

BÖLÜM 5

AĞAÇLAR

ANAHTAR KAVRAMLAR

Ağaç, İkili Ağaç, İkili Arama Ağacı, AVL Ağacı



ÖĞRENME HEDEFLERİ

-  1 Ağaç veri yapısının temellerini bilir ve uygulayabilir
-  2 İkili ağaç yapısını farklı veri yapıları oluşturmak için kullanabilir
-  3 İkili arama ağaçlarını tanımlayabilir ve uygulayabilir
-  4 AVL ağaçlarını tanımlar ve uygulayabilir

GİRİŞ

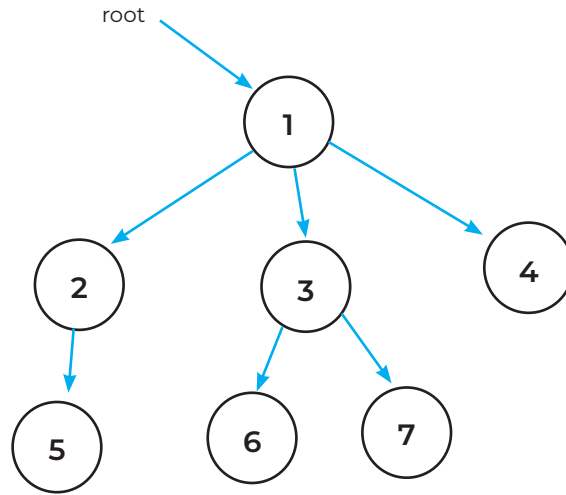
Bu bölümde ağaç veri yapısı anlatılacaktır. Ağaç veri yapısı doğrusal olmayan bir veri yapısıdır ve düğümler arasında hiyerarşik bir ilişki vardır. Bu veri yapısında kök (root) düğüm dışında tüm düğümlerin bir ebeveyn (parent) düğümü vardır. Bir ebeveyn düğüme bir dal (edge) ile bağlı olan düğüme çocuk (child) denir. Çocuğu olmayan düğümlere yaprak (leaf) düğüm adı verilir. Eğer bir ağaç veri yapısında bir düğümün en fazla iki çocuğu olabiliyorsa bu ağaç yapısına ikili (binary) ağaçlar denir. İkili ağaçların belirli kurallar çerçevesinde tanımlanan farklı biçimleri vardır. İkili arama ağaçları ve AVL ağaçları farklı formlarda tanımlanmış ikili ağaçlardır. Bu bölümde detaylı olarak bu ağaç tipleri üzerinde gerçekleştirilecek işlemler anlatılacaktır.



1. AĞAÇLAR

Ağaç veri yapısı düğümler arasında ebeveyn-çocuk ilişkisi kuran hiyerarşik bir veri yapısıdır. Bir ağaç veri yapısı kök düğümü ile başlar. Kök düğümün ebeveyn düğümü bulunmamaktadır. Kök düğüm dışında tüm düğümlerin bir ebeveyn düğümü vardır. Bir düğümün birden fazla ebeveyni olamaz. Bu özelliği dolayısıyla ağaç veri yapısı çizgelerden ayrılır. Çizgelerde her bir düğüm birden fazla düğüm ile bağlantı oluşturabilir. Bu nedenle ağaç veri yapısı çizgelerin bir alt kümesi, bir özel durumu olarak değerlendirilebilir. Kök düğümünden başlayarak ağaç veri yapısında yer alan tüm düğümlere ulaşmak mümkündür. Genel ağaç yapısında her bir düğümün bir veya birden fazla çocuğu olabilir. Çocuğu olmayan düğümlere yaprak düğüm adı verilir. Ağaç içerisinde yapraklar son düğümler olarak değerlendirilir.

Şekil 5.1'de verilen ağaç yapısında 7 düğüm vardır. Kök düğümün 3 çocuğu vardır. Burada verilen her bir çocuk düğüm (2, 3 ve 4) aynı zamanda kendi alt ağaçlarının kök düğümüdürler. Dolayısı ile bir ağaç veri yapısı, özyinelemeli olarak her bir düğümün bir alt ağacın kök düğümü olduğu yaklaşımı ile de tanımlanabilir. Ağaç veri yapısının yüksekliği ağacın kökünden en altta yer alan yaprak düğüme kadar ulaşan yolun uzunluğu olarak tanımlanır. Bir düğümün derinliği ise kök düğüme olan uzaklığıdır. Kök düğümün derinliği 1 olarak kabul edilir. Şekil 5.1'de yer alan ağacın yüksekliği 2'dir. Aynı ağaçta yer alan 6 düğümünün derinliği 3, 4 düğümünün derinliği 2'dir. Aynı ebeveyne sahip düğümler kardeş (sibling) düğümler olarak değerlendirilirler.



Şekil 5.1. Örnek bir ağaç veri yapısı

Bir ağaç yapısında eğer bir ebeveyn en fazla iki çocuk ile sınırlanmışsa bu tür ağaç veri yapısına ikili ağaç yapısı ismi verilir. Bu tür ağaç yapısında

düğümlerde hiç çocuk olmayabilir, bir çocuk olabilir veya iki çocuk olabilir. İkili ağaç yapısında kullanılan düğüm yapısı Şekil 5.2'de verilmiştir.



ARAŞTIRALIM

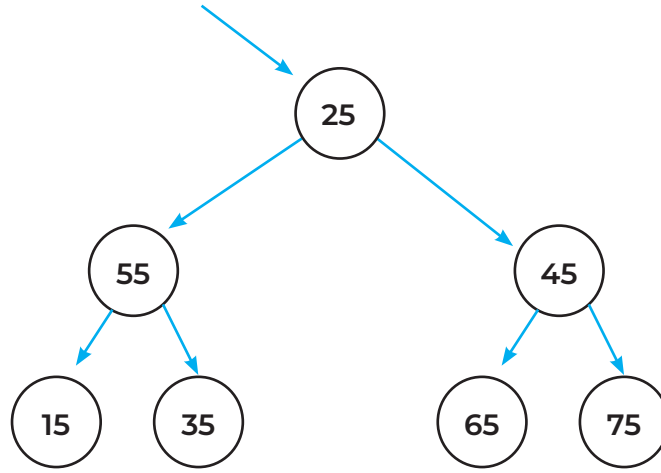
Bu bölümde ağaçlar için bağlı listeler kullanılmıştır. Ağaçların uygulanmasında dizilerin kullanımını araştırılabilir.

