

BÖLÜM 1

ALGORİTMANIN TEMELLERİ

ANAHTAR KAVRAMLAR

Algoritma, Tasarım Tekniği,
Sözde Kod, Akış Şeması,
Zaman Karmaşıklığı, Alan Karmaşıklığı

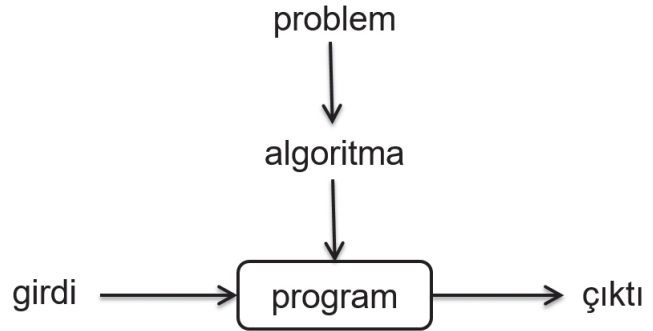
ÖĞRENME HEDEFLERİ

- 1 Algoritma kavramının tanımını bilir
- 2 Algoritmaların sözde kod, akış şeması gibi farklı formlarda nasıl ifade edilebileceğini bilir
- 3 Algoritmanın doğruluğunun ne olduğu ve nasıl gösterilebileceğini bilir
- 4 Algoritmanın performans analizinde kullanılan zaman karmaşıklığı ve alan karmaşıklığı gibi kavramları bilir

GİRİŞ

Bilinen ilk formel örnekleri çok daha eski çağlara dayansa da; kelime olarak ‘algoritma’, 9. yüzyılda yaşamış İranlı matematikçi el-Harizmi’nin isminin Latince karşılığından türetilmiştir. Tanımı üzerine bir mutabakat bulunmasa da; algoritma, belirli bir problemi çözmeye yönelik oluşturulmuş açıkça belirtilmiş adımlar kümesidir denilebilir. Süpermarkette belirli bir ürünü ararken, arkadaşımıza evimizi tarif ederken, ya da internetten aldığımız bir mobilyanın kurulumunu yaparken, aslında farkında olmasak da günlük pratikte sık sık algoritmalara başvurmaktayız.

Yukarıda verilen tanımı bilgisayar bilimlerine uygulayacak olursak; Şekil 1.1’de de gösterildiği gibi algoritma, belirli bir problemin çözümü için tasarlanmış, geçerli veri ya da veri kümesini girdi olarak alan ve sonlu adımda istenen çıktı değerini üreten talimatlar bütünüdür.



Şekil 1.1 Algoritmanın Tanımı (Levitin, 2012)

Bu bölümde ‘algoritma’ kavramının daha iyi anlaşılabilmesi için, algoritmanın temel özellikleri diyebileceğimiz belli bir takım kriterler irdelenecektir.

1. PROBLEMİN ANLAŞILMASI

Verilen bir problem için etkin bir algoritma tasarımında ilk yapılması gereken elbette üzerinde çalışılan problemin iyi bir şekilde anlaşılması olacaktır. Burada problemin anlaşılmasından kasıt, algoritmanın hangi elemanları girdi olarak alacağını, ve neticede hangi sonucu çıktı olarak üreteceğinin açık bir şekilde tespit edilmesidir. Ayrıca; yukarıdaki tanımdan da anlaşılacağı üzere, algoritmanın sadece bazı girdiler için değil, geçerli bütün girdiler için istenen çıktıyı üretmesi gerekmektedir.

Verilen bir tamsayı dizisinin en büyük elemanını bulan bir algoritma tasarladığımızı düşünelim. Tasarlayacağımız algoritma girdi olarak kullanıcıdan belli sayıda tamsayı değeri alacak ve çıktı olarak girilen tamsayıların en büyüğünü üretecektir. Bu problemde olduğu gibi algoritmanın üzerinde çalışacağı girdiyi ve sonuçta üreteceği çıktıyı tespit etmek her problem için bu kadar kolay olmamaktadır. Örneğin verilen iki DNA dizisinin (eşit ya da farklı uzunluklu) benzer

