BÖLÜM 2

ALGORİTMA TASARIM TEKNİKLERİ

ANAHTAR KAVRAMLAR
Algoritma, Tasarım Tekniği,
Kaba Kuvvet, Azalt & Fethet,
Böl & Fethet, Açgözlü Yaklaşım

ÖĞRENME HEDEFLERİ

1. Tasarım tekniğinin ne olduğunu ve algoritmik problem çözümünde ne ifade ettiğini bilir
2. Kaba kuvvet tasarım tekniğinin nasıl uygulanacağını bilir
3. Azalt & fethet tasarım tekniğinin hangi problem sınıfına nasıl uygulanacağını bilir
4. Böl & fethet tasarım tekniğinin hangi problem sınıfına nasıl uygulanacağını bilir
5. Açgözlü yaklaşım tasarım tekniğinin nasıl uygulanacağını bilir
6. Farklı tasarım tekniklerinden hangisinin verilen probleme daha etkin bir şekilde uygulanabileceğini bilir

İKİNCİ BÖLÜM

GİRİŞ

Algoritma tasarım tekniği, bilgisayar bilimlerindeki belirli bir takım problemlere uygulanan ve problem çözümünde izlenmesi gereken adımları belirleyen genel bir yaklaşım şeklinde tanımlanabilir. Bilgisayar bilimlerindeki problemlerin çözümü için geliştirilen algoritmaları, tasarım esnasında izlenen belirli bir takım ortak özellikler ve yaklaşımlar üzerinden tasarım tekniği başlığı altında gruplamak; benzer problemlerin çözümü için daha hızlı ve etkili bir biçimde yeni algoritmalar geliştirebilmemize imkan sağlarken, aynı zamanda algoritmaları üzerinde çalıştığımız problemler yerine algoritma tasarım süreciyle ilintili daha uygun bir kriter üzerinden sınıflandırmamıza da imkan sağlamaktadır. Bu bölümde kaba kuvvet, azalt & fethet, böl & fethet, açgözlü yaklaşım gibi bilgisayar bilimlerinde yaygın olarak kullanılan tasarım teknikleri incelenecektir.

1. KABA KUVVET (BRUTE FORCE)

Kaba kuvvet tasarım tekniği kullanılarak geliştirilen algoritmalar (kaba kuvvet algoritmaları) algoritmik problem çözümünde genellikle ilk akla gelen en toy ve en aşikar algoritmalardır denilebilir. Bu tasarım tekniğinde problem çözümü için üzerine kafa yorulmuş ve zekice tasarlanmış algoritmalar yerine, çoğunlukla problemin tanımından hareketle geliştirilmiş daha basit ve ilkel algoritmalar tercih edildiği için, kaba kuvvet algoritmalarının performansı diğer tasarım tekniklerine nazaran daha kötüdür. Bu nedenle kaba kuvvet etkili bir tasarım tekniği olarak değerlendirilmemektedir. Kaba kuvvet tasarım tekniği ile algoritmaların nasıl geliştirildiği örnekler üzerinden anlatılacaktır.

Verilen n tane tamsayıyı küçükten büyüğe sıralama problemini ele alalım. Bu problem için akla ilk gelen basit ve aşikar çözüm dizinin küçük elemanlarını bulup dizinin ön sıralarına taşımak olacaktır. Kabaca, önce dizinin en küçük elemanı bulunup dizinin ilk sırasına, sonrasında dizinin en küçük ikinci elemanı bulunup dizinin ikinci sırasına yerleştirilecektir. Bu şekilde dizinin 3., 4., ..., n. elemanları bulunup sırasıyla dizinin 3., 4., ..., n. sırasına yerleştirilecek ve nihayetinde küçükten büyüğe sıralı bir dizi elde edilecektir. Burada bahsi geçen algoritma sıralama problemi için geliştirilmiş kaba kuvvet algoritmalarından biri olup literatürde seçmeli sıralama (selection sort) ismiyle yer almaktadır. Seçmeli sıralama algoritmasının konuşma dilindeki gösterimi Şekil 2.1'de verilmiştir.

Girdi : N tane sıralanabilir elemandan oluşan bir dizi
Çıktı : Dizinin elemanlarının küçükten büyüğe sıralanışı

Adım 1: K isimli bir değişken tanımla ve 1 değerini K değişkenine ata

Adım 2: Dizinin en küçük K. elemanını bul ve dizinin K. sırasına yerleştir, ve K'yı bir artır

Adım 3: Eğer K > N ise, diziye dön; değilse, adım 2'ye geri dön

Şekil 2.1 Seçmeli Sıralama Algoritmasının Konuşma Dilinde Gösterimi

Aşağıda bir kaba kuvvet sıralama algoritması olan seçmeli sıralama algoritmasının verilen bir örneklem üzerinde nasıl çalıştığı ve sıralı diziyi nasıl bulduğu adım adım gösterilmiştir. Burada algoritma girdi olarak [7, 12, 5] dizisini almıştır.

Adım 1: K = 1
Adım 2: Dizinin en küçük elemanı 5'i finalde çıktı olarak dönülecek dizinin ilk sırasına yerleştir ve K = 2
Adım 3: K > N olmadığından Adım 2'ye dön
Adım 2: Dizinin en küçük ikinci elemanı 7'yi finalde çıktı olarak dönülecek dizinin ikinci sırasına yerleştir ve K = 3
Adım 3: K > N olmadığından Adım 2'ye dön
Adım 2: Dizinin en küçük üçüncü elemanı 12'yi finalde çıktı olarak dönülecek dizinin üçüncü sırasına yerleştir ve K = 4
Adım 3: K > N olduğundan yeni sıralamaya dön ([5, 7, 12])

Şimdi de dizgi eşleştirme (string matching) problemini ele alalım. Dizgi eşleştirme problemi, verilen bir metnin verilen bir dizgiyi içerip içermediğini belirleme problemi olarak tanımlanmaktadır. Bu problem için ilk akla gelen çözüm, metindeki dizgi ile aynı boyuttaki bütün kelimeleri belirlemek ve sonrasında dizginin bunlardan herhangi biri ile eşleşip eşleşmediğine bakmak

olacaktır. Tasarlanan bu kaba kuvvet algoritması; eğer dizgi kelimelerden biri ile eşleşiyorsa ilgili kelimenin ilk sembolünün metindeki indeksine dönecek, içermiyorsa sıfıra dönecektir. Kimi zaman, ana problemin bir parçası olan daha küçük problemler için algoritmalar geliştirip bunları alt yordam olarak kullanmak algoritma tasarımında esneklik sağlayacaktır. Bu problemde de önce verilen eşit uzunluklu iki dizginin eşleşip eşleşmediğini belirleyen bir algoritma tasarlanacak ve yukarıda açıklanan kaba kuvvet algoritmasında alt yordam olarak kullanılacaktır.