Şekil 5.3'te 7 düğümden oluşan örnek ikili ağaç yapısı verilmektedir. Bu ağaç yapısında düğümlerde yer alan değerler arasında herhangi bir düzen verilmemiştir. Ancak sonraki alt bölümler-

de düğümlerde yer alan veriler arasında belirli düzenlerin olduğu ikili ağaç tipleri de gösterilecektir.

```
1 struct treeNode
2 {
3     int value;
4     struct treeNode *leftChild;
5     struct treeNode *rightChild;
6 };
```

Şekil 5.2. İkili ağaç yapısında kullanılan düğüm yapısı



Şekil 5.3. Örnek bir ikili ağaç veri yapısı



DİKKAT EDELİM

Yaprak düğümlerde sol çocuk ve sağ çocuk düğümleri göstermek için kullanılan işaretçiler NULL değerine sahiptir.

Bu düğüm yapısında veri olarak sadece tam sayı kullanılmıştır. Ayrıca sol çocuğu gösteren leftChild işaretçisi ve sağ çocuğu gösteren

rightChild işaretçisi bu düğümün diğer elemanlarıdır. Yaprak düğümlerde her iki işaretçinin değeri NULL olarak yer alır.



ARAŞTIRALIM

Genel bir ağaç yapısını, ikili ağaç yapısına dönüştürme yöntemini araştıralım.

Bir ikili ağaç veri yapısında en önemli işlemlerden birisi ağaç üzerinde gezinme işlemidir. Temel olarak üç farklı gezinme yöntemi vardır.

1. Preorder Gezinme
2. Inorder Gezinme
3. Postorder Gezinme

```

1 void preorder(struct treeNode *tree)
2 {
3     if(tree!=NULL)
4     {
5         cout<<tree->value<<' ';
6         preorder(tree->leftChild);
7         preorder(tree->rightChild);
8     }
9 }

```

Şekil 5.4. Preorder gezinme fonksiyonu

Preorder gezinme yönteminde kök düğüm çocuklardan önce ziyaret edilir veya bir başka ifade ile işlem görür (Şekil 5.4). Daha sonra sol çocuk ve son olarak sağ çocuk ziyaret edilir. Sol veya sağ çocuk ziyaret edildiğinde bu düğümler, bir alt ağacın kök düğümleri olarak düşünülürler. Dolayısı ile özyinelemeli olarak çağrılan **preorder** gezinme fonksiyonu bu düğümleri kök olarak kabul eder. Şekil 5.3'te verilen ikili ağaçta yer alan düğümlerin **preorder** gezinme ile ziyaret sırası aşağıdaki gibidir.

25 55 15 35 45 65 75

Öncelikli olarak **preorder** gezinme fonksiyonuna kök 25 düğümü gönderildiğinde bu değer ekrana yazdırılır. Sonra sol çocuk ziyaret edilir. Bu düğüm sol alt ağacın kök düğümüdür. Bu nedenle ekrana yazdırılır ve bu düğümün sol çocuğu olan 15 düğümüne gelinir. 15 ekrana yazdırılır. 15 değerine sahip olan düğümün önce sol çocuğuna sonra sağ çocuğuna bakılır. Her ikisi de NULL olduğu için, 15 değerinin kök olduğu ikili ağacın işlemi tamamlanır. Bu durumda işlem sırası, kök değeri 35 olan alt ağaca geçer ve bu süreç bu şekilde devam eder.

```

1 void inorder (struct treeNode *tree)
2 {
3     if(tree!=NULL)
4     {
5         inorder(tree->leftChild);
6         cout<<tree->value<<' ';
7         inorder(tree->rightChild);
8     }
9 }

```

Şekil 5.5. Inorder gezinme fonksiyonu

Inorder gezinme işleminde kök düğüm ziyaret edilmeden önce sol çocuk ziyaret edilir (Şekil 5.5). Sol alt ağaçta yer alan işlemler tamamlandıktan sonra kök düğüm, daha sonra sağ alt düğüm işlem görür. Tabii ki sol çocuk veya sağ çocuk ziyaret edildiğinde bu düğümler ilgili alt ağacın

kök düğümü olarak işlem görürler. Özyinelemeli olarak çağrılan **inorder** fonksiyonu bu düğümleri kök olarak alır. Şekil 5.3'te verilen ikili ağaçta yer alan düğümlerin **inorder** gezinme ile ziyaret sırası aşağıdaki gibidir.

15 55 35 25 65 45 75

```

1 void postorder (struct treeNode *tree)
2 {
3     if(tree!=NULL)
4     {
5         postorder(tree->leftChild);
6         postorder(tree->rightChild);
7         cout<<tree->value<<' ';
8     }
9 }

```

Şekil 5.6. Postorder gezinme fonksiyonu

Postorder gezinme işleminde önce sol alt ağaç, sonra sağ alt ağaç ve son olarak kök düğüm ziyaret edilir (Şekil 5.6). Şekil 5.3'te yer alan ikili ağaç üzerinde **postorder** gezinme gerçekleştirdiğimizde elde edilen gezinme sırası aşağıdaki gibidir.

15 35 55 65 75 45 25

Eğer **inorder** gezinme sonucu ile beraber **pre-order** veya **postorder** gezinme sonucu verilmiş ise ikili ağacı tekrar oluşturmak mümkündür.



