Giriş

Etkin programlama yaparken esas alınan iki temel veri yapıları ve algoritma analizidir. Bugün algoritma analizine giriş yapıcağız.

Algoritma Analizine Giriş

Algoritma, bir problemin çözmek için bir prosedür veya formüldür. Problemi çözmek için takip edilmesi gereken yönergeler kümesidir. Matematikte ve bilgisayar biliminde bir işi yapmak için tanımlanan, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumunda sonlanan, sonlu işlemler kümesidir.

Program, bir programlama dilinde algoritmanın gerçekleştirilmesidir.

Algoritmaların Özellikleri

1. Giriş (Input): Bir algoritmanın sıfır veya daha fazla giriş değişkeni vardır. Giriş değişkenleri algoritma işlemeye başlamadan önce, algoritmaya verilen değerler kümesidir veya değer kaydetmesi için verilen hafıza bölgesidir.
2. Belirlilik (Definiteness): Bir algoritmanın her adımı için kesin olarak ne iş yapacağı belirlenmelidir ve belirsizlik olmamalıdır. Her durum için hangi işlem gerçekleştirilecekse, o açık olarak tanımlanmalıdır.
3. Çıkış (Output): Her algoritmanın bir veya daha fazla çıkış değeri vardır. Çıkış değerleri ile giriş değerleri arasında bağıntılar vardır.
4. Etkililik (Efficiency): Olabildiğince hızlı çalışmalıdır, olabildiğince az hafıza kullanmalıdır. Bunun anlamı yapılan işlemler yeterince temel işlemler olacak ve sınırlı zaman süresince işleyip bitmelidir.
5. Sınırlılık (Boundedness): Her algoritma sınırlı sayıda çalışma adımı sonunda bitmelidir. Bir algoritma için sınırlılık çok önemlidir. Aynı işlemi yapan iki algoritmadan biri bir milyar adımda bitiyor olsun ve diğeri de yüz adımda bitiyor olsun. Bu durumda yüz adımda biten algoritma her zaman daha iyidir. Bunun anlamı sınırlılık kavramı ile anlatılmak istenen mümkün olan en az sayıda adım ile işlemin bitirilmesidir.

Algoritmaları tasarlamada kullanılacak yöntemler

Özyineleme

Problemin çözümünün tekrarlı olması durumunda bilinen bir veya birkaç çözümden faydalanarak bir sonraki çözümü elde etme ve elde edilen çözüm ile önceki çözümlerin birkaçının kullanılması ile bir sonraki çözümün elde edilmesi ile problemi çözme işlemine özyineleme yöntemi denir.

Böl ve fethet

Kompleks problemlerin bir bütün olarak çözülmeleri çok zor olacağından dolayı, bu problemler alt problemlere bölünürler. Bu bölünme işleminin yapılabilmesi için alt problemlerin bir üst seviyedeki problem ile aynı özelliği sağlamalıdır. Bu yöntem ile algoritma tasarımı yapmaya böl ve fethet yöntemi denir.

Dinamik programlama

Böl ve yönet yöntemine benzer olarak alt problemlerin çözümlerini birleştirerek çözüme gitme mantığına sahip olup alt problem tekrarı varsa, bunlardan bir tanesi çözülür ve bu çözüm diğer tekrarlarda kullanılır. Bu yöntem ile yapılan algoritma tasarım yöntemine dinamik programlama yöntemi denir.

Kaba Seçim veya Haris (Greedy) algoritması

Optimizasyon problemlerinin çözümü için yerel optimumların seçilmesi ilkesinden yola çıkar ve veriyi belli bir kritere göre düzenledikten sonra ilk veri her zaman optimum çözüme götürür mantığına sahiptir. Temel amaç en iyi sonucu elde etmek için en iyi ara adım çözümlerini seçmeye yönelik bir yöntem olduğundan bu yönteme haris algoritması yöntemi denir.

Bir veri yapısı icat etme

O ana kadar var olamayan bir veri yapısının icat edilmesi ile problemin çözülmesine veri yapısı icat etme yöntemi denir.

