BÖLÜM 22. BİLİNEN PROBLEME İNDİRGEME

Bu algoritma tasarım yönteminde verilen karmaşık bir problemin çözümünü yapmada izlenecek yol ya da mantık tartışılacaktır.

Bu yöntemde, karmaşık olan problem çözümü yapılmadan önce problem bilinen problemlerden birine dönüştürülür ve ondan sonra bilinen problemin çözümü nasıl yapılıyorsa, bu problemin de çözümü benzer şekilde yapılır. Problemi bilinen bir probleme dönüştürme işlemi sofistike bir işlemdir, bundan dolayı çok karmaşık problemlerde her zaman başarılı olmak mümkün olmayabilir. Belki de problem alt problemlere bölündükten sonra her alt problemin bilinen probleme dönüşümü yapılacaktır.

Bu yöntem her zaman çözümü bilinmeyen problemlere uygulanır şeklinde bir kural yoktur. Çözümü yapılmış problemlerin algoritmalarının daha etkili hale getirilmesi için de bu yönteme başvurulabilir. Bu tasarım yönteminin daha iyi anlaşılması için birkaç örnek ile konu biraz daha detaylandırılabilir.

Örnek 22.1

Verilen bir dizi ya da liste içerisinde tekrar eden sayılar var mıdır? Tekrar varsa, tekrar eden sayıdan kaç tane vardır? Birbirinden farklı kaç tane tekrar eden sayı vardır ve her birinden kaç tane vardır?

Bu sorulara cevap vermenin farklı yolları olabilir. Bunlar içinde en etkili algoritma hangi yöntemle elde edilmişse, o çözüm en iyi çözüm olarak kabul edilir.

I. YOL

Birinci yol olarak bütün ikililer birbiri ile karşılaştırılırlar. Bu işlemi yapan algoritma Algoritma 22.1' de görülmektedir.

Algoritma 22.1. TekrarBul(A,B)

▷ A parametresi içinde tekrar eden sayıların olup olmadığının kontrol edileceği dizidir ve B parametresinin i. elemanı A dizisinin i. elemanından kaç tane olduğunu tutan bir dizidir.

```
i←1 ... n kadar
   B[i] ← -1
i←1 ... (n-1) kadar
   eğer B[i]=-1 ise
      B[i] ←1
      j←(i+1) ... n kadar
         eğer A[i]=A[j] ise
            B[i] ←B[i]+1
            B[j] ←0
```

Bütün sayılar ikili olarak karşılaştırılırlar ve bunun sonucunda yapılan karşılaştırma sayısı aşağıdaki gibi olur. Aşağıda yapılan karşılaştırma hesabı en kötü durum analizidir.

1. sayı için n-1 karşılaştırma yapılır. Çünkü ilk sayı ile geriye kalan n-1 tane sayının karşılaştırması yapılır. 2. sayı için ise geriye kalan n-2 tane sayı ile karşılaştırma yapılır ve böylece bu sayı için toplam n-2 tane karşılaştırma yapılır. Bu şekilde devam edilerek (n-1). sayı için 1 tane karşılaştırma yapılır ve son sayı için karşılaştırma yapılmaz. Bunun sonucunda elde edilen karşılaştırma toplamları

$$(n-1)+(n-2)+....+1=\frac{n(n-1)}{2}$$

olur. Bunun sonucunda bu algoritmanın mertebesi $O(n^2)$ olur. Bu mertebe bu algoritmanın en kötü durum analizidir ve en kötü durum A dizisi içindeki bütün sayıların farklı çıkması durumudur. Eğer A dizisi içindeki bütün sayılar aynı iseler, dış döngünün değişkeninin ilk değeri için iç döngü baştan sona kadar çalışır ve ondan sonraki

değerler için iç döngü hiç çalışmaz. Bunun sonucunda en iyi durum elde edilir ve en iyi durumun mertebesi Θ(n) olur.

II. YOL

Aynı problemin bilinen bir yöntemle çözülmesi daha iyi sonuç verebilir. Bunun anlamı problemi bilinen bir probleme dönüştürmektir ve bilinen problem sıralama işlemidir. A dizisi içindeki bütün sayılar sıralanır ve ondan sonra birinci elemandan başlanarak sona doğru ardışıl olan elemanlar karşılaştırılır. Bu şekilde kaç tane tekrar olduğu bulunur.