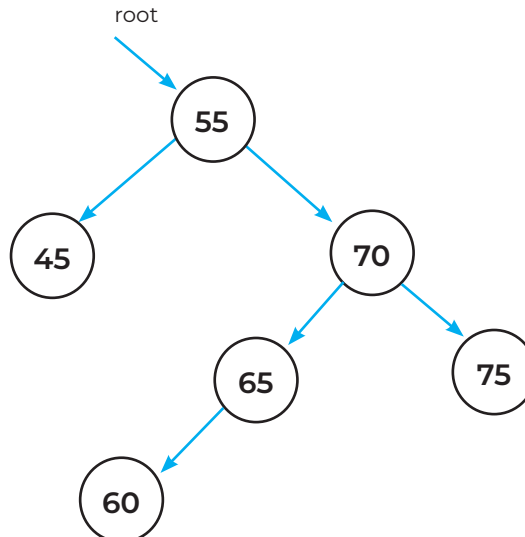
ARAŞTIRALIM

Preorder veya postorder gezinme ile inorder gezinme sonuçlarının beraber verildiği durumda ikili ağaç yapısını oluşturmayı araştıralım.

2. İKİLİ ARAMA AĞAÇLARI (BINARY SEARCH TREES)

Genel ikili ağaç yapısında, düğümlerde yer alan değerlerin büyüklükleri arasında herhangi bir ilişki yoktur. İkili arama ağaçları ikili ağaç yapısının özel bir formudur ve düğümlerin değerleri arasında büyüklük-küçüklük ilişkisi söz konusudur. Sıralı bir listedeki arama işlemini verimli bir şekilde gerçekleştirmek için geliştirilmiş bir yapıdır. Arama, ekleme ve silme işlemleri temel olarak **O(logn)** sürede gerçekleştirilir. Bir düğümün değeri, sol alt ağaçtaki tüm değerlerden büyük,

sağ alt ağaçtaki tüm değerlerden küçük veya eşit olmalıdır. Şekil 5.7'de örnek bir ikili arama ağacı verilmiştir. Kök düğümde yer alan 55 değeri sol alt ağaçtaki her değerden büyük, sağ alt ağaçtaki her değerden küçüktür. Aynı şekilde bu kural ikili arama ağacındaki tüm düğümler için geçerlidir. Sağ alta ağaçta 70 değeri kendisinin sol alt ağacındaki tüm değerlerden büyük, sağ alt ağacındaki tüm değerlerden küçüktür.



Şekil 5.7. Örnek bir ikili arama ağacı

Şimdi de ikili arama ağaçlarında arama işleminin nasıl gerçekleştirildiğine bakalım. Arama işlemi için önce aranan değer kökte yer alan değer ile karşılaştırılır. Eğer aranan değer kök değerinden küçükse sağ alt ağaç elenir ve arama işlemi sol alt ağaç ile devam eder. Aksi takdirde aranan değer kök değerinden büyükse sol alt ağaç elenir arama işlemi sağ alt ağaç ile devam eder. Eğer kök değeri aranan değer ile eşit ise arama işlemi başırlı bir şekilde sonlandırılır ve ilgili düğümün adresi çevrilir. Bu süreç özyinelemeli olarak bu şekilde devam eder. Eğer alt ağaçlar ile devam etme sürecinde NULL değerine ulaşırsa aranan

değerin ikili arama ağacında olmadığı anlaşılır. Örnek olarak 73 değerini Şekil 5.7'de arayalım. 73 değeri kökte yer alan 55 değeri ile karşılaştırılır. 73 55'ten büyük olduğu için arama işlemine 55 düğümünün sağ alt ağacı ile devam edilir. 73 değeri 70 ile karşılaştırılır ve büyük olduğu için 70 düğümünün sağ alt ağacı ile devam edilir. İlgili sağ alt ağacın kök değeri 75 ile karşılaştırılır ve bu değerden küçük olduğu için sol alt ağaca devam edilir. Ancak 75'in sol alt ağacı NULL'dır ve 73 değerinin ikili arama ağacında olmadığı anlaşılır. Arama işlemi için yazılan **search** fonksiyonu Şekil 5.8'de verilmiştir.

```

1 struct treeNode* search(struct treeNode *tree, int sValue)
2 {
3     if(tree==NULL || tree->value == sValue)
4         return tree;
5     else
6     {
7         if(sValue < tree->value)
8             return search(tree->leftChild, sValue);
9         else
10            return search(tree->rightChild, sValue);
11     }
12 }
```

Şekil 5.8. İkili arama ağacında bir değeri arama fonksiyonu

İkili arama ağacına yeni bir değer eklemek için arama işleminde olduğu gibi kök düğümdeki değer ile eklenmek istenen değer karşılaştırılır. Eğer eklenmek istenen değer küçükse sol alt ağaç ile büyükse sağ alt ağaç ile devam edilir. Bu süreç

NULL bir sol işaretçi veya sağ işaretçiye ulaşıncaya kadar devam eder. Şekil 5.9'da ikili arama ağacına bir değer eklemek için yazılan **insert** fonksiyonu verilmiştir.

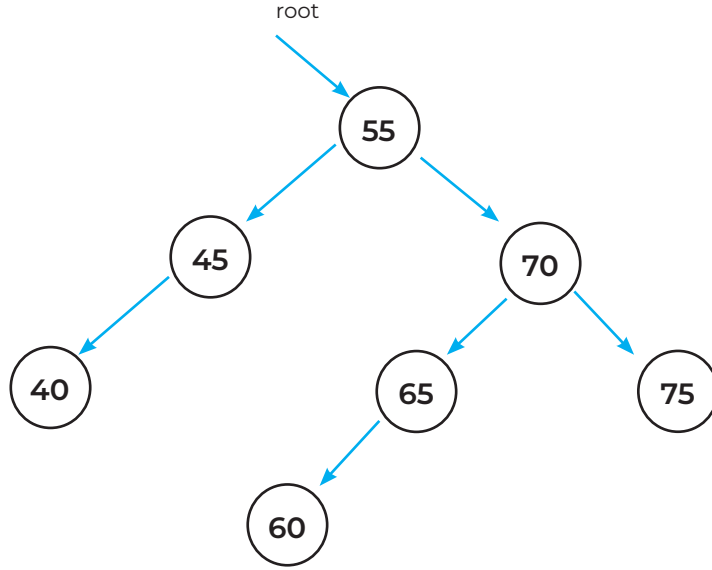
```

1 void insert(struct treeNode *tree, int nValue)
2 {
3     if(tree==NULL)
4     {
5         tree=new treeNode;
6         tree->value=nValue;
7         tree->leftChild=NULL;
8         tree->rightChild=NULL;
9     }
10    else
11    {
12        if(nValue < tree->value)
13            insert(tree->leftChild, nValue);
14        else
15            insert(tree->rightChild, nValue);
16    }
17 }
```