Bilinen probleme indirgeme

Kompleks olan bir problemin çözümünü yapmak için çözümü bilinen bir veya birden fazla başka probleme dönüştürüp bu şekilde problemi çözme işlemine bilinen probleme indirgeme yöntemi denir.

İhtimali (olasılıksal) çözümler

Bazı durumlarda gelişigüzellik ilkesi ile etkili bir şekilde problem çözümü yapılabilmektedir. Bunlara örnek olarak Las Vegas polinom-zamanlı ve Monte Carlo polinom-zamanlı algoritmalar verilebilir. Gelişigüzellik kullanılarak yapılan problem çözümlerine ihtimali çözümler yöntemi denir.

Yaklaşım çözümleri

Çözümü deterministik Turing makinası ile yapılamayan yani karmaşık hesaplamaların belirli bir yöntem ile çözülemediği bu problemlerin bir kısmına bazı kriterler uygulayarak yaklaşım mantığı ile çözüm üretilebilmektedir. Bundan dolayı bu mantık ile yapılan algoritma tasarımına yaklaşım çözümler yöntemi adı verilir.

Algoritmik Performans : iki yönü vardır.

Zaman(Time),Alan(Space)

Algoritma analizi, özel uygulamalardan, bilgisayarlardan veya veriden bağımsızdır.

Algoritma analizi, tasarlanan program veya fonksiyonun belirli bir işleme göre matematiksel ifadesini bulmaya dayanır.

Algoritmaları analiz etmek;

İlk olarak, algoritmanın etkinliğini değerlendirmek için belirli bir çözümde anlamlı olan işlemlerin kaç adet olduğu sayılır.

Daha sonra büyüme fonksiyonları kullanılarak algoritmanın verimliliği ifade edilir.

Problem	n elemanlı giriş	Temel işlem
Bir listede arama	liste ***n*** elemanlı	karşılaştırma
Bir listede sıralama	liste ***n*** elemanlı	karşılaştırma
İki matrisi çarpma	***n x n*** boyutlu iki matris	çarpma
Bir ağaçta dolaşma	***n*** düğümlü ağaç	Bir düğüme erişme
Hanoi kulesi	***n*** disk	Bir diski taşıma

Not: Temel işlem tanımlayarak bir algoritmanın karmaşıklığını ölçebiliriz ve giriş büyüklüğü n için bu temel işlemi algoritmanın kaç kez gerçekleştirdiğini sayabiliriz.

Anlamlı olan işlemler hakkında önemli not:

Eğer problemin boyutu çok küçük ise algoritmanın verimliliğini muhtemelen ihmal edebiliriz.

Algoritmanın zaman ve bellek gereksinimleri arasındaki ağırlığı dengelemek zorundayız.

Dizi tabanlı listelerde geri alma işlemleri O(1)'dir. Bağlı listelerde geri alma işlemi ise O(n)'dir. Fakat eleman ekleme ve silme işlemeleri bağlı liste uygulamalarında çok daha kolaydır.

Çalışma zamanı veya koşma süresi (running time) fonksiyonu:

'n' boyutlu bir problemin algoritmasını çalıştırmak için gerekli zamandır ve T(n) ile gösterilir.

Başka bir ifadeyle **T(n): bir programın veya algoritmanın işlevini yerine getirebilmesi için, döngü sayısı, toplama işlemi sayısı, atama sayısı gibi işlevlerden kaç adet yürütülmesini veren bir bağıntıdır.**

Basit **if** bildirimi

	Cost	Times
`if (n < 0)`	c1	1
`  absval = -n;`	c2	1
`else`		
`  absval = n;`	c3	1

- Toplam maliyet <=c1+max(c2,c3)

Tahmin için Genel Kurallar

Döngüler (Loops)

Bir döngünün çalışma zamanı en çok döngü içindeki deyimlerin çalışma zamanının iterasyon sayısıyla çarpılması kadardır.

İç içe döngüler (Nested Loops)

İç içe döngülerde grubunun içindeki deyimin toplam çalışma zamanı, deyimlerin çalışma sürelerinin bütün döngülerin boyutlarının çarpımı kadardır. Bu durumda analiz içten dışa doğru yapılır.