olup olmadıklarını tespit eden bir algoritma tasarladığımızı düşünelim. Tasarladığımız algoritma A, C, G ve T harflerinden oluşan iki kelimeyi girdi olarak alacaktır. Fakat yukarıdaki problem-den farklı olarak bu iki dizinin benzer olup olmadıklarının tespiti için nasıl bir sonuç üretileceği problem-den doğrudan anlaşılammamaktadır. Algoritma tasarımına başlamadan önce benzerliği tespit etmemize imkan verecek sonuç değerinin belirlenmesi gerekmektedir. Örneğin verilen iki DNA dizisinin en uzun ortak alt dizisinin boyutunun kısa dizinin boyutuna oranı benzerliğin ölçümü için kriter olarak kullanılabilir. Böylece geliştirilen algoritmayı problemin aşağıda verilen örneğine uygularsak; girdide de belirtildiği gibi verilen iki DNA dizisinin en uzun ortak alt dizisi CACT'in boyutu 4 ve kısa dizinin boyutu 8 olduğundan çıktı değeri 1/2 olacaktır.

girdi : ACCACTGGT, ATCACTTG

çıkıtı : 1/2

2. TASARIM TEKNİĞİNİN BELİRLENMESİ

Verilen bir probleme etkili bir algoritma geliştirmek için yapılması gerekenlerden bir diğeri de algoritma için probleme uygun tasarım tekniğinin belirlenmesi olacaktır. 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. Kitabın ikinci bölümünde kaba kuvvet, azalt & fethet, böl & fethet, ağaçlı yaklaşım gibi bilgisayar bilimlerinde yaygın olarak kullanılan tasarım teknikleri incelenecektir.

Tasarım tekniğine karar verirken, bazı tekniklerin uygulanabilmesi için problemin belirli bir takım şartları sağlaması gerektiği unutulmamalıdır. Örneğin verilen problemin çözümünde 'böl & yönet' tasarım tekniğinin uygulanabilmesi için, problemin alt problemlere (çoğunukla birbirinden bağımsız) bölünebilmesi ve alt problemlerin çözümü ile orjinal problemin çözüm arasında ilişki kurulabilmesi gerekmektedir.



DİKKAT EDELİM

Algoritma gösteriminde hangi yöntem kullanılırsa kullanılsın, algoritmayı oluşturan adımların açık ve sarıh bir biçimde ifade edilmesi gerekmektedir.



3. ALGORİTMANIN NASIL İFADE EDİLECEĞİNİN BELİRLENMESİ

Algoritma tasarlandıktan sonra elbette belirli bir formda ifade edilmesi gerekmektedir. Verilen bir problemin çözümü için geliştirilmiş algoritma konuşma dilinde ifade edilebileceği gibi, algoritma girdilerinin görsel olarak hangi adımlar üzerinden çıktıya dönüştüğünü gösteren akış şeması ya da sözde kod (pseudocode) üzerinden de gösterilebilmektedir. Burada sözde kod konuşma dili ile programlama dilleri arasında köprü vazifesi gören ve algoritmaların ilgili programlama dillerine kolayca aktarılabilmesine olanak sağlayan bir ara formdur. Algoritma gösteriminde hangi yöntem kullanılırsa kullanılsın, algoritmayı oluşturan adımların açık ve sarıh bir biçimde ifade edilmesi gerekmektedir. Bu doğrultuda;

algoritmaları sözde kodla gösterirken, okuyanın işini ve ilgili algoritmanın programlama dillerine aktarılma sürecini kolaylaştırmak amacıyla kodun gerekli yerlerine ‘//’ notasyonunu kullanarak açıklamalar eklemek faydalı olacaktır.

Algoritmaların yukarıda zikredilen formlar üzerinden nasıl ifade edildiğini bir örnek üzerinden anlamaya çalışalım. Şekil 1.2’de yukarıda da bahsi geçen verilen bir tamsayı dizisinin en büyük elemanını bulma problemi için konuşma dili kullanılarak gösterilmiş bir algoritma verilmiştir. Önce algoritmanın girdi ve çıktı değerleri belirtilmiş ve sonrasında alınan girdi değerleri üzerinden algoritmanın nasıl sonuç değerini ürettiğini adımlar şeklinde gösterilmiştir.