Şekil 5.9. İkili arama ağacına bir değeri ekleme fonksiyonu

Şekil 5.7'de yer alan ikili arama ağacına 40 değerini ekleyelim. 40 kök düğümde yer alan 55 değerinden küçük olduğu için işleme sol alt ağaç ile devam edilir. 40 sol alt ağacın kök düğümünde yer alan 45'ten de küçüktür. Dolayısıyla işleme 45 düğümünün sol alt ağacı ile devam edilir. Bu

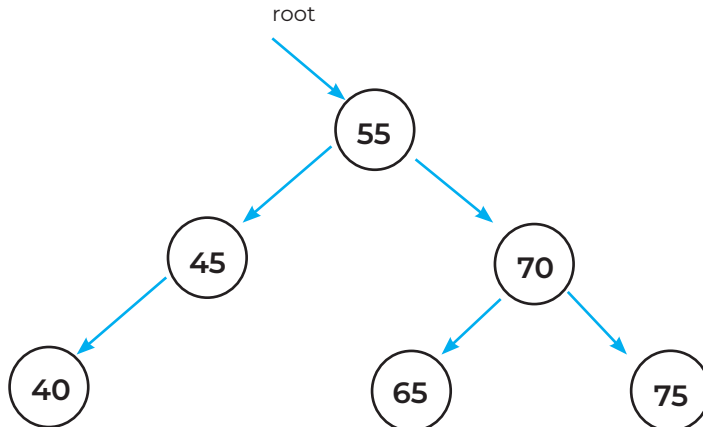
düğümün sol alt ağacı gösteren sol işaretçi değeri NULL olduğu için yeni düğümü ekleme noktası bulunmuş olur. 45 düğümünün sol çocuğu olarak 40 düğümü ikili arama ağacına eklenir. Yeni ikili arama ağacı Şekil 5.10'da verilmiştir.



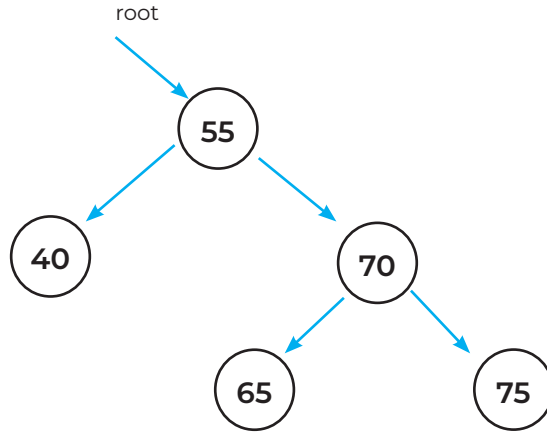
Şekil 5.10. Şekil 5.7’de yer alan ikili arama ağacına 40 düğümünün eklenmesi ile oluşan yapı

Bir düğümü ikili arama ağacından silmek için önce ilgili düğümün ağaçta arama işlemi gerçekleştirilir. Eğer düğüm ağaç yapısı içinde mevcutsa silme işlemi gerçekleştirilir. Bir ikili arama ağacından bir düğümü silme işlemi için üç farklı durum söz konusudur. Bu durumlar silinecek düğümün kaç tane çocuğunun olması ile ilgilidir. Bir düğümün hiç çocuğu olmayabilir, bir çocuğu olabilir veya iki çocuğu olabilir. En kolay silme işlemi düğümün hiç çocuğunun olmaması durumudur. Düğüm ağaç yapısından uzaklaştırılır. Örneğin Şekil 5.10’da yer alan ikili arama ağacından 60 değerine sahip düğümü silmeye çalışalım. Bu düğümün çocuğu yoktur ve düğüm doğrudan farklı bir işleme gerek kalmadan silinir. Elde edilen ikili arama ağacı Şekil 5.11’de verilmiştir. İkinci durum ise sadece bir çocuğu olan düğümün silinme durumudur. Bu durumda silinecek

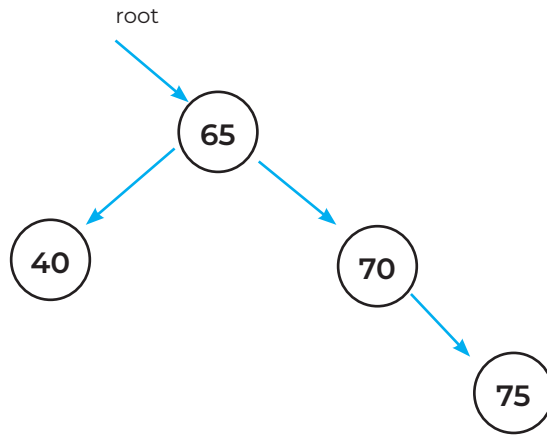
düğümün yerine tek çocuğu yerleştirilir. Şekil 5.11’deki ikili arama ağacından tek çocuğu olan 45 düğümü silindiğinde elde edilen ağaç yapısı Şekil 5.12’de verilmektedir. Son durum iki çocuğa sahip bir düğümün ikili arama ağacından silinmesidir. Bu durumda silinecek düğümün halefi (successor) bulunur ve silinecek düğümün yerine yazılır. Şekil 5.12’de verilen ikili arama ağacından kök düğümün silinme işlemi bu duruma örnektir. 55 değerine sahip olan düğümün iki çocuğu vardır. Bu düğümün halefi sağ alt ağaçta yer alan en küçük değere sahip düğümdür. Bir başka ifade ile sağ alt ağaçta en solda yer alan düğümdür. Bu düğüm ilgili örnekte 65 değerine sahiptir. 55 değerine sahip düğüm silinir ve 65 değerine sahip halefi ikili arama ağacında onun yerini alır. Bu durum Şekil 5.13’te yer almaktadır.