Girdi : N tane sembolden oluşan iki dizgi
Çıktı : Dizgiler aynıysa 1, değilse 0

Adım 1: K isimli bir değişken tanımla ve 1 değerini K değişkenine ata

Adım 2: İki dizgenin K. elemanı aynıysa, K'yı bir artır; değilse, 0'a dön

Adım 3: Eğer K > N ise, 1'e dön; değilse, adım 2'ye geri dön

Şekil 2.2 Eşleştirme Algoritmasının Konuşma Dilinde Gösterimi

Şekil 2.2'de verilen eşit uzunluklu iki dizginin eşleşip eşleşmediğini (aynı olup olmadığını) belirleyen algoritmanın konuşma dilindeki gösterimi, Şekil 2.3'de ise bu algoritmayı alt yordam olarak kullanan kaba kuvvet dizgi eşleştirme algoritmasının konuşma dilindeki gösterimi verilmiştir.

Girdi : N tane sembolden oluşan bir metin ve M tane sembolden oluşan bir dizgi
Çıktı : Metin dizgiyi içeriyorsa, dizgiyle eşleşen metindeki ilk kelimenin ilk harfinin metindeki indeksi; değilse, 0

Adım 1: Metindeki dizgi ile aynı boyutlu bütün kelimeleri belirle (toplamda N – M + 1 tane)

Adım 2: K isimli bir değişken tanımla ve 1 değerini K değişkenine ata

Adım 3: Adım 1'de belirlenen K. kelime ile dizgi üzerinde, Şekil 1.8'de verilen eşleştirme algoritmasını çalıştır; eşleştirme algoritmasının çıktısı 1 ise K değerine dön, değilse K'yı artır

Adım 4: Eğer K > N – M + 1 ise, 0'a dön; değilse, adım 3'e geri dön

Şekil 2.3 Dizgi Eşleştirme Algoritmasının Konuşma Dilinde Gösterimi

Aşağıda dizgi eşleştirme problemi için geliştirilmiş Şekil 2.3'deki kaba kuvvet algoritmasının verilen bir örneklem üzerinde nasıl çalıştığı adım adım gösterilmiştir. Burada algoritma girdi olarak 'ATA_BAK' metni ile 'BAK' dizgisini almıştır.

Adım 1: ATA, TA_, A_B, _BA, BAK
Adım 2: K = 1
Adım 3: ATA ile BAK kelimesi için eşleştirme algoritması 0 değerini döneceğinden K = 2
Adım 4: K > N – M + 1 olmadığından Adım 3'e dön (burada N – M + 1 = 7 – 3 + 1 = 5)
Adım 3: TA_ ile BAK kelimeleri için eşleştirme algoritması 0 değerini döneceğinden K = 3
Adım 4: K > N – M + 1 olmadığından Adım 3'e dön
Adım 3: A_B ile BAK kelimeleri için eşleştirme algoritması 0 değerini döneceğinden K = 4

Adım 4: K > N – M + 1 olmadığından Adım 3'e dön
Adım 3: _BA ile BAK kelimeleri için eşleştirme algoritması 0 değerini döneceğinden K = 5
Adım 4: K > N – M + 1 olmadığından Adım 3'e dön
Adım 3: BAK ile BAK kelimeleri için eşleştirme algoritması 1 değerini döneceğinden algoritma K = 5 değerine dönecektir

Son olarak da sırt çantası (knapsack) problemini ele alalım. Sırt çantası problemi; M kapasitesine sahip bir çanta ve ağırlıkları ve değerleri belli çantaya koyulabilecek N eşya verildiğinde, çantayı ağırlıkları toplamı kapasiteyi aşmayacak şekilde değerleri toplamı maksimum olan eşyalarla doldurma problemi olarak tanımlanmaktadır. Bu gibi problemler için kaba kuvvet algoritmalarında genellikle yapılan bütün olası durumları belirlemek ve bu olası durumları tek tek inceleyerek hangisinin verilen kısıtı sağlayan optimum çözüm olduğunu bulmak olmaktadır. Bu problem için de geliştireceğimiz kaba kuvvet algoritması önce verilen eşyalar üzerinden olası tüm eşya kümelerini belirleyecek ve sonrasında sırasıyla bu kümeleri inceleyerek hangi kümenin ağırlık kısıtını sağlayan değerleri toplamı en yüksek eşya kümesi olduğunu bulacaktır. Sırt çantası problemi için tasarladığımız kaba kuvvet algoritmasının konuşma dilindeki gösterimi Şekil 2.4'de verilmiştir.

Girdi : M kapasiteli bir sırt çantası ve sırt çantasına konulabilecek ağırlığı ve değeri verilmiş N tane eşya
Çıktı : Değerleri toplamı maksimum olan ağırlıkları toplamı M'yi aşmayan eşya kümesi

Adım 1: Olası bütün eşya kümelerini belirle (toplamda 2^N tane)

Adım 2: K, A ve B isimli üç değişken tanımla ve 1 değerini K değişkenine 0 değerini A ve B değişkenlerine ata

Adım 3: Adım 1'de belirlenen K. eşya kümesi için; kümedeki eşyaların ağırlıkları toplamı M'yi aşmıyorsa ve kümedeki eşyaların değerleri toplamı A'dan büyükse, eşyaların değerleri toplamını A'ya ve K'yı B'ye ata. K'yı 1 artır

Adım 4: Eğer $K > 2^N - 1$ ise, adım 1'de belirlenen B. eşya kümesine ve A değerine dön; değilse, adım 3'e geri dön

Şekil 2.4 Sırt Çantası Algoritmasının Konuşma Dilinde Gösterimi

Aşağıda sırt çantası problemi için geliştirilmiş Şekil 2.4'deki kaba kuvvet algoritmasının verilen bir örneklem üzerinde nasıl çalıştığı adım adım gösterilmiştir. Burada algoritma girdi olarak 50 ağırlık kapasiteli bir sırt çantası ve ağırlıkları sırasıyla (10, 20, 30) ve değerleri sırasıyla (60, 100, 120) olan 3 eşya almıştır.