Girdi : N elemanlı bir tam sayı dizisi
Çıktı : Dizinin en büyük elemanı

Adım 1: A ve K adıyla iki değişken tanımla ve dizinin ilk elemanını A değişkenine, 2 değerini K değişkenine ata

Adım 2: Dizinin K. elemanı A’dan büyük ise, A’ya dizinin K. elemanını ata ve K’yı bir artır; değilse, K’yı bir artır

Adım 3: K > N ise, algoritmanın çıktısı olarak A değerine dön; değilse, adım 2’ye geri dön

Şekil 1.2 En Büyük Elemanı Bulma Algoritmasının Konuşma Dilinde Gösterimi

Aşağıda algoritmanın 9, 5 ve 12 sayılarından oluşan 3 elemanlı bir tamsayı dizisi üzerinde na-

sıl çalıştığı ve en büyük elemanı nasıl bulduğu adım adım gösterilmiştir.

Adım 1: A = 9, K = 2

Adım 2: Dizinin K. elemanı (yani ikinci) A’dan büyük olmadığından, K = 3

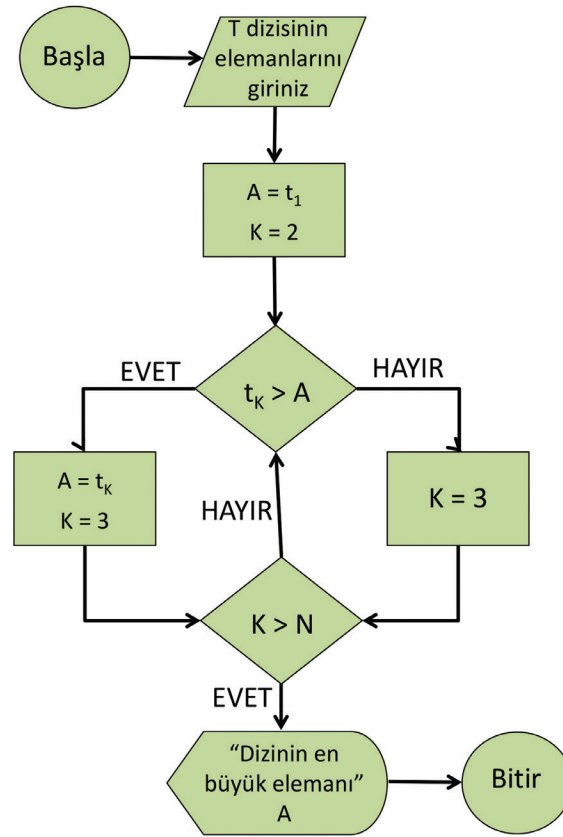
Adım 3: K N’den büyük olmadığından, Adım 2’ye dön

Adım 2: Dizinin K. elemanı (yani üçüncü) A’dan büyük olduğundan, A = 12, K = 4

Adım 3: K N’den büyük olduğundan A’nın değerini yani 12’yi çıktı olarak belirle

Şekil 1.3’de yukarıda konuşma dili kullanılarak gösterilen ve verilen bir tamsayı dizisinin en büyük elemanını bulmaya yarayan algoritmanın akış şeması verilmiştir. Şekilde de görüleceği üzere akış şeması algoritmanın adımlarını belirten kutucuklardan ve adımlar arası geçişleri

gösteren oklardan meydana gelmektedir. Algoritmanın nasıl çalıştığını görsel bir dille anlattığı için diğer alternatifleriyle kıyaslandığında akış şemaları algoritmaların daha anlaşılır bir dille ifadesine imkan sağlamaktadır.