Şekil 5.11. İkili arama ağacından çocuğu olmayan 60 düğümünün silinmesi ile oluşan yapı



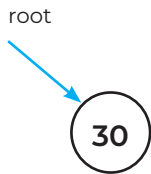
Şekil 5.12. İkili arama ağacından bir çocuğu olan 45 düğümünün silinmesi ile oluşan yapı



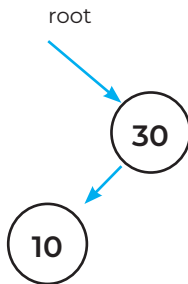
Şekil 5.13. İkili arama ağacından iki çocuğu olan 55 düğümünün silinmesi ile oluşan yapı

Bir listedeki değerlerin sırasıyla ikili arama ağacına yerleştirilmesi işleminde değerlerin listedeki yerine göre farklı ağaçlar oluşabilir. Örnek olarak 30, 10, 20 ve 40 değerlerinin bu sırada ikili arama ağacına yerleştirildiğini düşünelim. Ağaç boşken ilk düğüm 30 kök olarak eklenir (Şekil 5.14. (a)). Listedeki ikinci değer 10, kök düğümde yer alan 30 değerinden küçüktür ve sol çocuk olarak eklenir (Şekil 5.14. (b)). Bir sonraki değer

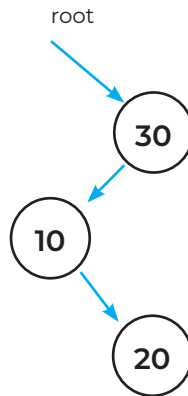
20, kök düğümün değeri 30 dan küçüktür. Yeni değerinin ekleneceği nokta için sol alt ağaç ile devam edilir. Sol alt ağacın kök değeri 10 ile yeni değer 20 karşılaştırılır. 20 bu değerden büyüktür ve 10 değerinin sağ çocuğu olarak ağaca eklenir (Şekil 5.14. (c)). Son değer 40 kök 30 değerinden daha büyüktür ve kökün sağ çocuğu olarak eklenir (Şekil 5.14. (d)).



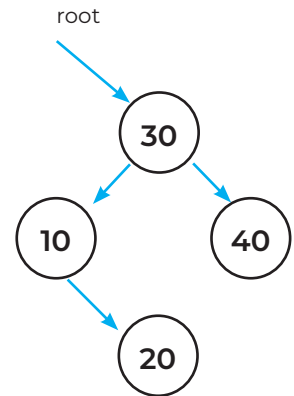
(a)



(b)



(c)



(d)

Şekil 5.14. İkili arama ağacına 4 değer sıralı olarak eklenmesi

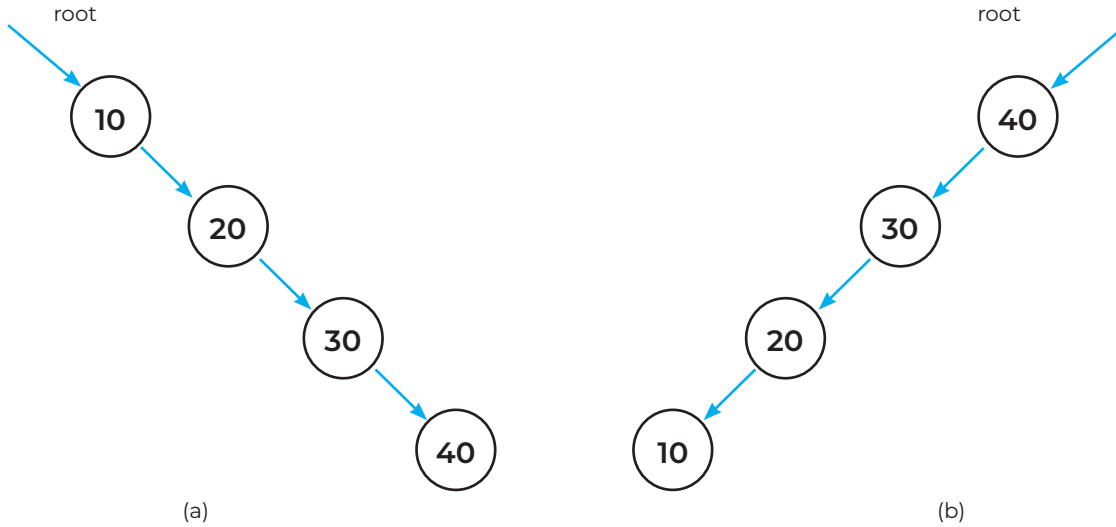
Aynı sayılar eğer farklı sırada, ikili arama ağacına yerleştirilirse Şekil 5.14'te yer alandan farklı bir ağaç yapısı elde edilir. Eğer listede yer alan elemanlar 10, 20, 30 ve 40 sıralaması ile ikili arama ağacına eklenirse Şekil 5.15. (a)'da yer alan ağaç elde edilir. Eğer listedeki elemanların düzeni 40, 30, 20 ve 10 şeklinde azalan sırada verilmişse Şekil 5.15. (b)'de yer alan ikili arama ağacı oluşur. Buradaki örneklerden görüldüğü gibi ikili arama ağaçları doğrudan elemanların eklenme sırasına bağlıdır. Eğer elemanlar arasında artan veya azalan bir sıralama söz konusu ise Şekil 5.15'teki gibi

sırasıyla sağa dayalı (right-skewed) ve sola dayalı (left-skewed) ikili arama ağaçları ortaya çıkabilir. Sağa dayalı ve sola dayalı ikili arama ağaçları Şekil 5.15'te de görüleceği gibi doğrusal yapılar oluşturur. Bu doğrusal yapılar, ekleme ve arama işlemleri için ikili arama ağaçlarının, hedeflenen çalışma zamanına ulaşmasını ($O(\log n)$) engeller. Her iki yapıda doğrusal çalışma zamanına sahiptir. Bazı durumlarda ikili arama ağaçlarının beklenen çalışma zamanı performansını göstermemesi AVL ağaçlarının oluşturulmasına yol açmıştır.