Adım 1: {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}
Adım 2: K = 1, A = 0, B = 0
Adım 3: Birinci kümedeki eşyaların ağırlıkları toplamı 10, 50'yi aşmadığından ve eşyaların değerleri toplamı 60, 0'dan büyük olduğundan A = 60, B = 1. K = 2
Adım 4: $K > 2^N - 1$ olmadığından Adım 3'e dön (burada $2^N - 1 = 7$)
Adım 3: İkinci kümedeki eşyaların ağırlıkları toplamı 20, 50'yi aşmadığından ve eşyaların değerleri toplamı 100, 60'dan büyük olduğundan A = 100, B = 2. K = 3
Adım 4: $K > 2^N - 1$ olmadığından Adım 3'e dön
Adım 3: Üçüncü kümedeki eşyaların ağırlıkları toplamı 30, 50'yi aşmadığından ve eşyaların değerleri toplamı 120, 100'den büyük olduğundan A = 120, B = 3. K = 4
Adım 4: $K > 2^N - 1$ olmadığından Adım 3'e dön
Adım 3: Dördüncü kümedeki eşyaların ağırlıkları toplamı 30, 50'yi aşmadığından ve eşyaların değerleri toplamı 160, 120'den büyük olduğundan A = 160, B = 4. K = 5

Adım 4: K > 2^N – 1 olmadığından Adım 3'e dön
Adım 3: Beşinci kümedeki eşyaların ağırlıkları toplamı 40, 50'yi aşmadığından ve eşyaların değerleri toplamı 180, 160'dan büyük olduğundan A = 180, B = 5. K = 6
Adım 4: K > 2^N – 1 olmadığından Adım 3'e dön
Adım 3: Altıncı kümedeki eşyaların ağırlıkları toplamı 50, 50'yi aşmadığından ve eşyaların değerleri toplamı 220, 180'den büyük olduğundan A = 220, B = 6. K = 7
Adım 4: K > 2^N – 1 olmadığından Adım 3'e dön
Adım 3: Yedinci kümedeki eşyaların ağırlıkları toplamı 60, 50'yi aştığından K = 8
Adım 4: K > 2^N – 1 olduğundan algoritma {2, 3} eşya kümesine ve A = 220 değerine dönecektir

Yukarıda da belirttiğimiz gibi performansının diğer alternatiflerine nazaran çok daha kötü olması sebebiyle kaba kuvvet etkili bir tasarım tekniği olarak değerlendirilmemektedir. Bununla birlikte, diğer alternatiflerinin aksine kaba kuvvet çok daha geniş yelpazedeki problemlere uygulanabilmektedir. Ayrıca; performansın çok da önemli olmadığı algoritmanın yalnızca küçük boyutlu ya da sınırlı sayıdaki girdilere uygulandığı durumlarda, kaba kuvvet tasarım tekniğini kullanmak tasarımcıyı daha etkili algoritmalar geliştirme zahmetinden kurtaracaktır.

2. AZALT & FETHET (DECREASE & CONQUER)

Eğer verilen bir problemin çözümü ile problemin daha küçük boyutlu alt probleminin çözümü arasında bir ilişki varsa, ve alt problemin çözümü bu ilişki üzerinden genişletilerek ana problemin çözümü elde edilebiliyorsa; akıllıca olan doğrudan ana problemin çözümü için uğraşmaktansa, görece daha kolay olan alt problemi çözmek ve sonrasında bu çözümü kullanarak ana problemin çözümünü elde etmek olacaktır. Azalt & fethet tasarım tekniğinde yapılan; verilen problem eğer yukarıda ifade edilen özelliği sağlıyorsa, ana problemi çözmek yerine girdi boyutunu küçülterek bir alt problem elde etmek ve görece daha kolay bu alt problem için çözüm üretmektir. Sonrasında elde edilen çözüm gözlemlenen ilişki üzerinden ana problemin çözümüne genişletilecektir.

Verilen iki sayının en büyük ortak bölenini bulma problemini ele alalım. Verilen iki M, N sayıları için; büyük sayının N olduğu kabul edilirse, OBEB(N, M) = OBEB(M, N mod M). Bu eşitlik bize ana problemin çözümü (N ve M sayılarının en büyük ortak böleni) ile daha küçük boyutlu alt problemin çözümünün (M ve N mod M sayılarının en büyük ortak böleni) aynı olduğunu, yani ana problemin çözümü ile daha küçük boyutlu alt problemin çözümü arasında doğrudan ilişki olduğunu söylemektedir. Dolayısıyla bu ilişkiden istifade ederek verilen probleme azalt & fethet tasarım yöntemini uygularsak Şekil 2.5'de sözde kod ile gösterimi verilen algoritmayı elde ederiz. Dikkat edilirse bu özyinemeli algoritma ile her adımda girdi boyutu belli oranda azaltılarak algoritmanın daha hızlı ve etkili bir biçimde çıktı üretmesi sağlanmıştır. Bu algoritma literatürde Öklid algoritması olarak yer almaktadır.

```
1   Öklid(N, M)
2
3   girdi : N ve M sayıları
4   çıktı : EBOB(N, M)
5
6   // sayılardan N'nin büyük olduğu varsayılmaktadır
7   if M = 0
8       return N
9   else
10      return Öklid(M, N mod M)
```

Şekil 2.5 Öklid Algoritmasının Sözde Kodu

Öklid algoritmasının verilen girdi üzerinde nasıl çalıştığını bir örneklem üzerinden anlamaya çalışalım. Örneklemde kullanılacak sayılar 24 ve 18 olsun. İkinci sayı 0'dan farklı olduğundan algoritma EBOB(24, 18)'i bulmak yerine Öklid (18, 6) kodu ile daha küçük boyutlu alt problem olan EBOB(18, 6)'yı hesap edecektir. Sonraki adımda yine ikinci sayı 0'dan farklı olduğundan algoritma EBOB(18, 6)'yı bulmak yerine Öklid (6, 0) kodu ile daha küçük boyutlu alt problemin çözümünü bulmaya çalışacaktır. Öklid(6, 0) için, ikinci sayı sıfır olduğundan algoritma ilk sayıya dönecektir. Yani algoritma EBOB(24, 18)'i 6 olarak bulacaktır.