Şekil 1.3 En Büyük Elemanı Bulma Algoritmasının Akış Şeması

Yukarıda da ifade edildiği gibi, sözde kod konuşma dili ile programlama dilleri arasında köprü vazifesi gören ve algoritmaların ilgili programlama dillerine kolayca aktarılabilmesine olanak sağlayan bir ara formdur. Her ne kadar sözde kod programlama dillerindekine benzer ifadeler içerse de, bu ifadelerin bilgisayarlar tarafından anlaşılıp uygulanması mümkün değildir. Dolayısıyla sözde kod ile ifade edilen algoritmaların salt yazıldığı gibi programlama dillerine aktarılması yerine ilgili programlama diline adapte edilmesi gerekmektedir. Sözde kod üzerinden gösterilen algoritmaların programlama dillerine aktarılması diğer alternatiflerine göre daha kolay olduğundan, algoritmaların sözde kod ile ifadesi etkin ve tercih edilen bir yöntemdir. Şekil 1.3’de akış şeması ile gösterilen ve verilen bir tamsayı dizisinin en büyük elemanını bulmaya yarayan algoritmanın sözde kod ile gösterimi Şekil 1.4’de verilmiştir.

```

1  EnBüyük([t1, t2, ..., tN])
2
3  girdi : N elemanlı bir tam sayı dizisi
4  çıktı : Dizinin en büyük elemanı
5
6  // lokal maksimumu tutacak bir A değişkeni tanımlandı
7  // sırasıyla dizinin elemanları tarayabilmek için K değişkeni tanımlandı
8
9  A ← t1
10 K ← 2
11 while K ≤ N
12   // lokal maksimumu tutacak bir A değişkeni tanımlandı
13   if tK > A
14     A ← tK
15     K ← K + 1
16 return A
  
```

Şekil 1.4 En Büyük Elemanı Bulma Algoritmasının Sözde Kodu

Konuşma dilinde olduğu gibi sözde kodla gösterimde de önce algoritmanın girdi ve çıktı bilgisi verilmiştir. Sözde kodla gösterimde farklı notasyonlar kullanılsa da bu kitapta değişkenlere değer atamak için ‘←’ notasyonu kullanılacaktır.

Bunun yanı sıra, okuyanın işini ve ilgili algoritmanın programlama dillerine aktarılma sürecini kolaylaştırmak amacıyla kodun gerekli yerlerine ‘//’ notasyonunu kullanarak açıklamalar eklenmiştir.

Girdi : Başlangıç ve bitiş saatleri verilmiş N tane seminer talebi
Çıktı : Maksimum sayıda seminer içeren bir seminer planı

Adım 1: Başlama saati en erken olan seminer talebini seminer planına ekle

Adım 2: Adım 1’de belirlenen seminer talebini ve bu taleple çakışan seminer taleplerini girdi kümesinden çıkar

Adım 3: Girdi kümesinde seminer talebi kalmadıysa seminer planına dön; değilse, adım 1’e geri dön

Şekil 1.5 Seminer Planlama Algoritması

4. ALGORİTMANIN DOĞRULUĞUNUN İSPATLANMASI

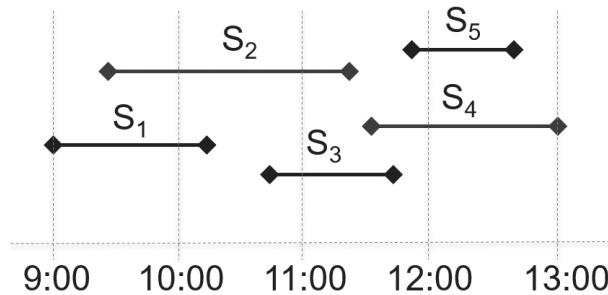
Algoritmalar yukarıda bahsi geçen formlardan biri yoluyla ifade edildikten sonra yapılması gereken algoritmaların doğruluğunun ispatlanması olacaktır. Algoritmaların doğruluğunun ispatıyla neyin kastedildiğini, seminer planı problemi üzerinden anlamaya çalışalım. Seminer planı problemi; tek bir seminer odası için oluşturulmuş başlangıç ve bitiş saatleri belirli bir dizi se-

miner talebi arasından, birbiriyle uyumlu maksimum sayıda semineri içeren bir seminer planı oluşturma şeklinde tanımlanabilir. Burada birbiriyle uyumludan kasıt, oluşturulan planda seminerlerin çakışmamasıdır. Problemi çözümü için Şekil 1.5’de konuşma dili kullanılarak gösterilmiş algoritmayı göz önünde bulunduralım.



DİKKAT EDELİM

Sözde kod ile ifade edilen algoritmaların salt yazıldığı gibi programlama dillerine aktarılması yerine ilgili programlama diline adapte edilmesi gerekmektedir.