DİKKAT EDELİM

İkili arama ağaçlarında elemanların ağaca eklenme sırasına göre sola dayalı veya sağa dayalı doğrusal yapılar ortaya çıkabilir.



Şekil 5.15. İkili arama ağacına 4 değer farklı sıralarda eklenmesi

3. AVL AĞAÇLARI

Bir önceki kısımda verilen örnekte bazı durumlarda özellikle sıralı listelerde ikili arama ağaçlarının doğrusal yapılar oluşturabileceği görülmüştü. Sağa dayalı ve sola dayalı ikili arama ağaçları olarak isimlendirilen bu durumlar ikili arama ağaçlarının verimli şekilde çalışmasını engellemektedir. Bu duruma çözüm olması amacıyla

AVL ağaçları geliştirilmiştir. AVL ağaçlarındaki temel amaç sol ve sağ alt ağaçlar arasındaki yükseklik farkının belirli bir sınır çerçevesinde tutulmasıdır. İki alt ağaç arasındaki yükseklik farkı en fazla 1 olabilir. Bu değere denge faktörü (balance factor) adı verilir. Alt ağaçlarının yüksekliği eşit ise denge faktörü 0, sol alt ağaç yüksekliği 1

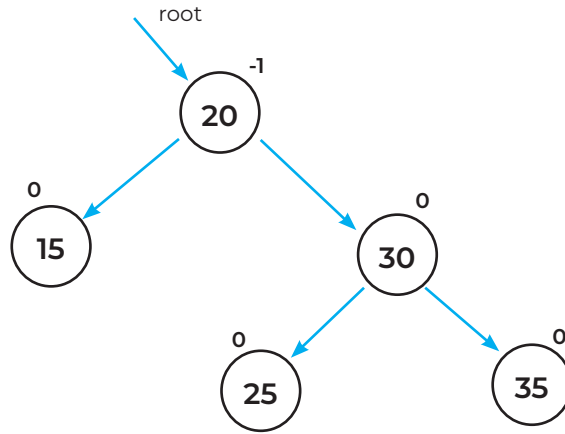
fazla ise denge faktörü 1 ve 1 eksi denge faktörü -1 olarak hesaplanır. -1, 0 ve 1 denge faktörü değerleri AVL ağacındaki düğümler için kabul edilebilir değerlerdir. Bu değerlerden farklı bir denge faktörüne sahip bir düğüme sahip ağaç AVL ağacı olarak değerlendirilmez. Şekil 5.16'da 5 düğümden oluşan bir AVL ağacı yer almaktadır. Düğümlerin yanında denge faktörü değerleri

de verilmektedir. Burada önemli olan nokta AVL ağaçlarının aynı zamanda ikili arama ağacı özelliğine de sahip olmalarıdır. Bu özelliğe ek olarak denge faktörü kavramı ile yükseklik dengeli (height-balanced) bir ağaç oluştururlar. Bu şekilde ikili arama ağaçlarının doğrusal yapı oluşturma ihtimali ortadan kaldırılmış olur.



ARAŞTIRALIM

AVL ağaçları dışında ikili arama ağaçlarının daha verimli kullanılmasını sağlayan farklı ağaç tiplerini araştırınız.



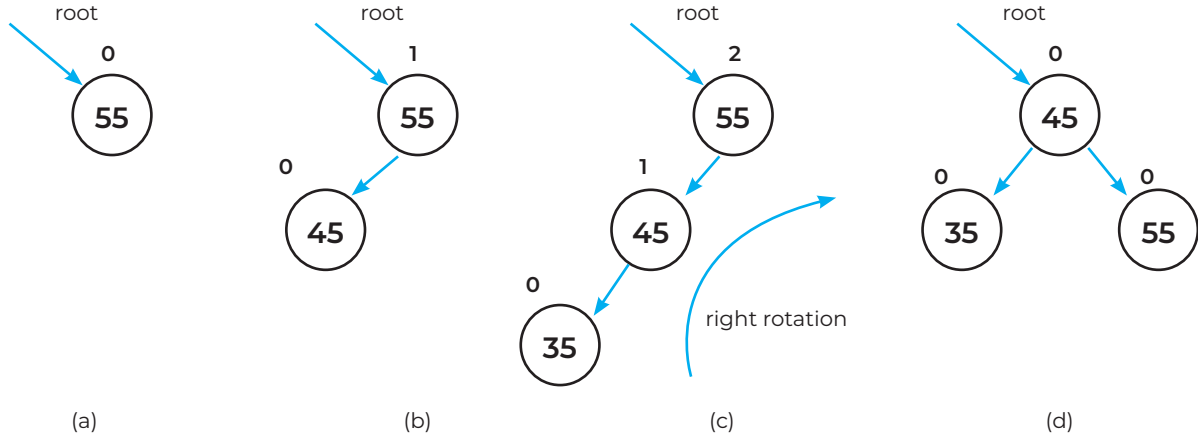
Şekil 5.16. 5 düğümden oluşan örnek bir AVL ağacı