Ardışık deyimler

Her deyimin zamanı birbirine eklenir.

if/else

En kötü çalışma zamanı: test zamanına then veya else kısmındaki çalışma zamanının hangisi büyükse o kısım eklenir.

- **Örnek:** Basit bir döngü — Maliyet — Tekrar
  - i = 1; — c1 — 1
  - sum = 0; — c2 — 1
  - while (i <= n) { — c3 — n+1
  - i = i + 1; — c4 — n
  - sum = sum + i; — c5 — n
  - }
  - Toplam maliyet=c1 + c2 + (n+1)*c3 + n*c4 + n*c5 =3n+3
  - T(n)=3n+3
- Bu algoritma için gerekli zaman ***n*** ile doğru orantılıdır.

- **Örnek:** İç içe döngü — Maliyet — Tekrar
  - i=1; — c1 — 1
  - sum = 0; — c2 — 1
  - while (i <= n) { — c3 — n+1
  - j=1; — c4 — n
  - while (j <= n) { — c5 — n*(n+1)
  - sum = sum + i; — c6 — n*n
  - j = j + 1; — c7 — n*n
  - }
  - i = i +1; — c8 — n
  - }
  - Toplam maliyet= c1 + c2 + (n+1)*c3 + n*c4 + n*(n+1)*c5+ n*n*c6+ n*n*c7+ n*c8
  - Bu algoritma için gerekli zaman n^2 ile doğru orantılıdır.

Asimptotik Analizi

Algoritmanın Büyüme Oranları

Bir algoritmanın orantılı zaman gereksinimi büyüme oranı (veya büyüme hızı) olarak bilinir.

T(n) nin büyüme oranı, algoritmanın hesaplama karmaşıklığıdır.

Hesaplama karmaşıklığı belirli bir uygulamadan bağımsız olarak n ile değişen T(n)' nin çalışma zamanını daha doğru bir şekilde bulmayı sağlar.

Genel olarak, az sayıda parametreler için karmaşıklıkla ilgilenilmez; eleman sayısı n'nin sonsuza gitmesi durumunda T(n) büyümesine bakılır.

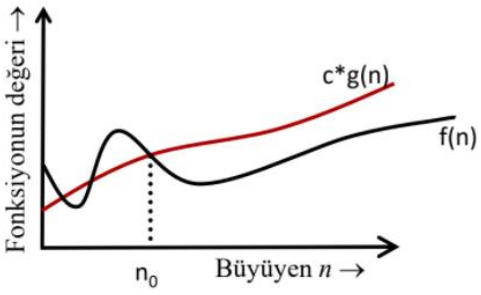
Karmaşıklığı belirtmek için asimtotik notasyon (simgelem) ifadeleri kullanılmaktadır.

Asimptotik simgelem (notasyon)

O-simgelemi (üst sınırlar)

Tüm $n \geq n\_0$ değerleri için sabitler $c > 0$, $n\_0 > 0$ ise
$0 \leq f(n) \leq cg(n)$ durumunda
$f(n) = O(g(n))$ yazabiliriz.

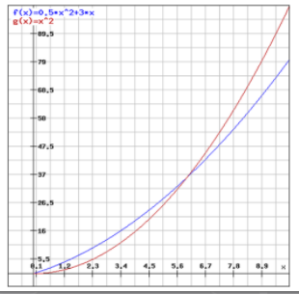
- ***f(n)*** *ve* ***g(n)*** verilen iki çalışma zamanı fonksiyonudur.
- ***f(n)*** $\leq$ ***c.g(n)*** ve ***n*** $\geq$ n_0 *koşullarını sağlayan* ***c*** *ve* n_0 *değerleri varsa* ***f(n)*** *zaman karmaşıklığı* ***O(g(n))*** *dir.*
- *Başka bir deyişle, n sayısı yeteri kadar büyük olduğunda, f(n), g(n) ile aynı büyüklüktedir.*
- *O-notasyonu* sabit bir katsayı içinde bir fonksiyon için üst sınırı verir



O-simgelemi (üst sınırlar)