Şekil 1.6 Seminer Planı İçin Bir Örneklem

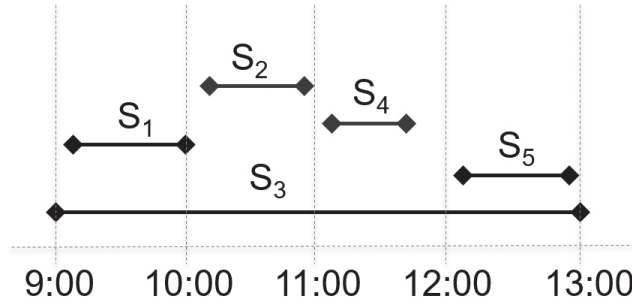
Şekil 1.6'deki örneklem üzerinden yukarıdaki algoritmanın doğruluğunu irdelemeye çalışalım. Örneklemde seminer odası için 9:00 – 13:00 saatleri arasına yerleştirilebilecek 5 talep bulunmaktadır.

Adım 1: S_1 'i seminer planına ekle
 Adım 2: S_1 'i ve S_1 'le çakışan S_2 'yi girdi kümesinden çıkar
 (girdi kümesinde geriye S_3 , S_4 ve S_5 kaldı)
 Adım 3: Girdi kümesinde seminer talebi kaldığından Adım 1'e dön
 Adım 1: S_3 'ü seminer planına ekle
 Adım 2: S_3 'ü ve S_3 'le çakışan S_4 'ü girdi kümesinden çıkar
 (girdi kümesinde geriye S_5 kaldı)
 Adım 3: Girdi kümesinde seminer talebi kaldığından Adım 1'e dön
 Adım 1: S_5 'i seminer planına ekle
 Adım 2: S_5 'i girdi kümesinden çıkar
 Adım 3: Girdi kümesinde seminer talebi kalmadığından seminer planına dön ($\{S_1, S_3, S_5\}$)

Bununla birlikte, algoritmanın bu örnek üzerinde, ya da bir kaç örneklem üzerinde doğru çalıştığını göstermiş olmak algoritmanın doğruluğunu ispatlamak için yeterli değildir. Örneğin aynı algoritma Şekil 1.7'de gösterilen örneklem üzerinde doğru sonuç üretmeyecektir. Şunu belirtmekte fayda var ki, bu örneklem gibi algoritmanın doğru sonuç üretmediği bir örneklem bulmak algoritmanın yanlışlığını ispatlamak için yeterli olacaktır. Diğer yandan, algoritmanın

maktadır. Aşağıda algoritmanın verilen örneklem üzerinde nasıl çalıştığı ve seminer planını nasıl bulduğu adım adım gösterilmiştir.

aşağıdaki örneklem benzer doğru sonuç üretmediği herhangi bir örneklem bulamıyor olmak da algoritmanın doğruluğunu ispatlamak için yeterli olmayacaktır. Algoritmanın doğruluğunu ispatlamak için yapılması gereken; doğrudan ispat, çelişki yoluyla ispat ya da tümevarım gibi matematiksel ispat yöntemleri kullanarak formel olarak algoritmanın bütün geçerli girdi değerleri için doğru çıktı değerini ürettiğini göstermektir.



Şekil 1.7 Seminer Planı İçin Alternatif Bir Örneklem

5. ALGORİTMANIN ETKİNLİK ANALİZİNİN YAPILMASI

Algoritmalar tasarlandıktan sonra yapılması gereken bir diğer şey de geliştirilen algoritmanın performansının ya da etkinliğinin analizini yapmak olacaktır. Etkinlik analizi algoritmanın pratikte uygulanıp uygulanamayacağına dair fikir verirken, aynı zamanda aynı problem için geliştirilmiş farklı algoritmaları da kıyaslayabilmemize imkan sağlamaktadır. Algoritmalar için iki tür etkinlik analizi göz önünde bulundurulmaktadır:

zaman etkinliğinin analizi (ya da zaman karmaşıklığı) algoritmanın verilen girdi için ne kadar sürede ya da ne kadar adımda istenen çıktıyı ürettiğinin belirlenmesi, alan etkinliğinin analizi (ya da alan karmaşıklığı) algoritmanın verilen girdi için istenen sonucu üretirken bellekten ne kadarlık kısma ihtiyaç duyduğu şeklinde tanımlanabilir. Algoritmanın etkinlik analizi Bölüm 5'te detaylıca incelenecektir.