Girdi : Bir birine tıpatıp benzeyen ve bir tanesi hafif olmak üzere geri kalanları aynı ağırlıkta olan N top ve eşit kollu bir terazi
Çıktı : Hafif top

Adım 1: N tekse, bir topu kenara ayırıp kalan topları iki eşit paya ayırıp bu payları terazinin kefelerine yerleştir; değilse topları iki eşit paya ayırıp bu payları terazinin kefelerine yerleştir

Adım 2: Eğer paylar eşit ağırlıktaysa, kenarda bırakılan topu çıktı olarak belirle

Adım 3: Paylar eşit ağırlıkta değilse ve hafif olan kefede tek bir top varsa, bu topu çıktı olarak belirle

Adım 4: Paylar eşit ağırlıkta değilse ve hafif olan kefede birden fazla top varsa, bu topları girdi kabul et ve adım 1'e dön

Şekil 2.6 Hafif Top Algoritmasının Konuşma Dilinde Gösterimi

Şimdi de hafif topu bulma problemini ele alalım. Hafif topu bulma problemi, birbirine tıpa tıp benzeyen ve bir tanesi hafif olmak üzere geri kalanları aynı ağırlıkta olan N top ve eşit kollu bir terazi verildiğinde, hafif topu minimum tartma işlemi ile bulma problemi olarak tanımlanmaktadır. N sayısının çift olduğunu varsayalım. Topların yarısını terazinin sağ kefesine, diğer yarısını da sol kefesine koyarsak, ve sol kefedeki toplar daha hafif gelirse, aradığımız hafif top sol kefedekilerden biridir diyebiliriz. Burada dikkat ederseniz, başlangıçta verilmiş olan N tane topun hafif olanıyla, sol kefedeki N/2 tane topun hafif olanı aynı olacaktır. Diğer bir ifadeyle, N boyutlu ana problemin çözümü ile sol kefedeki N/2 boyutlu alt problemin çözümü arasında doğrudan bir ilişki olacaktır. Dolayısıyla bu ilişkiden istifade ederek verilen probleme azalt & fethet tasarım yöntemini uygularsak Şekil 2.6'da konuşma dili ile gösterimi verilen algoritmayı elde ederiz. Dikkat edilirse burada da yukarıdaki algoritmaya benzer biçimde özyinemeli olarak her adımda girdi boyutu belli oranda azaltılıp algoritmanın daha hızlı ve etkili bir biçimde çıktı üretmesi sağlanmıştır.

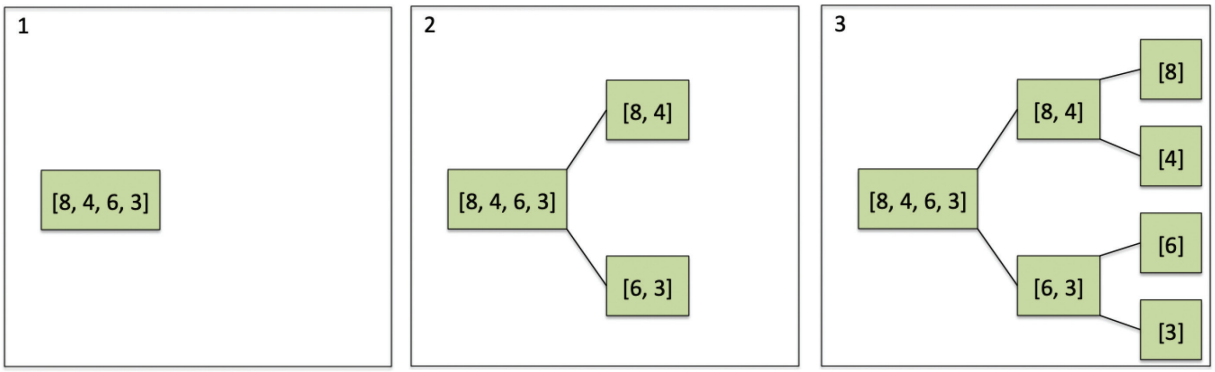
3. BÖL & FETHET (DIVIDE & CONQUER)

Tasarım teknikleri içerisinde en bilineni ve yaygın olarak kullanılanı böl & fethet tasarım tekniğidir. Azalt & fethet tekniğinde olduğu gibi, eğer verilen bir problemin çözümü ile problemin daha küçük boyutlu alt problemlerinin çözümleri arasında bir ilişki varsa, ve alt problemlerin çözümü birleştirilerek ana problemin çözümü elde edilebiliyorsa; böl & fethet tasarım tekniğinde de yapılan ana problemi doğrudan çözmek yerine, problemi alt problemlere ayırıp özyinelemeli bir şekilde alt problemleri çözmektir. Sonrasında bu çözümler belirlenen ilişki üzerinden birleştirip ana problemin çözümü elde edilmektedir.

```
1   EnKucuk([t_i, t_i+1, ..., t_j])
2
3   girdi : N elemanlı bir tam sayı dizisi
4   çıktı : Dizinin en küçük elemanı
5
6   // dizide tek bir eleman kaldıysa algoritma o elemana döner
7   if i = j
8       return t_i
9   else
10      mid ← (i + j)/2
11      // dizinin ilk yarısının en küçük elemanı
12      kucuk1 ← EnKucuk([t_i, ..., t_mid])
13      // dizinin ikinci yarısının en küçük elemanı
14      kucuk2 ← EnKucuk([t_mid+1, ..., t_j])
15  if kucuk1 < kucuk2
16      return kucuk1
17  else
18      return kucuk2
```