- ***Örnek:*** $2n^2 = O(n^3)$ *için c,* n_0 *değerlerini bulunuz?*
- $0 \leq f(n) \leq c.g(n)$, $0 \leq 2n^2 \leq cn^3$
- ***c=1*** için $n_0 = 2$, *şartı sağlar.*
- ***Örnek:*** *(1/2)*n^2*+ 3n için üst sınırın* $O(n^2)$ *olduğunu gösteriniz.*
- *c=1 için*
- $(1/2)n^2 + 3n \leq n^2$
- $3n \leq 1/2n^2$
- $6 \leq n$,
- $n_0 = 6$

Çözüm kümesini sağlayan kaç tane n_0 ve **c** çifti olduğu önemli değildir. Tek bir çift olması notasyonun doğruluğu için yeterlidir.



O-notasyonu

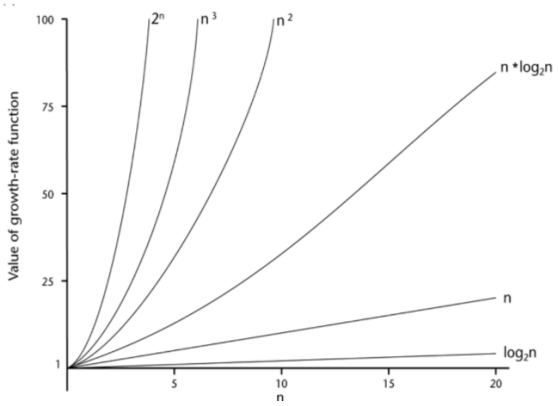
- **Örnek**
- $3n^2+2n+5 = 0(n^2)$ olduğunu gösteriniz
- $10\ n^2 = 3n^2 + 2n^2 + 5n^2$
- $\geq 3n^2 + 2n + 5$, $n \geq 1$
- $c = 10$, $n_0 = 1$

Ortak Büyüme Oranları

Maliyet artar

Fonksiyon	Büyüme oranı ismi
c	Sabit, komut bir veya birkaç kez çalıştırılır. Yenilmez!
log n	Logaritmik, Büyük bir problem, her bir adımda sabit kesirler tarafından orijinal problemin daha küçük parçalara ayrılması ile çözülür. İyi hazırlanmış arama algoritmalarının tipik zamanı
$\log^2 n$	Karasel logaritmik
n	Doğrusal, Küçük problemlerde her bir eleman için yapılır. Hızlı bir algoritmadır. N tane veriyi girmek için gereken zaman.
n log n	Doğrusal çarpanlı logaritmik. Çoğu sıralama algoritması
n^2	Karasel. Veri miktarı az olduğu zamanlarda uygun (n<1000)
n^3	Kübik. Veri miktarı az olduğu zamanlarda uygun (n<1000)
2^n	İki tabanında üssel. Veri miktarı çok az olduğunda uygun (n<=20)
10^n	On tabanında üssel
n!	Faktöriyel
n^n	n tabanında üstel (çoğu ilginç problem bu kategoride)

Büyüme oranı fonksiyonlarının karşılaştırılması



Notasyonlarda '=' gösterimi

A=B ise B=A anlamında değil mi?

Hayır değil. Buradaki = anlamı üyelik işlemi (Є) olarak tercih edilmiştir.

f(n) = O(g(n)) → f(n) Є O(g(n)) : Burada O(g(n)) bir küme anlamına gelir.

f(n) = O(g(n)) → O(g(n)) ={ f(n) gösterimi doğrudur.

Ω notasyonu-Örnek

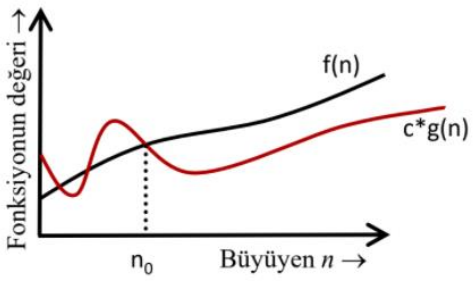
- $2n + 5 \in \Omega(n)$ olduğunu gösteriniz
  - $n_0 \geq 0$, $2n+5 \geq n$, olduğundan sonuç elde etmek için $c=1$ ve $n_0 = 0$ alabiliriz.
- $5*n^2 - 3*n = \Omega(n^2)$ olduğunu gösteriniz.
  - $5*n^2 - 3*n \geq n^2$, $c=1$, $n_0 = 0$ değerleri için sağlar

