İKİLİ ARAMA AĞAÇLARINDA DÜĞÜMLERE HIZLI ULAŞMAK İÇİN BİR YÖNTEM VE GERÇEKLENMESİ

İbrahim ATEŞ, Nejat YUMUŞAK

***Özet*-Bu makalede arama ağaçları (Search Tree) üzerindeki işlemlerin daha hızlı yapılmasına yönelik bir yöntem ve birleşik bir veri yapısı önerilmektedir. Bu yöntemin sözcükler ve sayılar için nasıl kullanılabileceğine ve mevcut ağaç yapılarıyla (AVL, RB Ağacı gibi) karşılaştırılmasına yer verilmiştir. Bu amaçla bir hash tablosu ve dengeli bir ikili arama ağacı kullanılmaktadır. Veriye uygun olarak anlamlı alt ağaçlar oluşturulmakta ve bu alt ağaçlara hash tablosu yardımıyla ulaşılmaktadır. Çok sayıdaki verilerin tek bir ağaçta toplanıp bu büyük ağaçta işlem yapmaktansa alt ağaçlara bölerek daha az veri üzerinde işlem yapmak amaçlanmıştır. Bu şekilde veriler üzerindeki işlemler daha az eleman üzerinde yapılmaktadır. Bu da performansı olumlu yönde etkilemektedir.**

***Anahtar kelimeler* - Ağaç veri yapısı, RB ağacı, AVL ağacı, ağaçların performans karşılaştırmaları.**

***Abstract*-In this study, it is proposed a method that makes easy to make process in the search trees. Also it is given a data structure using this method. It is explained how this method is used for strings and numbers. It is shown performance comparison between other trees (like AVL, RB tree). A hash table and a blanced binary search tree are used to implement this data structure. It is builded the categorized subtrees according to data. Hash table is used to access data in the subtrees. It is aimed to process above relatively little amount of data instead of huge amount of data. In this way the number of the process will be decrease. It will be good affect for the program performance.**

***Keywords* - Tree Data Structures, RB Tree, AVL Tree, Performance comparision of trees.**

İ.Ateş;Bilgisayar ve Bilişim Mühendisliği Bölümü, Fen Bilimleri Enstitüsü, Sakarya Üniversitesi, 54187 Esentepe Sakarya
e-posta:ibrahimates@yahoo.com
N.Yumuşak;Bilgisayar Mühendisliği Bölümü, Muhendislik Fakultesi, Sakarya Üniversitesi, 54187 Esentepe Sakarya
e-posta: nyumusak@sakarya .edu.tr

I. GİRİŞ

Bir yazılım için kullandığı veri yapısı önemlidir. Eğer yazılım büyük miktarda verilerle çalışıyorsa kullandığı veri yapısı daha da önem taşımaktadır. Veri silme, veri ekleme veya veriler üzerinde arama, sıralama işlemleri ne kadar hızlı yapılabilirse yazılım o kadar etkin çalışır. Veriyi yerleştirme açısından veri yapıları ikiye ayrılırlar; Ardışıl (sequence) ve ilişkili (associative)[3].

Ardışıl veri yapıları, verileri değeriyle tutar. Bu uygun olmayan bir yapıdır çünkü istenilen bir elamana ulaşabilmek için O(n) adet tarama gerektirir. Bu nedenle veriyi anahtarıyla saklayabilen ve bu sayede daha hızlı bir erişime olanak veren yapılara ihtiyaç duyulmaktadır. Bu tip yapılar ilşkisel veri yapıları olarak isimlendirilirler. Bu tip yapılarda veriler pozisyonlarına göre değil değerlerine göre güncellenirler. Örnek olarak ağaç yapıları verilebilir[1].

AVL (Adelson-Velskii ve Landis) ağacı dengelenmiş bir ikili arama ağacıdır. Ağacın derinliği O(logn) seklinde ayarlanmaktadır. Bunun anlamı sağ ve sol alt ağaçlarının aynı derinliğe sahip olmasıdır. Herhangi bir düğüme bağlı alt ağaçların yükseklikleri arasındaki fark en fazla bir olmalıdır. Ağacı dengeli kurmak için AVL ağacı kullanılırsa, ağaç üzerindeki ekleme, arama ve silme gibi işlemleri yapan algoritmaların karmaşıklığı O(logN) kadar olur[2].