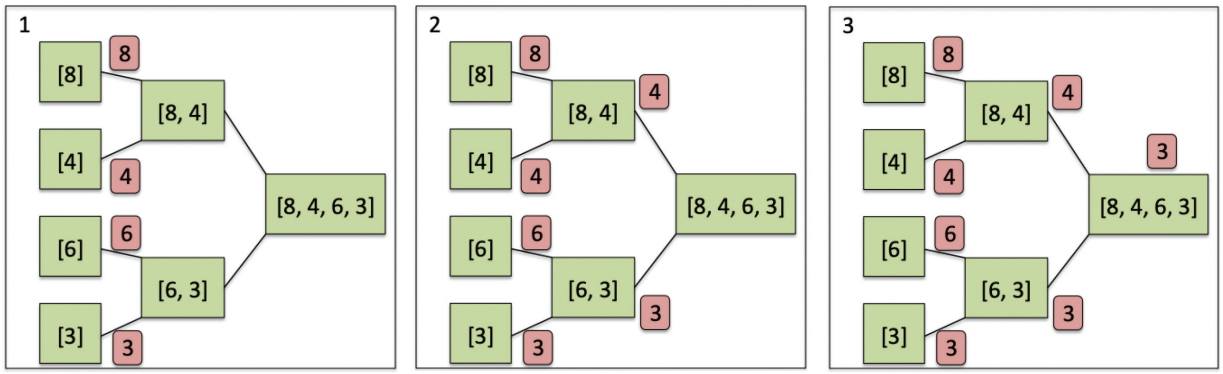
Şekil 2.7 En Küçük Elemanı Bulma Algoritmasının Sözde Kodu

Böl & fethet nasıl uygulandığını, verilen N elemanlı bir sayı dizisinin en küçük elemanını bulma problemi üzerinden anlamaya çalışalım. Burada problemi doğrudan çözmek yerine; problemi, sırasıyla dizinin ilk N/2 elemanının en küçük elemanını bulma ve son N/2 elemanının en küçük elemanını bulma problemleri şeklinde iki alt probleme bölelim ve bu alt problemlerin çözümlerini bulalım. Dikkat edilirse bu alt problemlerin çözümlerinden küçük olanı, yani dizinin sırasıyla ilk yarısının en küçük elemanıyla ikinci yarısının en küçük elemanından küçük olanı ana problemin çözümü olacaktır. Burada geliştireceğimiz algoritma alt problemleri özyinelemeli bir şekilde çözecektir. Yani, ana problemde uyguladığımız böl ve fethet stratejisi alt problemlerin çözümünde de kullanılacaktır. Problemin çözümü için tasarladığımız böl & fethet algoritmasının sözde kod gösterimi Şekil 2.7'de verilmiştir.



Şekil 2.8 Problemin Alt Problemlere Ayrılması

Böl & fethet tasarım tekniğini daha iyi anlayabilmek adına, Şekil 2.7'de verilen algoritmanın [8, 4, 6, 3] örneklemi üzerinde adım adım nasıl çalıştığını inceleyelim. Algoritma Şekil 2.8'de gösterildiği gibi problemi alt problemlere, verilen alt problemleri de daha küçük boyutlu alt problemlere ayıracaktır. Bu süreç özyinelemeli olarak alt problemler tek bir eleman içerene kadar devam edecektir. Sonrasında ise, Şekil 2.9'da gösterildiği gibi alt problemlerin çözümleri birleştirilerek daha büyük alt problemlerin çözümleri elde edilecektir. Yine benzer şekilde bu süreç de özyinelemeli olarak ana problemin çözümü elde edilinceye kadar devam edecektir.



Şekil 2.9 Alt Problemlerin Çözümlerinin Birleştirilmesi

Şimdi de büyük sayıların çarpımı problemini ele alalım. Günümüzde bazı uygulamalar, özellikle bir takım modern kriptografik teknikler 300 basamaklı gibi çok büyük sayıların çarpımını gerektirmektedir. Bu boyuttaki sayıların pratik bir şekilde çarpılabilmesi modern kriptografideki süreçleri hızlandıracaktır. Böl & fethet tasarım tekniğinin bu probleme nasıl uygulanabileceğini, 6 basamaklı 123456 ve 654321 sayılarının çarpımı gibi basit bir örnek üzerinden anlamaya çalışalım. Burada doğrudan çarpmak yerine, daha etkili bir yöntem elde etmek adına sayıları 123456 = 123000 + 456 ve 654321 = 654000 + 321 şeklinde iki parçaya ayrılmış olarak ele alalım. Verilen sayıların çarpımını bu formlar üzerinden yeniden oluşturursak: $123456 \times 654321 = 123 \times 654 \times 10^6 + (123 \times 321 + 456 \times 654) \times 10^3 + 456 \times 321$ denklemini elde etmiş oluruz. Görüleceği üzere, burada 6 basamaklı iki sayının çarpımı problemi 4 adet 3 basamaklı iki sayının çarpımı alt problemine indirgenmiştir. Genelleyecek olursak, N basamaklı A ve B sayıları verildiğinde, sayılar önce $A = A_1 \times 10^{N/2} + A_2$ ve $B = B_1 \times 10^{N/2} + B_2$ şeklinde N/2 basamaktan oluşan iki parçaya ayrılmış olarak ele alınacak ve bu parçaların çarpımları $A_1 \times B_1$, $A_1 \times B_2$, $A_2 \times B_1$ ve $A_2 \times B_2$ hesaplanacaktır. Sonra bu çözümler $A \times B = A_1 \times B_1 \times 10^N + (A_1 \times B_2 + A_2 \times B_1) \times 10^{N/2} + A_2 \times B_2$ denklemi üzerinden birleştirilecek ve ana problemin çözümü bulunmuş olacaktır. Şekil 2.10'da büyük sayıların çarpımı problemi için geliştirilmiş böl & fethet algoritmasının sözde kod ile gösterimi verilmiştir.

