ALGORİTMALAR

Algoritmanın Tanımı, Matematikteki Yeri ve Önemi

- Endüstri ve hizmet sektörü organizasyonlarının karmaşıklığının artan bir yapıda olması, büyük ölçekli optimizasyon problemleri için çözümleri ve mevcut verimliliği devam ettirmede yeni alternatiflerin belirlenmesini gerektirir.

- Bu büyük ölçekli optimizasyon problemlerinin formülasyonu, kompleks matematik modellere yol açar ve bu modellerin çözümü yalnızca çok güçlü hesaplama kaynakları kullanılarak elde edilebilir.

Algoritmanın Tanımı, Matematikteki Yeri ve Önemi

- Böyle bir matematik uygulamanın kesikli bir yapıda olması gerekir. Bundan dolayı, algoritma fikri bu tip uygulamalarda önemli bir rol oynar.

- Algoritmanın kesikli bir yapıya sahip uygulamalardaki rolü, matematiğin klasik branşlarında bir fonksiyon bilgisinin önemi gibidir.

- Matematikte yeri ve önemi böylesine net olan algoritmalara genel anlamda bir prosedür gözü ile bakılabilir. Bu anlamda, bir algoritma için yaygın olarak kullanılan tanımlardan bazıları şunlardır:

Algoritmanın Tanımı, Matematikteki Yeri ve Önemi

- Algoritmalar, problemleri çözmek için adım adım prosedürlerdir.
- Algoritma, bilgisayarda problemlerin bir sınıfını çözmek için bir metoddur.
- Algoritma, bir mekanik kural veya otomatik metod veya bazı matematiksel işlemlerin düzenlenmesi için programdır.
- Algoritma, soruların herhangi verilen bir sınıfına cevaplar bulmakta kullanılabilen bir hesaplama prosedürü (nümerik olması gerekmeyen) için etkili komutların kümesidir.
- Algoritma, açık olarak tanımlanmış olan ve herhangi bir bilgisayara icra edilen bir prosedürdür.

Algoritmanın Tanımı, Matematikteki Yeri ve Önemi

- Algoritma için bu tanımları verdikten sonra, algoritmanın, matematikteki yeri ve önemini şöyle özetleyebiliriz:

- Her algoritmanın bünyesinde aritmetik ve mantıki adımları bulundurması, tersine matematiksel çözüm isteyen ve analitik yapıda olmayan bütün problemlerin bir algoritma ile ifade edilmesi ve algoritmaların çalışma sürelerinin fonksiyonel bir yapıda olması, algoritmaların matematikteki yeri ve önemini belirtir.

Algoritmalar Sistemi

- Bir algoritma, belirli bir problem için tatmin edici bir çözüme yakınsamayı sağlayan bir prosedürdür.

- Endüstrideki problemlerin bazıları çok kompleks olduğu için, Yöneylem Araştırmasının standart modelleri, bu tip problemler için uygun değildir.