Red-Black ağacı her düğümü RED veya BLACK renk özeliğine sahip özel bir ikili arama ağacıdır. Ekleme hariç diğer ağaç işlemleri O(logn) zamanda yapılır. Herhangi bir eleman eklendiğinde tüm ağacın tekrar güncellenmesi gerekmektedir. Çünkü bir eleman eklendiğinde ağacın dengesi bozulmaktadır. Yeniden dengeleme işi dönüşüm (rotation) olarak bilinen basit bir değişiklikle sağlanmaktadır[1].

II. ÖNERİLEN YÖNTEM (SALKIM VERİ YAPISI)

Bu yöntemde, veri yapısı olarak bir hash tablosu ve RB ikili arama ağacı birlikte kullanılmaktadır. Hash tablosu için istenmeyen bir durum olan çatışma (collision) durumunun verileri sınıflandırmak için kullanılması söz konusudur. Kullanılan hash fonksiyonu aynı kategoriye ait değerler için aynı indis değerini üretmektedir[4].

Veriler ikili arama ağacının özelleştirilmiş bir şeklinde saklanır. Bu ağaç yapısında kök tek elemanlıdır ve sadece hash tablosuyla ağaç arasındaki bağlantıyı sağlamak amacıyla kullanılmaktadır. Kök, arama ağacına bağlantılı liste yardımıyla bağlanmaktadır.Veriler, tek bir ağaçta saklanmak yerine, anlamlı alt ağaçlara bölünerek saklanmaktadır. Veriler üzerinde herhangi bir işlem yapılmak istendiğinde, tüm ağaç yerine verinin ait olduğu alt ağaç üzerinde işlemler yapılmaktır.

II.1. Sözcükler İçin Gerçekleme

Sözcükler için bu yapı kullanıldığında, alfabedeki her harf için hash tablosunda bir kayıt oluşturulur. Hash tablosuna yerleşim,

H(x) =asciicode(x)-65;

hash fonksiyonu yardımıyla bulunan indis değerine bağlı olarak yapılmaktadır. İndis değeri hash tablosunda, harfin hangi sırada yazıldığını gösteren bir değerdir. Hash tablosunun ikinci bölmesinde ise ilgili alt ağacın kök elemanının adresi bulunmaktadır. Bu adres yardımıyla alt ağaçlara dallanılır.

