix matching)](http://bilgisayarkavramlari.com/2007/12/22/en-uzun-onek-eslesmesi-longest-prefix-matching/) gibi problemlerin çözümünde de avantaj sağlar.

İkili arama ağaçlarındaki aynı anahtar değere sahip düğümlerin eklenmesinde sorunlar çıkarken metina ağçlarında böyle bir sorun yaşanmamaktadır.





Metin Ağacına Kelime Ekleme Algoritması

\*\*Procedure Insert(trie, keyWord)

begin

lenght <- length(keyWord)

next <- trie;

//

if trie = NULL then

trie = create empty internal node

new\_leaf = create leaf with keyWord

trie->pChildren[keyWord[0]-'a'] <- new\_leaf

exit

else

index <- next

end if

inWordIndex <- 0//

while inWordIndex < lenght and index->NotLeaf = true and

index->pChildren[keyWord[inWordIndex]-'a'] <> NULL)) do

parent <- next;

next <- index->pChildren[keyWord[inWordIndex]-'a'];

index <- next;

inWordIndex <- inWordIndex + 1

end while

if inWordIndex < lenght and index->pChildren[keyWord[inWordIndex]-'a'] = NULL and

index->NotLeaf = true then

new\_index <- NewLeaf(keyWord)

index->pChildren[keyWord[inWordIndex]-'a'] <- new\_index

exit

else

data <- next

if data->word = keyWord then

print "Word already exists in trie !!!"

else

oldChildren <- parent->pChildren[keyWord[inWordIndex-1]-'a']

newWord <- NewLeaf(keyWord)

prefixLenght <- lenght(keyWord)

if data->word[0] <> '\0' then

if lenght(data->word) < prefixLenght then

prefixLenght = lenght(data->word)

createIntern <- false

while inWordIndex <= prefixLenght and (data->word[0] <> '\0' and

(data->word[inWordIndex-1] = keyWord[inWordIndex-1]) or (data->word[0] == '\0' ) do

intern <-NewIntern()

parent->pChildren[keyWord[inWordIndex-1]-'a'] <- intern

parent->NotLeaf <- true

parent <- intern

inWordIndex <- inWordIndex +1

createIntern = true

end while

if createIntern then

inWordIndex <- inWordIndex -1

if inWordIndex <> prefixLenght or (inWordIndex = prefixLenght and

length(keyWord) = length(data->word)) then

parent->pChildren[data->word[inWordIndex]-'a'] <- oldChildren

parent->pChildren[keyWord[inWordIndex]-'a'] <- newWord

else

if data->word[0] <> '\0' then

if lenght(data->word) <= prefixLenght then

parent->pChildren[26] = oldChildren

parent->pChildren[keyWord[prefixLenght]-'a'] = newWord

else

parent->pChildren[26] = newWord

parent->pChildren[data->word[prefixLenght]-'a'] = oldChildren

end if

else

for (int count = 0 ; count < 27;count++)

parent->pChildren[count] = oldChildren->pChildren[count]

parent->pChildren[26] = newWord

end if

end if

exit

end

Metin Ağacında Kelime Arama Algoritması

\*\*Procedure FIND(trie, string)

begin

if trie = NULL then

return FALSE

else

nex <- index <- trie

count <- 0

while index->NotLeaf and count < lenght(keyWord) and

index->pChildren[keyWord[count]-'a'] <> NULL do

next <- index->pChildren[keyWord[count]-'a']

index <- next

count <- count +1

end while

if next = NULL then

return TRUE

else

data <- next

if data->word = keyWord then

return TRUE

else

if data->pChildren[26]->word = keyWord then

return TRUE

else

return NULL

end

Metin Ağaçlarının Kullanım Alanları

* Metin ağaçlarındaki arama en kötü ihtimalle O(n) kadar bir karmaşıklığa sahip olduğundan ve arama hızı ağacın büyüklüğüyle değişmediğinden dolayı sözlük yapımında kullanılırlar.
* Arama motorlarındaki indexler sıkıştırlımış merin ağacnda tutulurlar.Metnin her bir yaprağı aranan bir kelimeyle ilgili veriyi tutar.
* Routerlar paket yönlendirme algoritmaları içerisinde metin ağaçlarını kullanırlar. Burada ağ alt geçini önek bir ağaçta tutar ve buradaki yollara göre yönlendirme yapar.