Algoritma 22.2. TekrarBul(A,B)

▷ A parametresi içinde tekrar eden sayıların olup olmadığının kontrol edileceği dizidir ve B parametresinin i. elemanı A dizisinin i. elemanından kaç tane olduğunu tutan bir dizidir.

```
YığınSırala(A)
j←1 ... n kadar
   B[j]←1
i←1
j←1 ... (n-1) kadar
  eğer A[j]=A[j+1] ise
      B[i]←B[i]+1
      B[j+1]←0
  değilse
      i←j+1
```

Bu algoritma iki kısımdan oluşmaktadır. Birinci kısmı A dizisinin sıralanması ve ikinci kısımda ise bir döngü ile tekrar sayısının bulunması işlemidir.

YığınSıralama algoritmasının mertebesinin $T_1(n)=\Theta(nlgn)$ olduğu daha önceden bilinmektedir ve ikinci kısımda ise bir tane döngü olduğundan, bu kısmın mertebesi $T_2(n)=\Theta(n)$ olur. Bunun sonucunda algoritmanın zaman bağıntısı T(n)

$T(n)=T_1(n)+T_2(n)$
$=\Theta(nlgn)+\Theta(n)$
$=\Theta(nlgn)$

sınıfına ait olur. Dikkat edilirse, ikinci yol ile elde edilen çözüm birinci yol ile elde edilen çözümden daha iyidir. Bilinen probleme indirgeme yapılarak elde edilen algoritma birinci algoritmaya göre daha etkili bir algoritmadır.

Örnek 22.2
İki boyutlu bir uzayda n tane noktadan hangi üç noktanın aynı doğru üzerinde olup olmadığı kontrolü yapılmak isteniyor. Bu problemin çözümü için en etkili algoritma nedir?

I. YOL
İlk olarak tasarlanacak olan algoritma klasik mantık olarak bütün nokta ikilileri arasındaki eğimler hesaplanır ve bu eğimler birbiri ile karşılaştırılarak hangi üç noktanın aynı doğru üzerinde olduğu belirlenir. Bu işlemi yapan algoritma Algoritma 22.3' te görüldüğü gibidir.

Algoritma 22.3. Doğru_Üz_Noktalar(A)
▷ A parametresi, her elemanı iki tane gerçel sayıdan oluşan bir iki boyutlu uzay noktaları kümesidir.

```
k←1 ... n kadar
   j←1 ... n kadar
      i←1 ... n kadar
         eğer k≠j≠i ise
            m1=eğim(A[k],A[j])
            m2=eğim(A[k],A[i])
            eğer m1=m2 ise
               (A[k],A[j]) ve (A[j],A[j]) noktaları aynı doğru üzerindedir.
```

Bu algoritmaya dikkat edilecek olursa, iç içe üç tane döngüden oluşmaktadır ve her döngü n kez çalışmaktadır. Birinci ve ikinci döngü birer kez çalıştıklarında üçüncü döngü n kez çalışmış olur. Birinci döngü bir kez, ikinci döngü n kez çalıştığında üçüncü döngü n^2 kez çalışmış olur. Saymanın temel prensibinden yola çıkarak üç tane işin n tane farklı yolla yapılabilmesi söz konusudur. Toplam yol sayısı bu algoritmanın zaman bağıntısını verir ve $T(n)=\Theta(n^3)$ olur.

Bu algoritmada aynı noktaların tekrar ele alınmaması için endekslerin başlangıç noktaları değiştirilerek algoritma tekrar yazılabilir ve algoritmanın yeni şekli Algoritma 22.4' te görülmektedir.

Algoritma 22.4. Doğru_Üz_Noktalar(A)

▷ A parametresi, her elemanı iki tane gerçel sayıdan oluşan bir iki boyutlu uzay noktaları kümesidir.

```
k←1 ... n kadar
  j←(k+1) ... n kadar
    i←(j+1) ... n kadar
      eğer k≠j≠i ise
        m1=eğim(A[k],A[j])
        m2=eğim(A[k],A[i])
        eğer m1=m2 ise
          (A[k],A[j]) ve (A[j],A[j]) noktaları aynı doğru üzerindedir.
```

Bu algoritmanın analizi yapılaması için aşağıdaki toplama işleminin yapılması gerekir (Her iterasyonda iki tane karşılaştırma yapıldığından dolayı toplam sembollerinin içerisi 2 olarak verilmiştir).

$$T(n)=\sum_{k=1}^{n}\sum_{j=k+1}^{n}\sum_{i=+1}^{n}2=\sum_{k=1}^{n}\sum_{j=1}^{n-k}\sum_{i=1}^{n-j}2=\sum_{k=1}^{n}\sum_{j=1}^{n-k}2(n-j)$$

$$=\sum_{k=1}^{n}\left(n^2-n-k^2+k\right)=n^3-n^2+\frac{n(n+1)}{2}-\frac{n(n+1)(2n+1)}{6}$$

$$=\frac{2n^3-3n^2+n}{3}$$

olur ve bunun sonucunda $T(n)=\Theta(n^3)$ olduğu görülmektedir. Yeni algoritmada meydana gelen karşılaştırma sayısındaki azalma algoritmanın mertebesinde sabit katsayı şeklinde olduğundan algoritmanın mertebesi değişmemektedir.