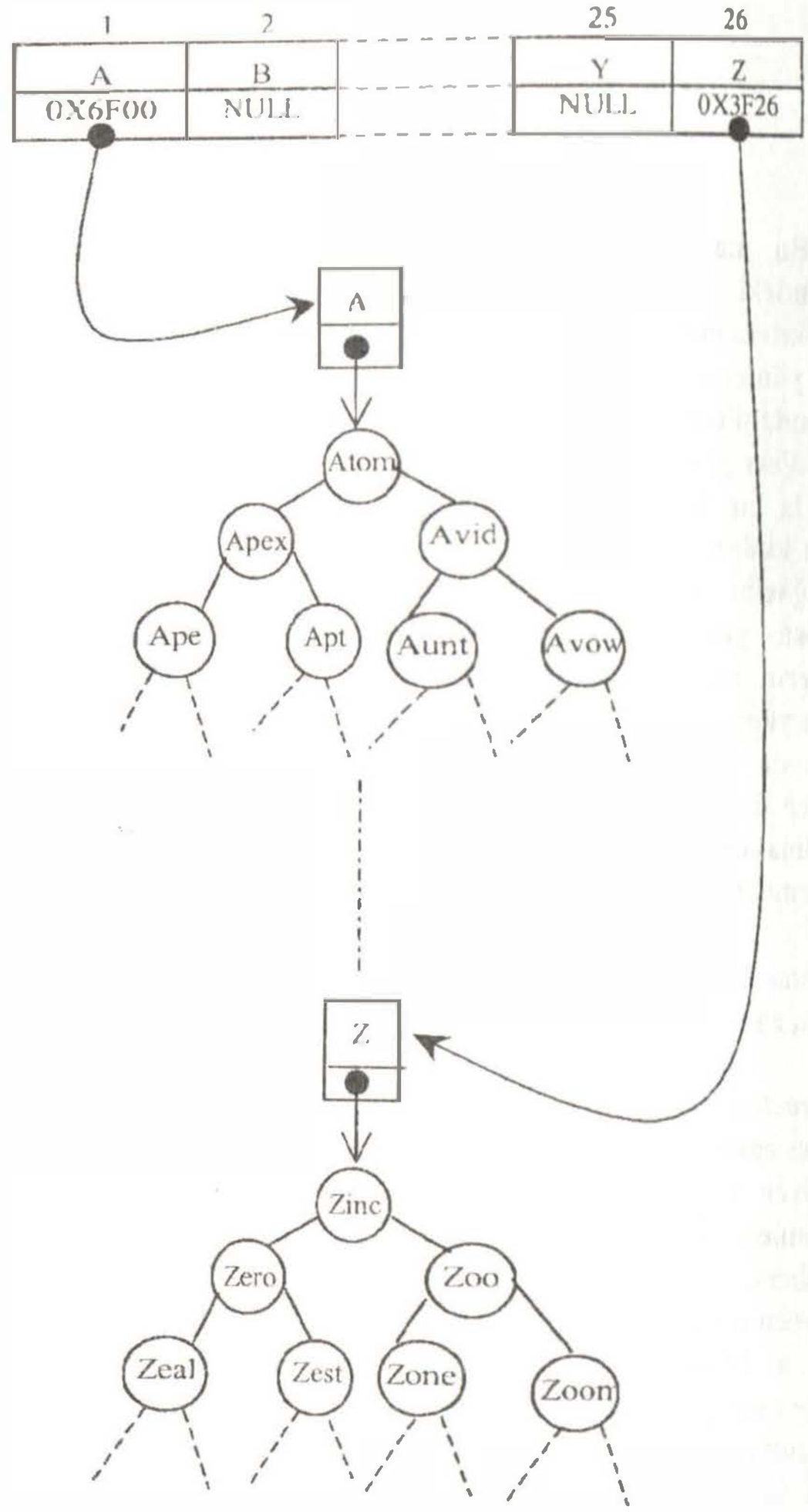
Hash tablosu statik olarak yaratılmaktadır. Sözcükler için yaratılan hash tablosunun birinci bölmesinde, A'dan Z'ye kadar harfler yer almaktadır. Hash tablosunun ikinci bölmesinde ise bu harflerle ilgili alt ağaçların adresleri bulunmaktadır. Başlangıçta adres alanları, NULL değerini göstermektedirler.

Bir harfe ait herhangi bir katar eklenmek istendiğinde o harfe ait alt ağaç olup olmadığına bakılır. Eğer varsa o alt ağaca giderek ekleme işlemi yapılır. Aksi taktirde alt ağaç için bir kök yaratılır ve katar, bu alt ağaca eklenir. Yaratılan kökün adresi hash tablosunda ilgili yere yazılır.

Arama yapmak için öncelikle, ilgili alt ağacın olup olmadığına bakılır. Eğer alt ağaç yaratılmamışsa, başka bir arama işlemine gerek duyulmadan, elemanın olmadığına karar verilir. Aksi taktirde ilgili alt ağaca dallanarak arama işlemi o alt ağaçta yapılır.

Örnek olarak "train" sözcüğünü bu yapıda aradığımızı düşünelim. Öncelikle "T" harfinin hash tablosundaki indisi, hash fonksiyonu yardımıyla bulunur. "T" harfinin indis değeri 19'dur. Hash tablosunun 19'uncu gözüne gidilir. Adres gözünde bir adres olup olmadığına bakılır. Eğer herhangi bir adres yoksa bu sözcüğün ağaçta olmadığına karar verilir. Aksi taktirde "T" alt ağacına dallanılır ve bu alt ağaçta arama yapılmış olur.

RB ikili ağacında aynı sözcük aransaydı verinin ağaçta olmaması durumunda bile tüm ağacın bu sözcük için taranması gerekecekti.



Şekil 1. Hash tablosuyla alt ağaçlara dallanma.

Bir sözcüğün herhangi bir yardımcı yöntem kullanılmadan RB ikili arama ağacında veya herhangi bir ikili arama ağacında saklanması durumunda "Train" sözcüğü aransaydı, "T" harfine gelene kadar bir çok düğüm ziyaret edilmiş olacak bu da arama işlemini yavaşlatmış olacaktı.