AVL ağacında arama ve ekleme işlemleri, ikili arama ağacındaki işlemlere benzer şekilde gerçekleştirilir. Arama işlemi birebir aynıdır. Ancak ekleme işleminden sonra düğümlerin denge faktörlerinde bozulma olabilir. Bir başka ifade ile denge faktörü -1, 0 ve 1'den farklı değerlere sahip olabilir. Bu durumda denge faktörlerinde oluşan bozulmayı düzeltmek için sola rotasyon (left rotation) veya sağa rotasyon (right rotation) işlemleri gerçekleştirilir. Şekil 5.17'de bu duruma bir örnek verilmiştir. Önce 55 değeri boş bir AVL ağacına eklenmiştir. Bu ilk düğümün denge faktörü 0'dır. Daha sonra 45 eklenmek istendiğinde 55 değerinden küçük olduğu için bu düğümün soluna ekleme işlemi gerçekleşir. Tabii ki ekleme

işleminin yapıldığı noktadan kök düğüme kadar ulaşan yoldaki tüm düğümlerin denge faktörleri güncellenir. Güncelleme sonrasında 55 değerine sahip düğümün denge faktörü 1 olarak hesaplanır. Sol alt ağacın yüksekliği sağ alt ağacın yüksekliğinden 1 fazladır. Bu durumda AVL özelliği bozulmaz. Yeni eklenen 45 düğümünün denge faktörü 0'dır. Görüleceği gibi yeni eklenen düğümler yaprak düğüm olarak eklendikleri için sol çocuk ve sağ çocuk işaretçi değerleri NULL'dır ve iki alt ağaç arasındaki yükseklik farkı 0'dır. Şekil 5.17 (b)'deki AVL ağacına 35 eklenmek istendiğinde, 35 değeri kök düğümün değeri 55'ten ve onun sol alt ağacının kök düğümünün değeri 45'ten küçüktür. Bu nedenle 45 düğümünün sol çocu-

ğu olarak eklenir. Denge faktörleri güncellenirler ve kök düğümün denge faktörünün 2 olduğu görülür. Sol alt ağacın yüksekliği sağ alt ağacın yüksekliğinden 2 fazladır. Bu durum AVL özelliğini bozar. Çözüm için sağ rotasyon kullanılır. 45

değerine sahip düğüm kök düğüm olur. Önceki kök düğüm yeni kök düğümün sağ çocuğu olarak sağa hareket ettirilir. Bu durum Şekil 5.17’de gösterilmektedir.



Şekil 5.17. 55, 45 ve 35 değerleri ile bir AVL ağacı oluşturulması



ARAŞTIRALIM

AVL ağaçlarından bir elemanın silinmesi işleminde oluşabilecek denge faktörü değişikliklerini ve çözüm yollarını araştırınız.

BÖLÜM ÖZETİ

Bu bölümde öncelikli olarak genel ağaç veri yapısı anlatılmıştır. Ağaçlar elemanlar arasında hiyerarşik bir ilişkiye sahip doğrusal olmayan veri yapılarıdır. Kök düğüm dışında ağaçta yer alan her bir düğümün bir ebeveyn düğümü vardır. Bir düğümün birden fazla ebeveyn düğüme sahip olması mümkün değildir. Bu yönü dolayısıyla ağaçlar, çizgelerden ayrılırlar. Bir düğümün en fazla iki çocuk ile sınırlandığı ağaçlara ikili ağaçlar denir. İkili ağaçlarda tüm düğümleri ziyaret etmek için üç farklı gezinme yöntemi vardır. Preorder gezinmede kök düğüm sol ve sağ alt ağaçtan önce, postorder gezinmede ise alt ağaçlardan sonra ziyaret edilir. Bir diğer yöntem inorder gezinmede önce sol alt ağaç ziyaret edilir daha sonra kök düğüm ve son olarak sağ alt ağaç ziyaret edilir. İkili arama ağacı, ağacı oluşturan düğümlerin değerleri arasında büyüklük-küçüklük ilişkisinin olduğu bir yapıdır. Kökte yer alan düğümün değeri tüm sol alt ağaçta yer alan değerlerden büyük, sağ alt ağaçta yer alan düğümlerin değerlerinden küçük veya eşittir. İkili arama ağaçları arama ve ekleme gibi veri yapısı işlemlerinin sıralı bir listede verimli bir şekilde gerçekleştirilmesi için oluşturulan bir yapıdır. Listedeki elemanların ikili arama ağacına eklenme sırası ağacın yapısını belirler. Bu nedenle bazı durumlarda sola dayalı ve sağa dayalı ağaç yapıları ortaya çıkabilir. Bu duruma çözüm için AVL ağaçları geliştirilmiştir. AVL ağaçları, ikili arama ağaçlarının özelliğine ek olarak denge faktörü kullanarak yükseklik dengeli bir yapı oluşturmaktadır.

GÖZDEN GEÇİRELİM

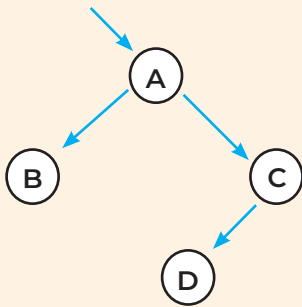
1. Aşağıdakilerden hangisi ağaç veri yapısı ile ilgili doğru bir ifadedir?

- a) Düğümlerin birden fazla ebeveyni olabilir.
- b) Yaprak düğümlerin bir çocuğu olabilir.
- c) Kök düğümün ebeveyni yoktur.
- d) Bir düğümde çocuk sayısı kadar ebeveyn vardır.
- e) Yaprak düğümlerin ebeveyni yoktur.

2. Ağaç veri yapısı ile ilgili aşağıda verilen ifadelerden hangisi yanlıştır?

- a) Ağaç veri yapısı doğrusal olmayan bir veri yapısıdır.
- b) Ağaçtaki tüm düğümlerin ebeveyni vardır.
- c) Bir düğümün iki ebeveyni olamaz.
- d) Bir düğümün birden fazla çocuğu olabilir.
- e) Yaprak düğümlerde çocukları gösteren işaretçiler NULL değerine sahiptir.

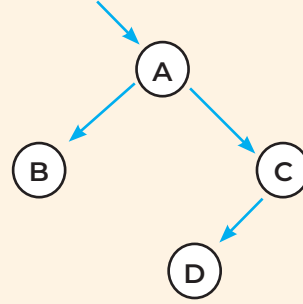
3.



Yukarıda verilen ikili ağaç yapısında aşağıdakilerden hangisi postorder gezinme ile elde edilir?

- a) B-D-C-A
- b) B-A-C-D
- c) D-C-B-A
- d) C-B-A-D
- e) D-B-C-A

4.