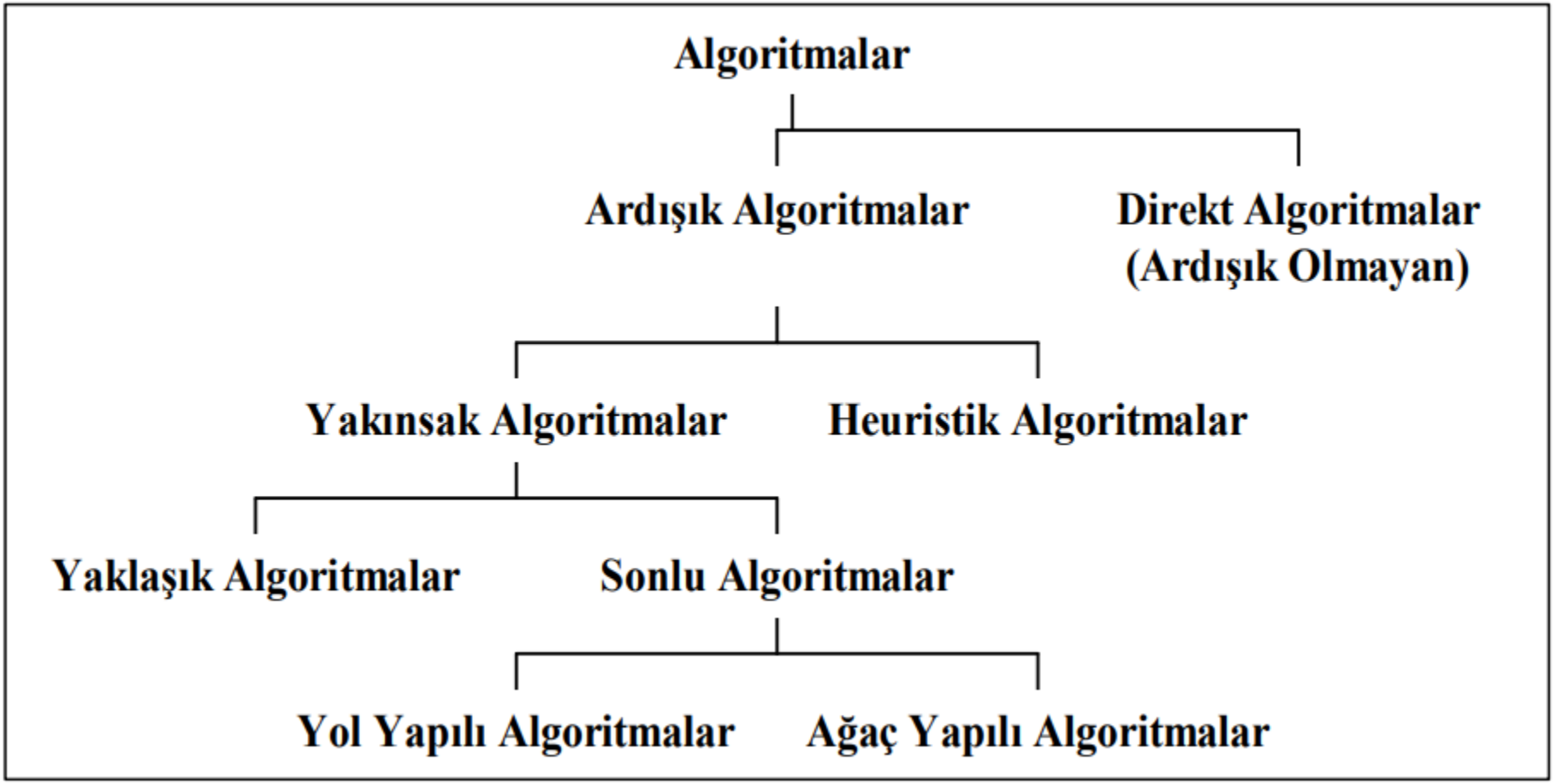
Algoritmalar Sistemi

- Bu durum;
  - Problemi tarif etmek için gereken verinin çok fazla olduğu
  - Problem için doğru bilgi toplamanın zor olduğu durumlarda ortaya çıkar.
- Buna karşılık algoritmalar, matematik terimlerle kurulan bir problemin çözümü için prosedürlerdir.
- Değişik algoritmalar arasında bazı temel farklılıklar vardır.

Algoritma Tipleri

- Algoritmalar, gerek yapıları, gerekse çalışma durumlarına göre farklılıklar gösterirler ve şekildeki gibi altı tipe ayrılırlar.

Algoritma Tipleri



Algoritma Tipleri

a) Direkt (Ardışık Olmayan) Algoritmalar

İterasyonlarda çalışmayan algoritmalar, direkt algoritmalar olarak adlandırılır. $y= ax^b$, $(a>0)$ polinomunun türevi $y' = abx^{b-1}$ bunun basit bir örneğidir.

b) Ardışık Algoritmalar

Belirli alt prosedürlerin pek çok kez tekrarlandığı algoritmalardır ve ardışık olarak çalışırlar. Algoritmaların çoğu ardışık olarak çalışır.

Algoritma Tipleri

- c) Yakınsak Algoritmalar
- Aranılan çözüme doğru yakınsayan ardışık algoritmalardır.
- d) Yaklaşık Algoritmalar
- Bazı yakınsak algoritmalar kesin çözümü elde edemezler, fakat bu çözüme yaklaşık bir değeri kesin çözüm alırlar. Yaklaşık algoritmalar sonlu değillerdir; fakat her bir ileri iterasyon onları kesin çözüme biraz daha yaklaştırır. Yaklaşık algoritmalara Değişken Kesen Metodu, Arama Teknikleri vb. çok bilinen birkaç örnek verilebilir.