BÖLÜM ÖZETİ

Bu bölümde; belirli bir problemi çözmeye yönelik oluşturulmuş açıkça belirtilmiş adımlar kümesi şeklinde tanımlanabilecek algoritma kavramının tasarım süreci; doğruluk, performans analizi gibi kriterler üzerinden incelenmiştir. Verilen bir problem için etkin bir algoritma tasarımında ilk yapılması gereken elbette üzerinde çalışılan problemin iyi bir şekilde anlaşılması olacaktır. Burada problemin anlaşılmasından kasıt, algoritmanın hangi elemanları girdi olarak alacağını ve neticede hangi sonucu çıktı olarak üreteceğinin açık bir şekilde tespit edilmesidir. Tasarım sürecinde bir sonraki adım; kaba kuvvet, böl & yönet, açgözlü yaklaşım gibi tasarım teknikleri arasından probleme en uygun olanını belirlemek olacaktır. Bunların yanı sıra, elbette geliştirilen algoritmanın konuşma dili, akış şeması ya da sözde kod gibi gösterim yöntemlerinden hangisi kullanılarak ifade edileceğinin belirlenmesi de gerekmektedir. Bölümde algoritmaların bu formlarda nasıl ifade edildikleri örnekler üzerinden anlatılmıştır. Algoritmalar bahsi geçen bu formlardan biri vasıtasıyla ifade edildikten sonra yapılması gereken bir diğer şey, algoritmaların doğruluğunun ispatı olmalıdır. Bölümde algoritmalar için doğruluk kriteri ile neyin kastedildiği örnek üzerinden detaylıca ele alınmıştır. Son olarak algoritma tasarımında atlanmaması gereken bir diğer şey de geliştirilen algoritmanın performansının ya da etkinliğinin analizini yapmak olmalıdır. Etkinlik analizi algoritmanın pratikte uygulanıp uygulanamayacağına dair fikir verirken, aynı zamanda aynı problem için geliştirilmiş farklı algoritmaları da kıyaslayabilmemize imkân sağlamaktadır.

GÖZDEN GEÇİRELİM

1. Algoritma kavramını tanımlayınız.

2. Seminer planı problemi; tek bir seminer odası için oluşturulmuş başlangıç ve bitiş saatleri belirli bir dizi seminer talebi arasından, biririyle uyumlu maksimum sayıda semineri içeren bir seminer planı oluşturma şeklinde tanımlanmaktadır. Buna göre seminer planı probleminin çözümü için geliştirilmiş algoritmanın hangi girdi değerlerini aldığını ve sonuçta hangi çıktı değerini ürettiğini belirleyiniz.

3. Adem N basamaklı bir merdiveni çıkmaya çalışıyor ve her adımda ya tek basamak ya da iki basamak çıkabiliyor. Adem'in merdiveni kaç farklı şekilde çıkabileceğini belirleyen bir algoritmanın hangi girdi değerlerini aldığını ve sonuçta hangi çıktı değerini ürettiğini belirleyiniz.

4. Girilen iki tamsayıdan büyük olanını belirleyen bir algoritma tasarlayınız ve algoritmanın akış şemasını gösteriniz.

5. Verilen bir tamsayı dizisinin elemanlarının toplamını hesaplayan bir algoritma tasarlayınız ve algoritmanın konuşma dilinde gösterimini yazınız.

6. Konuşma dilindeki gösterimi aşağıdaki gibi olan algoritmanın sözde kod ile gösterimini yazınız.

Girdi : N elemanlı bir 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 A'ya 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

7. Sözde kod ile gösterimi aşağıdaki gibi olan algoritmanın konuşma dilindeki gösterimini yazınız.

Büyük($[t_1, t_2, \dots, t_N], X$)