Örnek olarak 26000 elemanlı dengeli bir arama ağacında herhangi bir elemanın aranma maliyeti yaklaşık olarak 15'tir. Buna karşın bu yöntemde "T" harfi ile başlayan elemanların sayısı 1000 varsayılırsa arama maliyeti yaklaşık olarak 10 olur.

N toplam eleman sayısı ve N_i, i harfi ile başlayan elemanların sayısı olmak üzere,

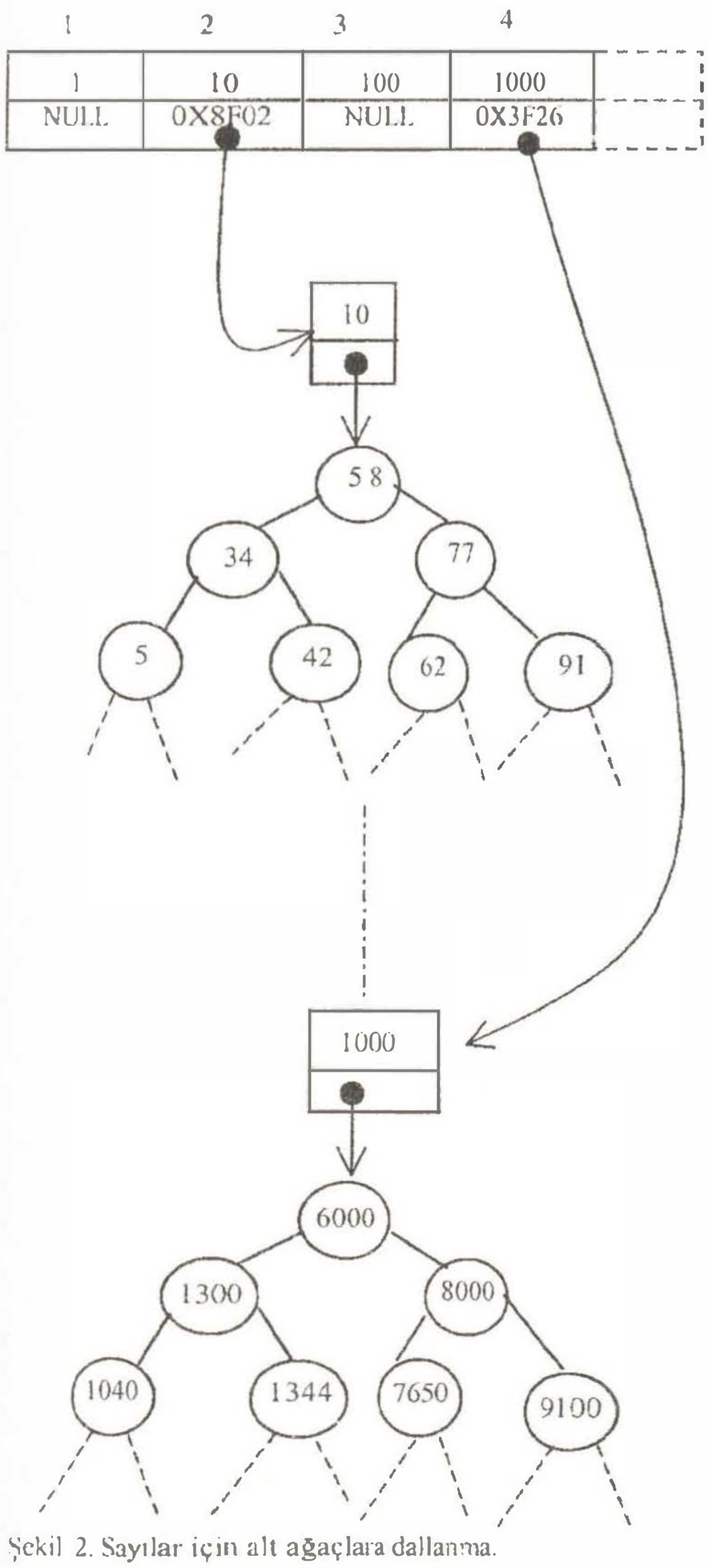
$N=N_A+N_B+ \ldots\ldots N_i \ldots +N_Z$

sözcüklerin hepsinin i harfi ile başlamadığı varsayımına dayanarak aşağıdaki durum sözkonusudur;

$Log_2N>log_2N_i$

O halde, burada sunulan yöntemin, genellikle sıradan bir ikili arama ağacından, daha iyi sonuç verdiği söylenebilir.