Algoritma Tipleri

- e) Sonlu Algoritmalar

- Bu algoritmalar, iterasyonların sonlu bir sayısında kesin çözümü garanti eden yakınsak algoritmalardır ve kendi arasında yol yapılı ve ağaç yapılı olmak üzere ikiye ayrılırlar.

Algoritma Tipleri

- 1. Yol Yapılı Algoritmalar:

- Sonlu algoritmaların pekçoğu yol yapısına sahiptir; bu yol yapısında bir iterasyon bir önceki iterasyonu iterasyon dizilerinde farklı dallar üretmeksizin takip eder.

- Böyle algoritmaların örnekleri, Tamsayı Programlamada Kesme Düzlem Algoritmalarının pekçoğu, Şebekelerde Maksimum Akış Algoritmaları, pekçok En Kısa Yol Algoritmaları ve Lineer Programlamanın Simpleks Algoritmaları ve herhangi pivot tekniği ile matris tersleridir.

Algoritma Tipleri

- 2. Ağaç Yapılı Algoritmalar:

- Diğer sonlu algoritmalarda iterasyon dizileri, pek çok paralel dalları içeren bir ağaç şeklindedir. Bir çok ağaç arama algoritmaları bu sınıfa aittir.

- Bu arama algoritmalarının bazıları, Dinamik Programlama, Dal ve Sınır, Sınırlı Sayım, Kapalı Sayım, Dal ve Çıkarma, Dal ve Atma vb. olarak sıralanabilir. Yaklaşık algoritmalar bu yol yapısına doğru yönelir. Diğer yandan heuristikler ise bir yol yapısı ile son bulabilen indirgenmiş bir ağaç yapısı olarak gözönüne alınabilir.

Algoritmanın Yapısı

- Bir problemi çözmek için adım adım uygulanacak bir prosedür olarak tanımlanan algoritmanın tipik adımları;

  - i. Atama adımları, (Bir değişkene bazı değerlerin atanması gibi)
  - ii. Aritmetik adımlar, (Toplama, bölme, çıkarma, çarpma gibi)
  - iii. Mantıki adımlardır. (İki sayının karşılaştırılması gibi)

Algoritmanın Yapısı

- Genel yapısı bu şekilde kısaca özetlenen algoritmalar, belirli bir hata kabulü içinde yaklaşık cevap verirler; uygun programlama dili seçildiğinde genellikle hızlı çalışırlar.

- Bir algoritmanın sahip olduğu adımların sayısı (ki bu, algoritmanın gereksinimi olan zamanı büyük ölçüde belirler) problemin bir örneğinden diğerine farklılık gösterir.

- Problemin bazı “iyi” örnekleri, bu algoritma yardımıyla hızlı olarak çözülebilirse de, problemin bazı “kötü” örneklerini bu algoritma yardımıyla çözmek çok uzun zaman alabilir.

- Bu, ilgili algoritmanın performansına bağlı bir faktördür.

Algoritmaların Dili

- Kodlama :
- 1)Procedure = Bir algoritmanın kodlanmasına başlanan ilk ifadedir.
- Bu ifadede algoritmanın ad Örnek: Procedure max(L = list of integers)
- Burada algoritmanın adı max iken tamsayıların L listesinin maksimumunu bulur

Algoritmaların Dili

- 2) Assignments=Atamalar ve ifadelerin diğer tipleri
- Assigments ifadesi değişkenlere değer atamada kullanılır.
- Bu ifadede sol taraf değerin adını alırken sağ taraf ise prosedürlerle tanımlanan fonksiyonları, değerleri atanan değişkenleri, sabitleri içeren bir ifade ya da deyimdir.
- Sağ tarafta ayrıca aritmetik işlemlerin herhangi biride bulunabilir.

Algoritmaların Dili

: = atamalar için semboldür.

Böylece ;

Variable : = expression

Max: = a (a'nın değeri max değişkenine atanır.)

Ayrıca, x : = Largest integer in the list L şeklinde bir ifade de kullanılır.

x ' i L'de en büyük tamsayı olarak atar.

Güncel bir programlama diline bu ifadeyi dönüştürmek için birden fazla ifade kullanmalıyız . Bunun için ; interchange a and b kullanılır. Karmaşık prosedürler için ifade blokları kullanılır . Bu begin ile başlar ve end ile sona erer.