II.YOL

İkinci çözüm şeklinde ise, ilk önce oluşabilecek iki nokta arasındaki eğimlerin hepsi hesaplanır. Meydana gelebilecek eğim sayısı n tane noktanın 2' li kombinasyonu olur ve eğim sayısı M olmak üzere

$$M=\binom{n}{2}=\frac{n(n+1)}{2}$$

olur. Bundan sonraki işlem M tane eğimi sıralamaktır ve ondan sonra M tane elemanlı dizide tekrar eden elemanın olup olmadığı kontrol edilir. Bu işlemleri yapan algoritma Algoritma 22.5' te görülmektedir.

C dizisindeki elemanlar kendisi ile aynı endekse sahip B dizisinin o elemanından kaç tane olduğunu tutmaktadır.

Bu algoritmanın mertebesi hesaplanacak olursa, algoritmada üç parçadan oluşan bir zaman bağıntısı elde edilir. İlk parça eğimleri hesaplama zamanı ve bu zaman $T_1(n)$ olsun. İkinci parça B dizisini sıralama zamanı ve bu zaman $T_2(n)$ olsun. Son parçada ise sıralı B dizisi içinde tekrar eden eleman olup olmadığını kontrol etme zamanıdır ve bu zaman $T_3(n)$ olsun. Bu zamanlar

Algoritma 22.5. Doğru_Üz_Noktalar(A)

▷ A parametresi, her elemanı iki tane gerçel sayıdan oluşan bir iki boyutlu uzay noktaları kümesidir. Bu dizideki her eleman çifti arasındaki eğim hesaplanır ve bu eğim B dizisine atılır. Ondan sonra B dizisi sıralanır ve bu dizinin tekrar eden elemanı olup olmadığı kontrol edilir.

```
k←1 ... n kadar
   j←(k+1) ... n kadar
       m=eğim(A[k],A[j])
       B[i]=m
       i←i+1
YığınSırala(B,M)
j←1 ... r kadar
   C[j]←1
i←1
j←1 ... (M-1) kadar
   eğer B[j]=B[j+1] ise
       C[i]←C[i]+1
       C[j+1]←0
   değilse
       i←j+1
```

$T_1(n)=\Theta(n^2)$
$T_2(n)=\Theta(MlgM)=\Theta(n^2lgn)$
$T3(n)=\Theta(M)=\Theta(n^2)$

olur. Algoritmanın mertebesi $T(n)=T_1(n)+T_2(n)+T_3(n)=\Theta(n^2lgn)$ olur.

Örnek 22.3

Son olarak bir binanın güvenlik işlemleri kamera tertibatı ile yapılmak isteniyor ve kurulacak olan kamera sistemi, en az sayıda kamera

içerecek ve binada görüş alanı dışında da yer kalmayacak şekil olacaktır. Bu problem nasıl çözülür?

Çözüm

İlk olarak problemin bilinen bir probleme dönüştürülmesi gerekir. Binada kirişler ve kolonlar ayrıt olarak düşünüldüğünde, kiriş ve kolonların birleştiği noktalar da düğüm olarak düşünülebilir. Bu şekilde binanın çizgesi çıkarılmış olur. Binaya yerleştirilecek kameraların görmediği kiriş veya kolon kalmamalı. Kiriş ve kolonlar ayrıt olduklarına göre çözüm minimum-düğüm kapsama probleminin çözümü olur. Binayı modelleyen çizge G=(V,E) olmak üzere problemin çözümü Algoritma 22.6' daki algoritma ile yapılır.

Algoritma 22.6. Düğüm_Kapsama(G)

▷ C kümesi hangi köşelere kamera konulacaksa, o köşeleri temsil eden düğümleri içerir.

```
C←∅
E'←E
E'≠∅ olduğu sürece devam et
        (u,v)∈E' olan bir ayrıt seç ve
        C←C∪{u,v}
        E' kümesinde u veya v düğümüne çakışık olan ayrıtların hepsini sil.
```