Yukarıda verilen ikili ağaç yapısında aşağıdakilerden hangisi inorder gezinme ile elde edilir?

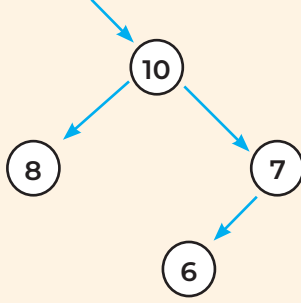
- a) D-C-B-A
- b) B-C-D-A
- c) B-C-A-D
- d) A-B-C-D
- e) B-A-D-C

5. Aşağıdaki gezinme yöntemlerinden hangisi ikili arama ağacında artan düzende sıralı bir liste üretir?

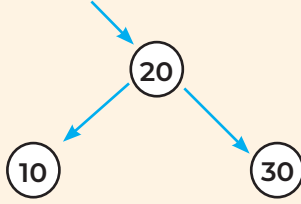
- a) levelorder gezinme
- b) postorder gezinme
- c) inorder gezinme
- d) preorder gezinme
- e) Sıralı liste oluşmaz

6. Aşağıdakilerden hangisi geçerli bir ikili arama ağacıdır?

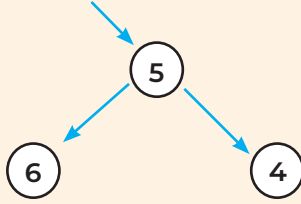
a)



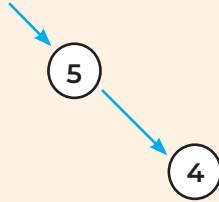
b)



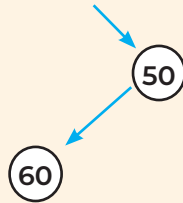
c)



d)



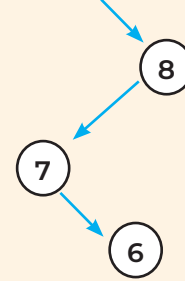
e)



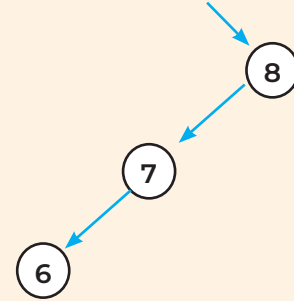
7. 8 – 7 – 6

Aşağıdaki verilen rakamlar verildiği sırada ikili bir arama ağacına yerleştirildiğinde aşağıdakilerden hangisi oluşur?

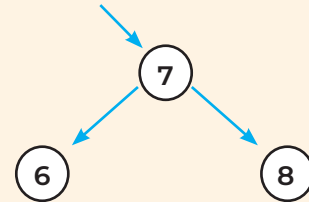
a)



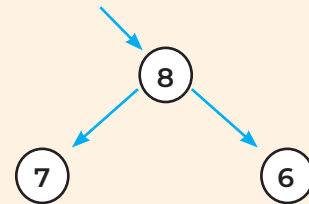
b)



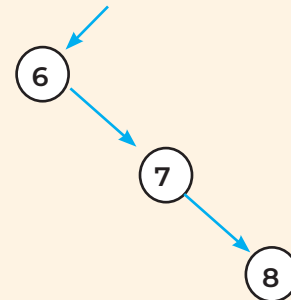
c)



d)



e)



8. Bir AVL ağacının ikili arama ağacından temel farkı aşağıdakilerden hangisidir?

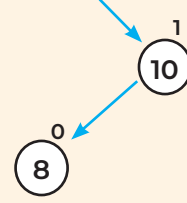
- a) Düğümlerdeki çocuk sayısı
- b) Düğümler arasındaki büyüklük ilişkisi
- c) Kırmızı düğümler
- d) Denge faktörü
- e) Siyah düğümler

9. Aşağıdakilerden hangisi AVL ağaçları için söylenemez?

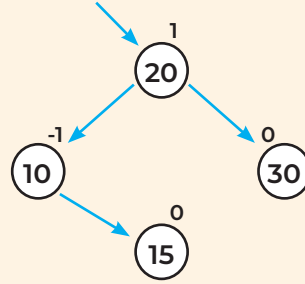
- a) Düğümlerde yer alan veriler arasında büyüklük küçüklük ilişkisi vardır
- b) 2 veya -2 denge faktörüne sahip düğümler geçerli bir AVL ağacında yer alamaz.
- c) Sağ ve sol alt ağaçlar arasındaki yükseklik farkı ikiden fazla olabilir.
- d) Bir düğümün en fazla iki çocuğu vardır.
- e) İkili arama ağaçlarının daha verimli bir formu olarak geliştirilmiştir.

10. Aşağıdakilerden hangisi geçerli bir AVL ağacı değildir?

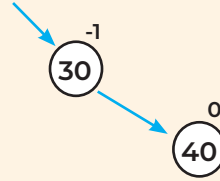
a)



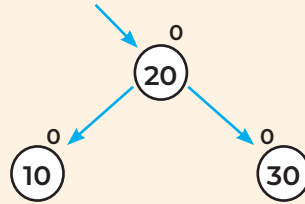
b)



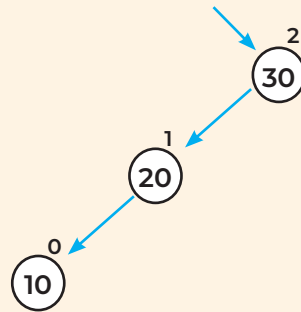
c)



d)



e)



Yanıt Anahtarı: 1-C 2-B 3-A 4-E 5-C 6-B 7-B 8-D 9-C 10-E

Kaynakça

Deitel, P., Deitel, H. (2017). C++ How to Program. 10th Edition. Pearson.

Thareja R. (2014) Data Structures Using C, 2nd Edition, Oxford University Press.

Levitin A. (2012) Introduction to The Design & Analysis of Algorithms, 3rd Edition, Pearson.

Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. (2009) Introduction to Algorithms, 3rd Edition, MIT Press.