II.2. Sayılar İçin Gerçekleme



Şekil 2. Sayılar için alt ağaçlara dallanma.

Sayılar için bu veri yapısının kullanılabilmesi için aşağıdaki gibi bir hash fonksiyonu kullanılabilir.
H(x)=floor (abs(logx))

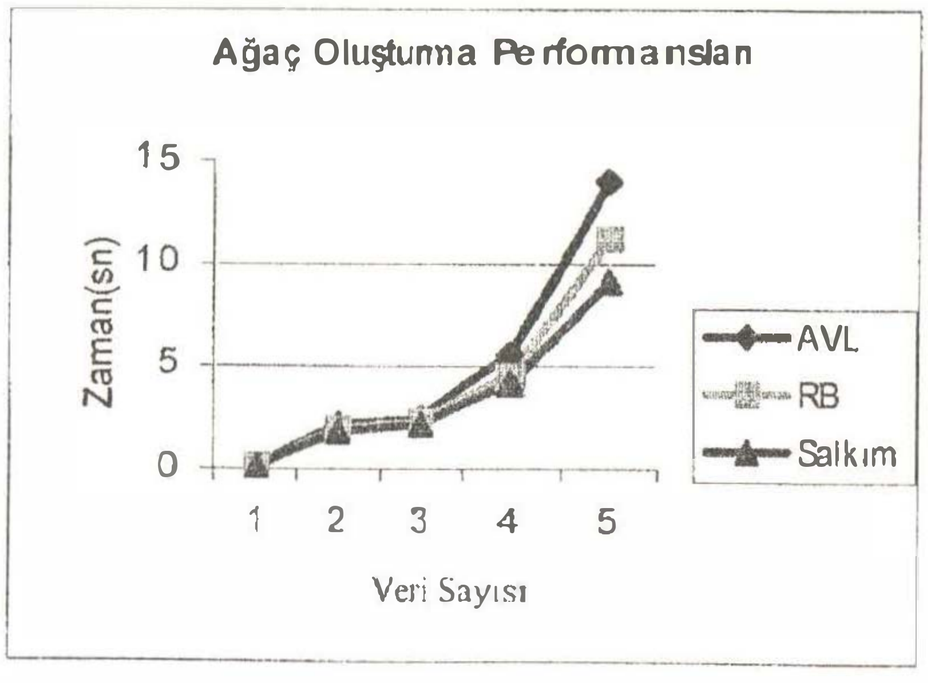
Verinin logaritma on tabanına göre logaritması alınır ve çıkan sonuçtan küçük ilk tamsayı değeri indis kabul edilir. Negatif ve pozitif sayılar için iki ayrı hash tablosu yaratmak uygun olacaktır. Negatif tamsayıların logaritması alınmadan önce mutlak değeri alınmalıdır. Ayrıca logaritma 0 tanımsız olduğu için 0 sayısına doğrudan 0 indisi verilir.

Örnek olarak 100 sayısı için indis olarak 2 üretecektir. 110 için de 2 üretmektedir. 1000 için 3, 10000 için 4 üretmektedir. Bu şekilde aralıklara göre alt ağaçlar üretilmektedir. O halde 10 alt ağacında minimum 10 değeri maksimum 99 değeri bulunacaktır. 1000 alt ağacında ise minimum 1000 maksimum ise 9999 sayısı bulunacaktır.

Bir arama yapılacağı zaman hash tablosu yardımıyla ilgili olan alt ağaca dallanılır ve arama bu alt ağaçta yapılır. Sayılar için üretilen hash tablosu dinamik olmaktadır. Gelen sayılara bağlı olarak hash tablosu oluşturulmaktadır.

III. ÖLÇÜMLER

Beş farklı dosya için üç veri yapısına ait, ağaç oluşturma performansları incelenmiştir. Birinci dosyada 25.010, ikinci dosyada 275.293, üçüncü dosyada 350.552, dördüncü dosyada 550.551, beşinci dosyada ise 1.101.100 eleman bulunmaktadır. Şekil 3'ün X eksenindeki rakamlar, bu sayıları temsil etmektedir. Örneğin grafikte, 275.293 eleman bulunan dosya 1 ile 1.101.100 eleman bulunan dosya ise 5 ile gösterilmiştir.