Algoritmaların Dili

```
Begin
      Statement 1
      Statement 2

     ..............
      Statement n
end.
```

Algoritmaların Dili

Şartlı Yapılar

1) **if condition then statement**

veya

if condition then

begin

blok of statements

end

Eğer durum doğru ise verilen ifade gerçekleştirilir

2) if condition then statement1 else statement 2

Genel form: İf condition 1 then statement 1 else if condition 2 then statement 2 else if condition 3 then Statement 3……..

Else if condition n then statement n else statement n+1

Eğer durum 1 doğruysa ifade 1 gerçekleştirilir v e programdan çıkılır .Eğer, durum1 yanlışsa program durum2 'nin doğruluğunu kontrol eder .Eğer durum2 doğruysa ifade2

gerçekleştirilir . Böylece devam edilir. Sonuç olarak ,eğer durum1 ile durum – n'nin hiçbiri doğru değilse (n+1).ifade yerine getirilir.

Algoritmaların Dili

Döngü Yapıları

1) for

2) while

1) for variable : = initial value to final value

statement

veya

for variable : = initial value to final value

begin

block of statements

end.

Algoritmaların Dili

- Eğer başlangıç değer sonuç değerden küçük yada eşitse değişken olarak başlangıç değer atanır ve sonuç değer değişken atanana kadar bu döngü gerçekleştirilir.

- Eğer, başlangıç değer sonuç değeri aşarsa döngü olmaz.

- Örneğin;

Algoritmaların Dili

```
sum : = 0
  for i : = 1 to  n
    sum : = sum+i
```

2) while condition statement

veya while condition

begin

block of statements

end şeklindedir. Burada durum doğruysa ifade gerçekleştirilir ve durum yanlış olana kadar bu döngü sürdürülür .

Algoritmaların Karmaşıklığı

- Bir algoritmanın karmaşıklığı, bu algoritmanın çalışma zamanı ile ilgili bir kavramdır.
- Bir algoritmanın çalışma zamanı olarak da isimlendirilen algoritmanın gereksinimi olan zaman, verinin hem yapısına hem de boyutuna bağlıdır.
- Büyük problemler çok daha fazla çözüm zamanı gerektirirken; verideki farklılıklara bağlı olarak aynı boyuttaki değişik problemler belirgin olarak farklı çözüm zamanları gerektirirler.
- Bir algoritmanın karmaşıklık fonksiyonu, problem uzayı boyutunun bir fonksiyonudur ve verilen boyutun zamanının herhangi bir problem örneğini çözmek için algoritma tarafından ihtiyaç duyulan en geniş zamanı belirtir.
- Bir başka deyişle, zaman karmaşıklığı fonksiyonu, problemin boyutu arttıkça çözüm zamanının gelişme hızını ölçer.

Algoritmaların Karmaşıklığı

- Burada dikkat edilmesi gereken konu, zaman karmaşıklığı fonksiyonunun, verilen bir problem uzayı boyutunda herhangi bir problem örneğini çözmek için gereken çalışma zamanını, çözüm için gerekli en büyük çalışma zamanını ölçerek hesapladığıdır.

- Karmaşıklık fonksiyonu, algoritmanın performansının ölçümünde, problemin giriş verisinin uygun ölçülmesine bağlı olarak bir performans garantisi sağlar.

- Bu nedenle zaman karmaşıklığı fonksiyonu aynı zamanda bir algoritmanın en kötü durum karmaşıklığı veya sadece karmaşıklığı olarak bilinir.

Algoritmaların Karmaşıklığı

- Polinom ve Üstel Zaman Algoritmaları

Bir algoritmanın etkinliği, en kötü durumda algoritma için gereken adımların sayısını sayarak ve bunu problem uzayı boyutu olan n'nin bir fonksiyonu olarak, n büyüdükçe çalışma zamanının davranışını saptamaktır. Algoritma için gereken adımların sayısı n'nin bir polinom fonksiyonu olduğunda, "*O*" sembolu kullanılır ve algoritmanın çalışma zamanı $O(n^k)$'olarak gösterilir. Burada, *k*, büyük *n*'ler için diğer terimler anlamsız olduğundan, polinomun en büyük derecesidir.