Diğer Asimptotik Notasyonlar

Ω-simgelemi (alt sınırlar)

$$\Omega(g(n)) = \{ f(n) : \text{tüm } n \geq n\_0 \text{ değerlerinde } c > 0,\ n\_0 > 0 \text{ ise, } 0 \leq cg(n) \leq f(n) \}$$

- Her durumda **$f(n) \geq c\, g(n)$** ve **$n \geq n_0$** koşullarını sağlayan pozitif, sabit **c** ve **n_0** değerleri bulunabiliyorsa **$f(n)=\Omega(g(n))$** dir.



Examples

- $5n^2 = \Omega(n)$

  $\exists\ c, n_0$ such that: $0 \leq cn \leq 5n^2 \Rightarrow cn \leq 5n^2 \Rightarrow c = 1$ and $n_0 = 1$

- $100n + 5 \neq \Omega(n^2)$

  $\exists\ c, n_0$ such that: $0 \leq cn^2 \leq 100n + 5$

  $100n + 5 \leq 100n + 5n\ (\forall\ n \geq 1) = 105n$

  $cn^2 \leq 105n \Rightarrow n(cn - 105) \leq 0$

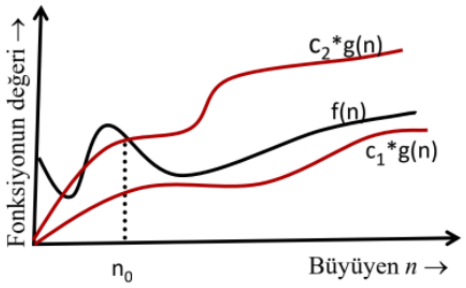
  Since n is positive $\Rightarrow cn - 105 \leq 0 \Rightarrow n \leq 105/c$

  $\Rightarrow$ contradiction: n cannot be smaller than a constant

- $n = \Omega(2n)$, $n^3 = \Omega(n^2)$, $n = \Omega(\log n)$

Θ Notasyonu – Sıkı Sınırlar

- Her durumda $c_1.g(n) \leq f(n) \leq c_2.g(n)$ ve $n \geq n_0$ koşullarını sağlayan pozitif, sabit c_1,c_2 ve n_0 değerleri bulunabiliyorsa $f(n)= \Theta(g(n))$ ifadesi doğrudur.



Θ notasyonu- Örnek

- $f(n) = 2n + 5 \in \Theta(n)$.
  - $2n \leq 2n+5 \leq 3n$, tüm $n \geq 5$ için

- $f(n) = 5*n^2 - 3*n \in \Theta(n^2)$.
  - $4*n^2 \leq 5*n^2 - 3*n \leq 5*n^2$, tüm $n \geq 4$ için

Examples

- $n^2/2 - n/2 = \Theta(n^2)$
  - $\frac{1}{2}\, n^2 - \frac{1}{2}\, n \leq \frac{1}{2}\, n^2 \; \forall n \geq 0 \Rightarrow c_2 = \frac{1}{2}$
  - $\frac{1}{2}\, n^2 - \frac{1}{2}\, n \geq \frac{1}{2}\, n^2 - \frac{1}{2}\, n * \frac{1}{2}\, n \; (\forall n \geq 2) = \frac{1}{4}\, n^2 \Rightarrow c_1 = \frac{1}{4}$

- $n \neq \Theta(n^2)$: $c_1\, n^2 \leq n \leq c_2\, n^2 \Rightarrow$ only holds for: $n \leq 1/c_1$
- $6n^3 \neq \Theta(n^2)$: $c_1\, n^2 \leq 6n^3 \leq c_2\, n^2 \Rightarrow$ only holds for: $n \leq c_2/6$

n değerini keyfi olarak belirlemek imkansızdır. Çünkü c_2 sabittir.