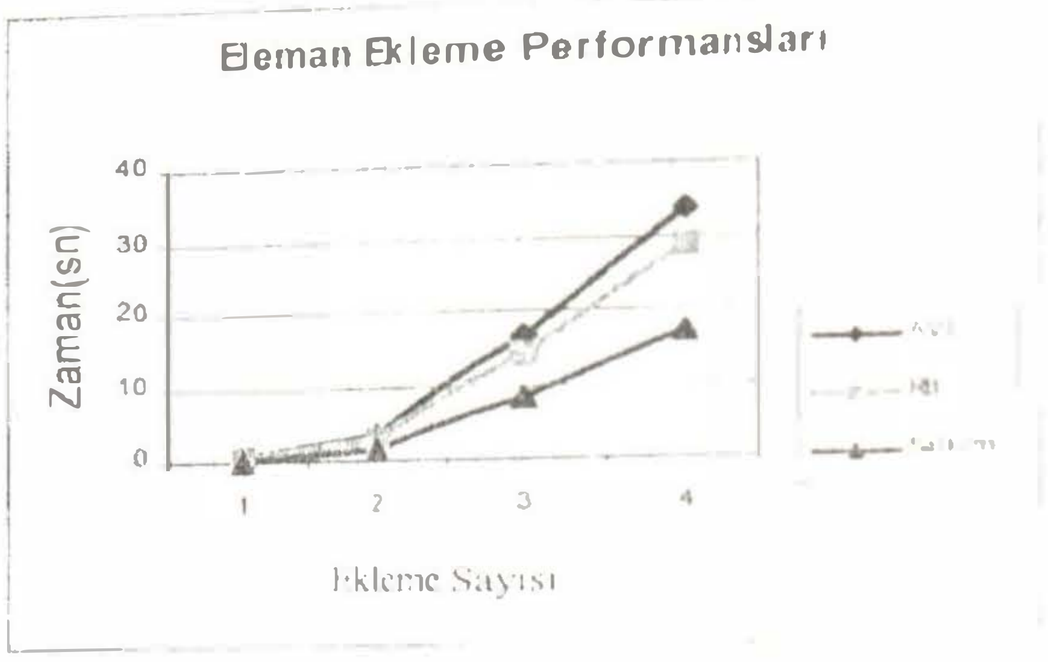


Şekil 3 . Ağaç oluşturma performansları.

Tablo 1. Ağaç oluşturmaya ilişkin ölçüm değerleri.

	Veri Sayısı	AVL	RB	Salkım
1	25.010	0,17	0,17	0,15
2	275.293	2,173	1,956	1,833
3	350.552	2,46	2,303	2,18
4	550.551	5,56	4,506	4,156
5	1.101.100	14,006	11,237	9,124

Ağaca eleman ekleme işlemine ait ölçüm değerleri Şekil 4'te görülmektedir. 100.000'den başlayarak 1.000.000, 5.000.000 ve 10.000.000 adet eleman ekleyerek, ağaç yapılarının eleman ekleme performansları incelenmiştir.