DİKKAT EDELİM

Azalt & fethet ve böl & fethet tasarım tekniklerinin probleme uygulanabilmesi için problemin alt probleme indirgenebilmesi veya alt problemlere ayrılabiliyor olması ve alt problem veya problemlerin çözümüyle ana problemin çözümü arasında ilişki kurulabilmesi gerekmektedir.

```
1   Carpim(A, B, N)
2
3   girdi : N basamaklı iki sayı
4   çıktı : Sayıların çarpımı
5
6   If n = 1
7       return AxB
8   // sayılar iki parçaya ayrılmış olarak ele alınır
9   A ← A1x10^N/2 + A2
10  B ← B1x10^N/2 + B2
11  //verilen parçalar üzerinden oluşturulan alt problemler
12  //hesaplanır
13  X ← Carpim(A1, B1, N/2)
14  Y ← Carpim(A1, B2, N/2)
15  Z ← Carpim(A2, B1, N/2)
16  T ← Carpim(A2, B2, N/2)
17  // alt problemlerin çözümleri birleştirilir
18  return Xx10^N + (Y + Z) x10^N/2 + T
```

Şekil 2.10 Böl & Fethet Çarpım Algoritmasının Sözde Kodu

4. AÇGÖZLÜ YAKLAŞIM (GREEDY APPROACH)

Verilen bir problemi açgözlü yaklaşımla çözerken; çözümün her adımında, o andaki mevcut ve problemin istediği şartları sağlayan lokal çözümlerden en uygun olanı tercih edilip ana problemin çözümüne eklenmektedir. Böylelikle adım adım lokal çözümler üzerinden ana problemin çözümü oluşturulmaktadır. Bu tasarım tekniğinde dikkat edilmesi gereken nokta, çözümün her adımında tasarım tekniğinin adından da anlaşılacağı üzere aç gözlü bir şekilde hareket edildiği ve lokaldeki çözümün bizi nihayetinde ana problemin istediği optimum çözüme ulaştırıp ulaştırmayacağının bilinmediği gerçeğidir. Bu tasarım tekniği genellikle optimizasyon problemleri için tercih edilmektedir.

```
1   Kasiyer([t1, t2, ..., tN], M)
2
3   girdi : M kuruş para üstü ve bozuk para çeşitleri [t1, t2, ..., tN]
4   çıktı : Minimum sayıda bozuk para
5
6   // Bozuk para çeşitleri t1 > t2 > ... > tN şeklinde sıralıdır
7   // A para üstü K ise toplam bozuk para miktarı için
8   // oluşturulmuş değişkenlerdir
9   A ← M
10  j ← 1
11  K ← 0
12  while j ≤ N
13      if tj ≤ A
14          A ← A - tK
15          K ← K + 1
16      else
17          j ← j + 1
18  return K
```

Şekil 2.11 Kasiyer Algoritmasının Sözde Kodu

Bu tasarım tekniğinde nasıl algoritma geliştirildiğini kasiyer problemi ile anlamaya çalışalım. Kasiyer problemi, para üstü için gerekli olan minimum sayıda bozuk parayı bulma problemi olarak tanımlanmaktadır. Burada kullanılabilecek bozuk para çeşitleri 25, 10, 5 ve 1 kuruş

olarak belirlenmiş olsun. Örneğin 62 kuruş para üstü için, bunu 6 tane 10 kuruş ve 2 tane 1 kuruş kullanarak verebileceğimiz gibi, 1 tane 25 kuruş, 3 tane 10 kuruş ve 7 tane 1 kuruş kullanarak da verebiliriz. İlk çözümde toplamda 8 tane bozuk para kullanılırken, ikinci çözümde 11 tane bozuk para kullanılmıştır. Peki optimum çözüm hangi bozuk paralardan oluşuyor olacaktır? Burada açgözlü yaklaşım tekniğini takip ettiğimizde yapacağımız ilk şey mümkün olan en büyük bozuk para ile çözümü oluşturmaya başlamak olacaktır. Dolayısıyla 62 kuruş para üstü için ilk kullanacağımız bir adet 25 kuruş olacaktır. Kalan 37 kuruş için aynı strateji izlersek yine bir 25 kuruş daha kullanmamız gerekecek. Kalan 12 kuruş için, kullanılabilecek en yüksek bozuk para çeşidi 10 kuruş olduğundan çözüme bir 10 kuruş eklememiz gerekecek. Son olarak kalan 2 kuruş için de 2 adet 1 kuruş kullanmamız gerekecek. Dolayısıyla finalde elde edeceğimiz optimum çözüm 2 adet 25 kuruş, 1 adet 10 kuruş ve 2 adet 1 kuruş , yani toplamda 5 adet bozuk para içerecektir. Burada anlattığımız açgözlü yaklaşım tekniği ile geliştirilmiş kasiyer algoritmasının sözde kod ile gösterimi Şekil 2.11'de verilmiştir.

BÖLÜM ÖZETİ

Bu bölümde kaba kuvvet, azalt&fethet, böl&fethet, açgözlü yaklaşım gibi bilgisayar bilimlerinde yaygın olarak kullanılan tasarım teknikleri anlatılmıştır. Algoritma tasarım tekniği, bilgisayar bilimlerindeki belirli birtakım problemlere uygulanan ve problem çözümünde izlenmesi gereken adımları belirleyen genel bir yaklaşım şeklinde tanımlanabilir. Kaba kuvvet tasarım tekniğinde problem çözümü için üzerine kafa yorulmuş ve zekice tasarlanmış algoritmalar yerine, çoğunlukla problemin tanımından hareketle geliştirilmiş daha basit ve ilkel algoritmalar tercih edilmektedir. Bu yüzden kaba kuvvet algoritmalarının performansı diğer tasarım tekniklerine nazaran daha kötüdür. Bölüm içerisinde bu tasarım tekniğinde nasıl algoritma geliştirildiği farklı problem sınıflarından seçilmiş örneklerle detaylıca incelenmiştir. Azalt & fethet ve böl & fethet tasarım tekniklerinin probleme uygulanabilmesi için problemin alt probleme indirgenebilmesi veya alt problemlere ayrılabiliyor olması ve alt problem veya problemlerin çözümüyle ana problemin çözümü arasında ilişki kurulabilmesi gerekmektedir. Azalt & fethet tasarım tekniğinde ana problem için çözüm bulmak yerine problemin alt problemi için çözüm geliştirilmekte ve sonrasında bu çözüm ana problemin çözümüne genişletilmektedir. Benzer şekilde böl & fethet tasarım tekniğinde de ana problem alt problemlere ayrılmakta ve özyinelemeli bir biçimde alt problemlerin çözümleri birleştirilerek ana problemin çözümü elde edilmektedir. Kaba kuvvet tekniğinde olduğu gibi bu tasarım teknikleri de örnekler üzerinden detaylı bir şekilde ele alınmıştır. Bölümde son olarak açgözlü yaklaşım tekniğinde algoritma geliştirme sürecinin nasıl yürütüldüğü bir örnek üzerinden açıklanmıştır.