Bir başka deyişle, eğer bir algoritmanın zaman karmaşıklığı fonksiyonu $O(n^k), (k \in N)$ (fonksiyon *k*.nci derecedendir) ise bu algoritmaya **Polinom Zaman Algoritması** denir ve **algoritma** $\boldsymbol{O(n^k)}$'dır denir. Örneğin, $k=1$ ise algoritmaya *Lineer*, $k=2$ ise algoritmaya *karesel* algoritma denir.

Algoritmaların Karmaşıklığı

Giriş verisi bir polinom fonksiyon ile sınırlandırılan bir çalışma zamanına sahip olan bir algoritma genellikle *"iyi"* olarak adlandırılır. Bu algoritma ile çözülen problemlere de *"kolay"* denir. Diğer yandan, çalışma zamanı bazı $c>1$ için en az c^n kadar hızla artan bir algoritmaya "**Üstel Zaman Algoritması**" denir.

Nlogn gibi ne polinom ne de üstel bir fonksiyon ile sınırlandırılmamış yarı üstel fonksiyonlar olmasına karşılık en kötü durum kriteri bir polinom fonksiyon ile sınırlandırılmamış herhangi bir algoritmaya kolaylık için "*Üstel Zaman Algoritması*" denir. **O(2n),O(n!), O(n^{logn})** üstel zaman algoritmalarına örnek olarak gösterilebilir. Polinom ve üstel zaman algoritmaları arasındaki farklılık, saniyede 10^6 işlem

Algoritmaların Karmaşıklığı

düzenleme kapasitesine sahip bir bilgisayarın farklı algoritmaları icra etme zamanı ile hesaplama gücü arttığında en büyük çözülebilir problemin problem uzayı boyutunun karşılaştırılmasıyla ortaya çıkar.

Aşağıdaki tabloda bazı polinom ve üstel fonksiyonların problem uzayının boyutunun n'ye bağlı büyüme oranları verilmiştir.