girdi : N elemanlı bir tamsayı dizisi ve X tamsayısı

çıktı : Dizinin X'ten büyük elemanlarının sayısı

// X'ten büyük elemanların sayısını tutacak A değişkeni tanımlandı

// sırasıyla dizinin elemanlarını tarayabilmek için K değişkeni tanımlandı

$A \leftarrow 0$

$K \leftarrow 1$

while $K \leq N$

if $t_K > X$

$A \leftarrow A + 1$

$K \leftarrow K + 1$

return A

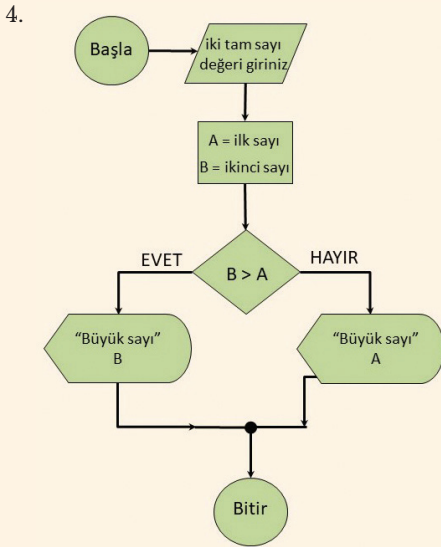
8. Verilen bir tamsayı dizisinin farklı elemanlardan oluşup oluşmadığını belirleyen bir algoritma tasarlayınız ve algoritmanın sözde kod ile gösterimini yazınız.

10. Zaman karmaşıklığı ve alan karmaşıklığı kavramlarını tanımlayınız.

9. Algoritmalar için doğruluk kriterini tanımlayınız.

Yanıt Anahtarı

1. Belirli bir problemi çözmeye yönelik oluşturulmuş açıkça belirtilmiş adımlar kümesidir.
2. Girdi: Başlangıç ve bitiş saatleri verilmiş N tane seminer talebi, Çıktı : Maksimum sayıda seminer içeren bir seminer planı
3. Girdi : Merdivendeki basamak sayısı N, Çıktı : N'nin bir fonksiyonu olarak Adem'in merdivenleri kaç farklı şekilde çıkabileceğinin sayısı



5. Girdi : N elemanlı bir tamsayı dizisi
Çıktı : Dizinin elemanlarının toplamı
Adım 1: A ve K adıyla iki değişken tanımla ve dizinin ilk elemanını A değişkenine, 2 değerini K değişkenine ata
Adım 2: Dizinin K. elemanını A'ya ekle ve K'yı bir artır
Adım 3: $K > N$ ise, algoritmanın çıktısı olarak A değerine dön; değilse, adım 2'ye geri dön
6. Arama($[t_1, t_2, \dots, t_N], X$)
Girdi : N elemanlı bir tamsayı dizisi ve X tamsayısı
Çıktı: Dizi X'i içeriyorsa, dizinin X'e eşit ilk

elemanının indeksi; değilse, 0 değeri
// sırasıyla dizinin elemanları tarayabilmek için K değişkeni tanımlandı
 $K \leftarrow 1$
while $K \leq N$
if $t_K = X$
return K
 $K \leftarrow K + 1$
return 0

7. Girdi: N elemanlı bir tamsayı dizisi ve X tamsayısı
Çıktı: Dizinin X'ten büyük elemanlarının sayısı
Adım 1: A ve K adıyla iki değişken tanımla ve 0 değerini A değişkenine, 1 değerini K değişkenine ata
Adım 2: Dizinin K. elemanı A'dan büyük ise, A'yı 1 artır ve K'yı bir artır; değilse, K'yı bir artır
Adım 3: $K > N$ ise, algoritmanın çıktısı olarak A değerine dön; değilse, adım 2'ye geri dön

8. Ayrık Vadi($[t_1, t_2, \dots, t_N]$)
girdi : N elemanlı bir tamsayı dizisi
çıkı : Dizi farklı elemanlardan oluşuyorsa 'true'; değilse, 'false'
for ($i = 1$ to N)
for ($j = i$ to N)
if $t_i = t_j$
return false
return true

9. Algoritmanın doğruluğu, algoritmanın bütün geçerli girdi değerleri için doğru çıktı değerini üretmesi şeklinde tanımlanmaktadır.

10. Zaman karmaşıklığı algoritmanın verilen girdi için ne kadar sürede ya da ne kadar adımda istenen çıktıyı ürettiğinin belirlenmesi, alan karmaşıklığı ise algoritmanın verilen girdi için istenen sonucu üretirken bellekten ne kadarlık kısma ihtiyaç duyduğu şeklinde tanımlanmaktadır.

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.