GÖZDEN GEÇİRELİM

1. **N elemanlı bir tamsayı dizisi ve bir X sayısı verildiğinde, kaba kuvvet tasarım tekniğini kullanarak dizinin toplamları X'e eşit olan iki sayı içerip içermediğini bulan bir algoritma geliştiriniz ve algoritmanın konuşma dilindeki gösterimini yazınız.**

2. **Verilen bir X tamsayısı ve N pozitif tamsayısı için, kaba kuvvet tasarım tekniğini kullarak "X'in N. kuvvetini hesaplayan bir algoritma geliştiriniz ve algoritmanın sözde kod ile gösterimini yazınız.**

3. **Verilen N tane harften oluşan bir kelime için, kaba kuvvet tasarım tekniğini kullanarak bu kelimenin A harfi ile başlayan anlamlı ya da anlamsız kaç tane alt kelime içerdiğini bulan bir algoritma geliştiriniz ve algoritmanın konuşma dilindeki gösterimini yazınız.**

4. **Aşağıda sözde kod ile gösterimi verilen ve verilen bir sıralı tamsayı dizisinin verilen bir X sayısını içerip içermediğini belirleyen Arama algoritması hangi tasarım tekniği ile geliştirilmiştir belirleyiniz?**

Arama (T, i, j, X)

Girdi : i başlangıç indeksi ve j bitiş indeksi olmak üzere j - i + 1 elemandan oluşan T tamsayı dizisi ve X tamsayısı
Çıktı : Dizi X'i içeriyorsa, dizinin X'e eşit ilk elemanının indeksi; değilse, 0

```
//başlangıç indeksi bitiş indeksinden büyükse, dizi X'i içermiyordur
if i > j
    return 0
//dizinin ortası belirlenir
mid = (i + j)/2
if X = t_mid
    return mid
//dizi sıralı olduğu için, X dizinin ortasındaki elemandan küçükse, X dizinin sol yarısında aranır
elseif X < t_mid
    Arama(T, i ,mid-1, X)
//X dizinin ortasındaki elemandan büyükse, X dizinin sağ yarısında aranır
else
    Arama(T, mid+1, j, X)
```

5. **Verilen bir problemin çözümü için algoritma geliştirilirken böl & fethet tasarım tekniğinin nasıl uygulandığını belirtiniz.**

6. **Böl &fethet tasarım tekniğini kullanarak verilen N elemanlı bir tamsayı dizisinin en büyük elemanının bulan bir algoritma geliştiriniz ve algoritmanın sözde kod ile gösterimini yazınız.**

7. **Aşağıda sözde kod ile gösterimi verilen, para üstü ve kullanılabilecek bozuk para çeşitleri verildiğinde, para üstü için gerekli olan minimum sayıda bozuk parayı bulan Kasiyer algoritmasının doğruluğunu ([10, 6, 1], 24) örneklemi üzerinden irdeleyiniz.**

Kasiyer($[t_1, t_2, ..., t_N]$, M)

Girdi : M kuruş para üstü ve bozuk para çeşitleri $[t_1, t_2, ..., t_N]$
Çıktı : Minimum sayıda bozuk para

// Bozuk para çeşitleri $t_1 > t_2 > ... > t_N$ şeklinde sıralıdır
// A para üstü K ise toplam bozuk para miktarı için oluşturulmuş değişkenlerdir

```
A ← M
j ← 1
K ← 0
while j =< N
    if t_j ≤ A
        A ← A - t_K
        K ← K + 1
    else
        j ← j + 1
return K
```

8 – **Verilen bir X tamsayısı ve N pozitif tamsayısı için, azalt & fethet tasarım tekniğini kullarak X'in N. kuvvetini hesaplayan bir algoritma geliştiriniz ve algoritmanın sözde kod ile gösterimini yazınız.**

9. **Konuşma dilindeki gösterimi aşağıdaki şekilde olan ve verilen bir T tamsayı dizisinin verilen bir X elemanını içerip içermediğini belirleyen kaba kuvvet tasarım tekniği ile geliştirilmiş algoritmanın (T = [6, 7, 8, 9], X = 9) örneklemi üzerinde nasıl sonuç ürettiğini adım adım gösteriniz.**

Girdi : N elemanlı bir T tamsayı dizisi ve X tamsayısı
Çıktı : Dizi X'i içeriyorsa, dizinin X'e eşit ilk elemanının indeksi; değilse, 0 değeri

Adım 1: K isimli bir değişken tanımla ve 1 değerini K değişkenine ata
Adım 2: Dizinin K. elemanı X'e eşitse K değerine dön; değilse, K'yı bir artır
Adım 3: K > N ise, algoritmanın çıktısı olarak 0 değerine dön; değilse, adım 2'ye geri dön

10. **Aşağıda sözde kod ile gösterimi verilen ve verilen iki büyük sayının çarpımını bulan böl & yönet tasarım tekniğinde geliştirilmiş Carpim algoritmasındaki alt problemlerin sayısı görüleceği üzere 4'tür. Algoritmayı daha az alt problem içerecek şekilde geliştirerek daha iyi performanslı bir algoritma elde ediniz.**

Carpim(A, B, N)

Girdi : N basamaklı iki sayı

Çıktı : Sayıların çarpımı

```
If n = 1
    return AxB
```

// sayılar iki parçaya ayrılmış olarak ele alınır

$A \leftarrow A_1 x 10^{N/2} + A_2$
$B \leftarrow B_1 x 10^{N/2} + B_2$

//verilen parçalar üzerinden oluşturulan alt problemler hesaplanır

$X \leftarrow Carpim(A_1, B_1, N/2)$
$Y \leftarrow Carpim(A_1, B_2, N/2)$
$Z \leftarrow Carpim(A_2, B_1, N/2)$
$T \leftarrow Carpim(A_2, B_2, N/2)$

// alt problemlerin çözümleri birleştirilir

return $Xx10^N + (Y + Z)\ x10^{N/2} + T$

Yanıt Anahtarı

1. Girdi : N elemanlı bir tamsayı dizisi ve bir X tamsayısı

   Çıktı : Dizi toplamları X'e eşit iki eleman içeriyorsa, bu sayılar; değilse, 0

   Adım 1: Dizinin olası bütün iki elemanlı alt kümelerini belirle

   (toplamda C (N,2) – N'nin ikili kombinasyonlarının sayısı – tane)