Another example

- Prove that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
  - **Determine c_1, c_2 and n_0 such that**

$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$$

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$$

$$\frac{1}{2} - \frac{3}{n} \le c_2 \rightarrow n \ge 1, c_2 \ge \frac{1}{2}$$

$$c_1 \le \frac{1}{2} - \frac{3}{n} \rightarrow n \ge 7, c_1 \le \frac{1}{14}$$

For any polynomial $p(n) = \sum_{i=0}^{d} a_i n^i$
$p(n) = \Theta(n^d)$

- $c_1 = 1/14$, $c_2 = 1/2$, $n_0 = 7$

O, Ω ve Θ notasyonları arasındaki ilişkiler

- Eğer g(n) = Ω(f(n)) ise→ f(n)=O(g(n))
- Eğer f(n)= O(g(n)) ve f(n)= Ω(g(n)) ise →f(n)= Θ(g(n))
- Θ(g(n)) = O(g(n)) ∩ Ω (g(n))

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

Bundan sonrakilerde şimdiye kadar incelediğimiz algoritma analizi konusunda ve basit iki tane pseudo codeları bulunuyor. Bazı açıklamaları da yaptım. Bunları önümüzdeki haftaya kadar tamamla, takıldığın yerde bana sor.

Alıştırmalar

1. **Aşağıdakileri kısaltarak tekrardan yazınız.**

- $O(n^3+4n^2+3n)$
- $O(8n^4)$

Diziler

Diziler, tipleri homojen olan birden fazla elemandan oluşan veri grubudur. Diziler tek boyutlu, iki boyutlu, üç boyutlu, vb. şeklinde tanımlanabilirler.

Diziler genelde diğer veri yapılarında kullanılan bir veri yapısıdır. Diziler **statik veriler** olarak da isimlendirilebilirler.

Örnek: Tek boyutlu A dizisi

0	1	2	3	4	5	6	7	8	9
6	7	12	23	46	78	12	5	8	2

2. **Buradaki pseudo codu python ile tekrar yaz. Opsiyonel olarak analizini yap.**

```
    Lineer Arama(n,x)
1.  i←1, Veri_Var←0
2.  (i<=n) ve ~ (Veri_Var) olduğu sürece devam et
3.      eğer Dizi[i]=x ise
4.          Veri_Var←1
5.      i←i+1
```

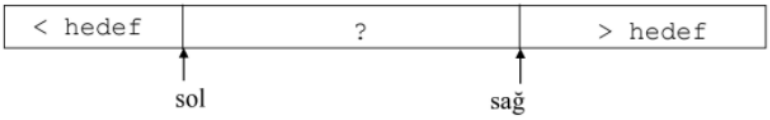
X, dizi içinde aranacak veri ve n dizinin boyutudur.

~ simgesinin anlamı önüne geldiği mantılsal ifadenin değilini almaktadır.

İkili Arama

İkili aramada dizi sıralanmış olduğundan, dizinin ortasında bulunan sayı ile aranan sayıyı karşılaştırarak arama boyutunu yarıya düşürülür ve bu şekilde devam edilir.

Bu algoritmanın temel mantığı aranacak elemanı dizinin ortasındaki eleman ile karşılaştırıp, aranacak eleman eğer bu elemana eşitse, amaca ulaşılmıştır. Eğer eşit değilse, bu durumda aranacak eleman dizinin hangi parçası içinde olabileceği kararı verilir. Bu sayede arama boyutu yarıya düşürülür ve bu şekilde devam edilir.



3. **Buradaki pseudo codu python ile tekrar yaz. Opsiyonel olarak analizini yap.**

```
İkili Arama(Dizi,n,x,bulundu, yeri)

1.  Yerel değişkenler
    alt, ust, orta : integer;
2.  Ust ← n, alt ← 1, bulundu ← 0
3.  (bulundu = 0) ve (ust ≥ alt) olduğu sürece devam et
4.      orta ← ⌊ ( alt + ust ) / 2 ⌋
5.      eğer x = Dizi[orta] ise
6.          bulundu ← 1
7.      değil ve eğer x < Dizi2 [ orta ] ise
8.          ust ← orta-1
9.      değilse
10.         alt ← orta+1
11. yeri ← orta
```