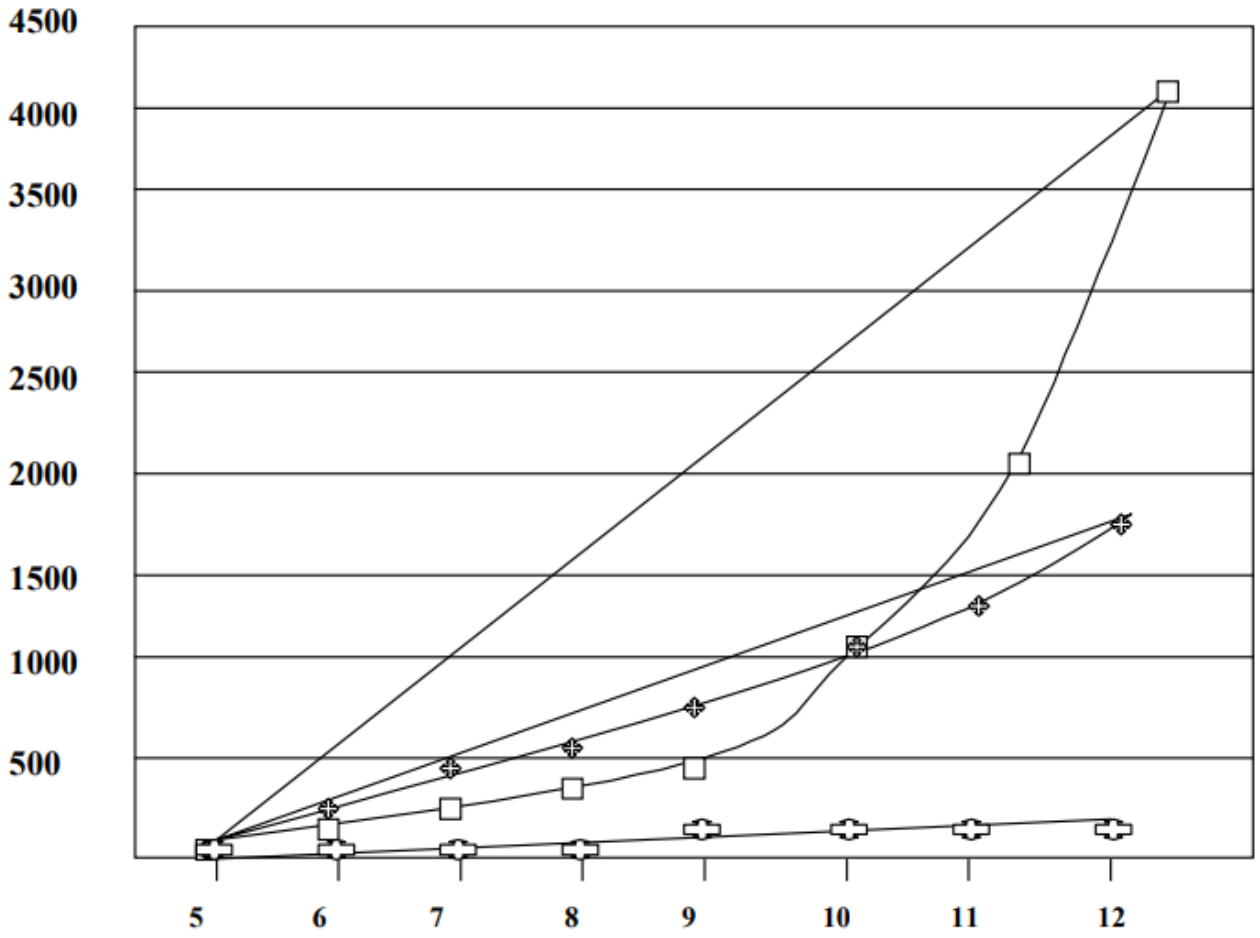
Algoritmaların Karmaşıklığı

Tablo 1.1. Bazı üstel ve polinom fonksiyonların büyüme oranları

n	log n	$n^{0.5}$	N^2	n^3	2^n	n!
10	332	316	10^2	10^3	10^3	$3,6 \times 10^6$
100	664	10	10^4	10^6	$1,27 \times 10^{30}$	$9,33 \times 10^{157}$
1000	997	31.62	10^6	10^9	$1,07 \times 10^{301}$	$4,02 \times 10^{2,567}$
10000	1329	100.00	10^8	10^{12}	$0,99 \times 10^{3010}$	$2,85 \times 10^{35,659}$

Algoritmaların Karmaşıklığı

Aşağıdaki şekilde de problem uzayının boyutu olan n'ye göre f(n) hesaplama adımlarının sayısı, $O(n^2), O(n^3)$ ve $O(2^n)$ algoritmalar için gösterilmiştir.



Algoritmaların Karmaşıklığı

- Gerek Tablo 1.1 gerekse Şekil 1.1'den anlaşılacağı gibi genelde polinom zaman algoritmaları üstel zaman algoritmalarına göre daha küçük derecelere sahip olduklarından daha iyi düzenlenirler.

Algoritmaların Performans Ölçümü

- Bir algoritmanın performansı, o algoritmanın performansının nasıl ölçüleceği sorusundan doğar.

- Bu sorun, bir problemin çözümü için ortaya konulan algoritmalar arasından “en iyi” algoritmayı seçme ile ortaya çıkar. Bir algoritmanın performans ölçümü için üç temel yaklaşım yaygın olarak kullanılır:

- Bunlar deneysel analiz, olasılık (ortalama durum) analizi ve en kötü durum analizidir. Şimdi bu analizleri kısaca açıklayalım:

Algoritmaların Performans Ölçümü

- Deneysel Analiz

- Deneysel analiz, örnek problemlerde denenmiş bir algoritmada hesaplama deneyiminde dayanır.
- Bu analizin amacı, pratikte algoritmanın nasıl davrandığını tahmin etmektir.
- Bu analizde, algoritma için bir bilgisayar programı yazılır ve problem örneklerinin bazı sınıfları üzerinde programın performansı test edilir.

Algoritmaların Performans Ölçümü

- Bu nedenle deneysel analiz, araştırmacı ve uygulamacıların en çok güvendiği analizlerden biridir ve bilimsel yaklaşımdan çok, uygulamaya yöneliktir.

- Deneysel analizin başlıca dezavantajları şunlardır:

Algoritmaların Performans Ölçümü

- i. Bir algoritmanın performansının, programı yazan programcının tekniği kadar kullanılan bilgisayara, derleyiciye ve programlama diline bağlı olması.

- ii. Bu analizin çok zaman alması ve düzenlenmesinin pahalıya mal olması

- iii. Algoritmaların karşılaştırılması; farklı algoritmaların problem örneklerinin farklı sınırlarını daha iyi düzenlemesi ve farklı deneysel çalışmaların birbirini tutmayan sonuçlar vermesi anlamında sıkça kesinlik taşımamasıdır.