   Adım 2: K isimli bir değişken tanımla ve 1 değerini K değişkenine ata

   Adım 3: Adım 1'de belirlenen K. alt küme için; kümedeki elemanların toplamı X'e eşitse K. alt kümenin elemanlarına dön; değilse, K'yı 1 artır

   Adım 4: Eğer K > C(N,2) ise, 0 değerine dön; değilse, adım 3'e geri dön

2. <u>Kuvvet(X, N)</u>

   girdi : X tamsayısı ve N pozitif tamsayısı

   çıktı : X^N

   A ← 1

   K ← 1

   while K =< N

   A ← A.X

   K ← K + 1

   return A

3. Girdi : N tane harften oluşan bir kelime

   Çıktı : Kelimenin içerdiği A ile başlayan alt kelimelerin sayısı

   «Adım 1: Kelimenin bütün alt kelimelerini belirle.»

   Adım 2: K ve A isimli iki değişken tanımla, 1 değerini K değişkenine ve 0 değerini A değişkenine ata

   Adım 3: Adım 1'de belirlenen K. alt kelime için; alt kelime A ile başlıyorsa A'yı 1 artır ve K'yı 1 artır, değilse K'yı 1 artır

   Adım 4: Eğer K > 2^N ise, A değerine dön; değilse, adım 3'e geri dön

4. Arama algoritmasının geliştirilmesinde azalt & fethet tasarım tekniği kullanılmıştır. Dikkat edilirse algoritmanın her adımında girdi boyuta yarıya düşürülerek algoritmanın daha etkin bir şekilde souç üretmesi sağlanmaktadır.

5. Önce problemin alt problemleri belirlenir ve ana problemler belirlenen alt problemlere bölünür

   - Ana problemi çözmek yerine görece daha kolay alt problemler özyinelemeli bir biçimde çözülür

   - Alt problemlerin çözümleri birleştirilerek ana problemin çözümü elde edilir

6. <u>EnBuyuk([t_i, t_2, ..., t_j])</u>

   girdi : N elemanlı bir tamsayı dizisi

   çıktı : Dizinin en büyük elemanı

   // dizide tek bir eleman kaldıysa algoritma o elemana döner

   if i = j

   return t_i

   else

   mid ← (i + j)/2

   // dizinin ilk yarısının en büyük elemanı

   buyuk1 ← EnBuyuk([t_i, ..., t_{mid}])

   // dizinin ikinci yarısının en büyük elemanı

   buyuk2 ← EnBuyuk([t_{mid}, ..., t_j])

   if buyuk2 < buyuk1

   return buyuk1

   else

   return buyuk2

7. Örneklem üzerinde Kasiyer algoritması çalıştırıldığında, algoritma ilk olarak mümkün olan en yüksek bozuk para çeşidi 10 kuruşla çözüme başlayacak ve finaldeki çıktı kümesine iki 10 kuruş ekleyecektir. Sonrasında kalan 4 kuruş için bozuk para çeşitlerinden 10 kuruş ve 6 kuruş kullanılamayacağından, algoritma bozuk para çeşitlerinden 1 kuruşla yoluna

devam edecek ve çıktı kümesine 4 tane 1 kuruş ekleyecektir. Yani algoritma sonunda elde edilen çözüm iki adet 10 kuruştan ve 4 adet 1 kuruştan, yani toplamda 6 adet bozuk paradan oluşacaktır. Halbuki, 24 kuruş para üstü 4 adet 6 kuruş kullanılarak da verilebilirdi. Dolayısıyla Kasiyer algoritması bu örneklem için optimum çözüm üretmemiştir. Her ne kadar bu algoritma bazı bozuk para çeşitleri kümesi için doğru souç üretse de, bütün bozuk para çeşitleri düşünüldüğünde doğruluk kriterini sağlamamaktadır.

8. Kuvvet2(X, N)
girdi : X tamsayısı ve N pozitif tamsayısı
çıktı : X^N
if N = 0
    return 1
else
    return X.Kuvvet2(X, N-1)

9. Adım 1: K = 1
Adım 2: Dizinin ilk elemanı 6 9'a eşit olmadığından K = 2
Adım 3: K > N olmadığından Adım 2'ye dön
Adım 2: Dizinin ikinci elemanı 7 9'a eşit olmadığından K = 3
Adım 3: K > N olmadığından Adım 2'ye dön
Adım 2: Dizinin üçüncü elemanı 8 9'a eşit olmadığından K = 4
Adım 3: K > N olmadığından Adım 2'ye dön
Adım 2: Dizinin dördüncü elemanı 9 9'a eşit olduğundan K = 4 değerine dön

10. Çarpım algoritmasında alt problemlerin çözümünü birleştirirken kullanılan $Xx10^N + (Y + Z) x10^{N/2} + T$ denklemi; $X = A_1 x B_1$, $Y = A_2 x B_2$ ve $Z = (A_1 + A_2)x(B_1 + B_2)$ olmak üzere, $Xx10^N + (Z - X - Y) x10^{N/2} + Y$ şeklinde düzenlenebilir. Böylelikle aynı problem yalnızca 3 alt problem üzerinden çözülebilecektir. Yeni elde edilen algoritmanın sözde kodu aşağıdaki gibi olacaktır.
YeniCarpim(A, B, N)
girdi : N basamaklı iki sayı
çıktı : Sayıların çarpımı
If n = 1
    return AxB
// sayılar iki parçaya ayrılmış olarak ele alınır
$A \leftarrow A_1x10^{N/2} + A_2$
$B \leftarrow B_1x10^{N/2} + B_2$
//verilen parçalar üzerinden oluşturulan alt problemler hesaplanır
$X \leftarrow YeniCarpim(A_1, B_1, N/2)$
$Y \leftarrow YeniCarpim(A_2, B_2, N/2)$
$Z \leftarrow YeniCarpim(A_1 + A_2, B_1 + B_2, N/2)$
// alt problemlerin çözümleri birleştirilir
return $Xx10^N + (Z - X - Y) x10^{N/2} + Y$

Kaynakça

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.