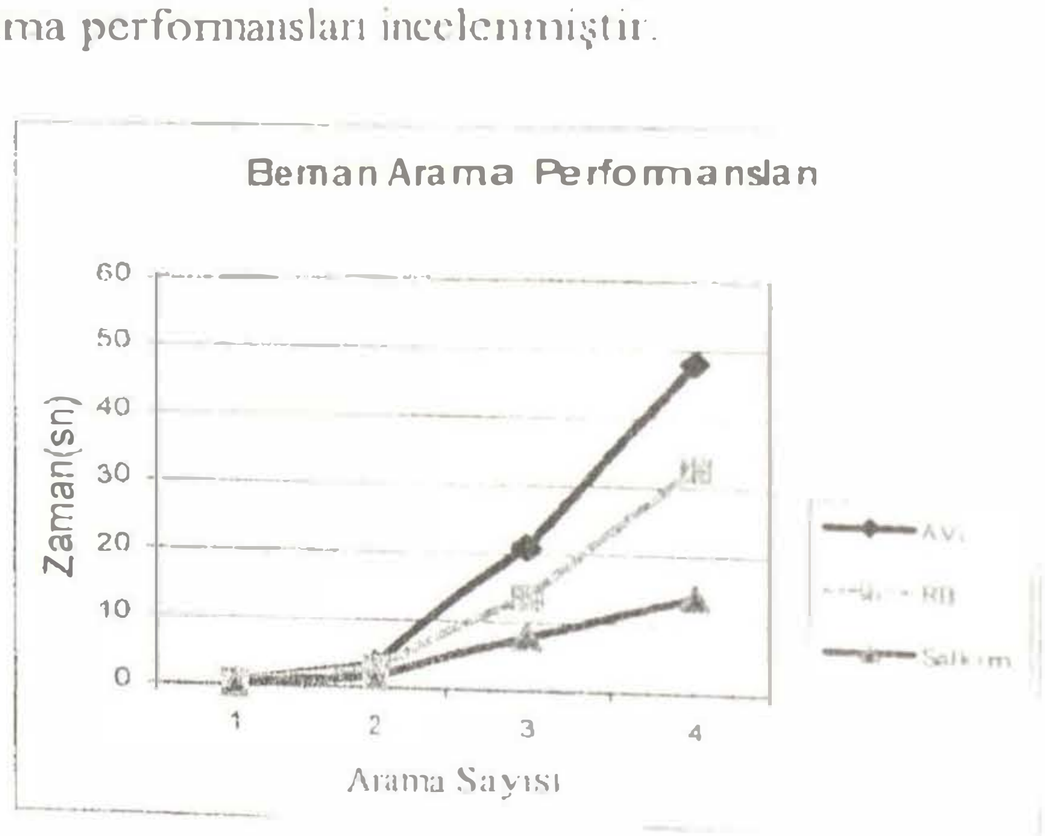


Şekil 4. Ağaca eleman ekleme performansları

Tablo 2. Eleman eklemeye ilişkin ölçüm değerleri

	Ekleme Sayısı	AVL	RB	Salkım
1	100.000	0,341	0,29	0,17
2	1.000.000	3,345	2,864	1,702
3	5.000.000	16,754	14,371	8,472
4	10.000.000	33,548	28,762	16,964

Şekil 5'te ağaç üzerinde eleman arama işlemine ait ölçüm değerleri görülmektedir. 100.000'den başlayarak 1.000.0000, 5.000.000 ve 10.000.000 eleman için ard arda arama işlemi yapılarak ağaç yapılarının eleman arama performansları incelenmiştir.



Şekil 5. Eleman arama performansları

Tablo 4. Eleman aramaya ilişkin ölçüm değerleri

	Arama Sayısı	AVL	RBTree	Salkım
1	100.000	0,421	0,33	0,141
2	1.000.000	3,395	2,744	1,542
3	5.000.000	20,92	13,71	7,752
4	10.000.000	48,037	32,837	14,01

IV. SONUÇ

Salkım modelinin, pek çok uygulamada tercih edilen AVL ve RB ağaçlarına gore, daha iyi sonuçlar ürettiği görülmüştür. Özellikle eleman arama ve eleman ekleme işlemde, veri sayısı arttıkça, Salkım modelinin performans artışı, diger ağaçlara oranla, daha da belirginleşmektedir.

KAYNAKLAR

[1] TREMBLAY J. P., "Data Structures and Software Development", Prentice Hall, 2003
[2] SEDGEWICK R., "Algorithms in C", Adisson Wessley, 2002
[3] FORD W., Top W., "Data Structures with C++ using STL", Prentice Hall ,2002
[4] ÇÖLKESEN R., "Veri Yapıları ve Algoritmalar", Papatya Yayıncılık,2002
[5] ROBSON R. , "Using the STL", Springer Publishing Company , 1999
[6] WEISS M. A., "Data Structures and Algorithm Analysis in C++", Addison-Wessley ,1994
[7] SCHILDT, H., "C++ Temel Öğrenim Kılavuzu", Alfa Yayınları, 1999
[8] SAHNI, Sartaj, "Data Structure,; Algorithms and Applications in C++", McGraw-Hill, 1997
[9] GONNET, G.H, "Handbook of Algorithms and Data Structure", Addison-Wessley Pub, 1984
[10] SHAFFER, C.A, "A Practica Introduction to Data Structure and Algorithm Analysis", Prentice-Hall,1997