- Bu analizin başlıca avantajı, güncel problemler için kesin ve geçerli olmasıdır .

Algoritmaların Performans Ölçümü

- Olasılık (Ortalama Durum) Analizi

- Bu analiz son zamanlarda yoğun bir şekilde kullanılmaya başlanmıştır. Bu analizin amacı, algoritmanın alması beklenen adımlarının sayısını tahmin etmektir.

- Olasılık analizinde, problem örnekleri için bir olasılık dağılımı seçilir ve algoritma için beklenen asimptotik çalışma zamanlarını üreten istatistik analiz kullanılır.

Algoritmaların Performans Ölçümü

- Olasılık analizinin başlıca dezavantajları şunlardır:
  - i. Belirli analizin, problem örneklerini temsil etmek için seçilmiş olasılık dağılımına kesin olarak bağlı olması ve olasılık dağılımındaki farklı seçimlerin algoritmaların yapısından kaynaklanan avantajlar nedeni ile ilgili farklı değerlendirmelere yol açabilmesi.
  - ii. Pratikte karşılaşılan problemler için uygun olasılık dağılımlarını belirlemenin genellikle zor olması.
  - iii. Olasılık analizinin, algoritmanın en basit tipini değerlendirmek için bile oldukça yoğun matematik alt yapı gerektirmesi ve bu analizin tipik olarak çok kompleks algoritmalar için başarılmasının oldukça zor olması.

Algoritmaların Performans Ölçümü

- Bir algoritmanın performansının önceden tahmini, o algoritmanın olasılık analizine dayanmasına rağmen, analistin problem örneklerinin büyük bir çoğunluğunu çözmesini gerektirmesi ve sonuçların dağılımı hakkında bilgi sağlamaması, bu analizin rağbet görmemesine yol açar.

Algoritmaların Performans Ölçümü

- En Kötü Durum Analizi

- En kötü durum analizi, özel bir H heuristiği ile bir P probleminin örneklerine uygulandığında ortaya çıkan optimalden sapma ile ilgili analizdir. Bu analiz, verilen bir algoritmanın herhangi bir problem örneği üzerinde alabileceği adımların sayısına bir üst sınır getirir.

- En kötü durum analizinde diğer iki analizde bulunan dezavantajların çoğu yoktur.

- Bu analizin iki dezavantajı, bir algoritmanın performansını belirlemek için pratikte son derece nadiren ortaya çıkan anlamsız örneklere izin vermesi, ve bir algoritmanın en kötü durum ortalama performansının önceden bilinmemesidir.

- En kötü durum analizi, hesaplama çevresinden bağımsızdır ve düzenlenmesi diğer analizlere nisbeten kolaydır. Ayrıca bu analiz, bir algoritmanın adımları (çalışma zamanı) üzerinde bir üst sınır sağlar.

Algoritmaların Performans Ölçümü

Bu nedenlerden dolayı bir algoritmanın performans ölçümü için bu üç analizden bilimsel literatürde en popüler olanı en kötü durum analizidir. Performans garantisi, en kötü durum oranı olan r ile ifade edilir. Buna göre, bazı $r>1$ için $C_h(I)$ ve $C(I)$ sırasıyla bir $I \in P$ minimizasyon problemi örneğinin heuristiği ve optimal çözümü olmak üzere;

$$C_h(I) = r \cdot C(I) \qquad (\exists\, r>1)$$

ile ifade edilir. Ayrıca heuristiğin performansı en kötü durum bağıl hatası $\varepsilon=r-1$ ile degerlendirilir. ε ve r ikilisi çok kullanılır ve duruma göre bunlardan önemli olanı seçilir [4].

Algoritmaların Performans Ölçümü

"O", "Ω", "Φ" Sembolleri

O, Ω, Φ sembolleri, araştırmacıların algoritmaların analizinde kullandığı sembollerdir. Şimdi sırasıyla bunları tanıyalım:

"O" : Eğer c ve n_0 sayıları için, bir algoritmanın gereksinimi olan zaman en fazla her $n \geq n_0$ için $c.f(n)$ ise bu algoritma $O(f(n))$ çalışma zamanına sahiptir denir.

Yani, *O* sembolü algoritmanın performansı üzerinde bir üst sınırı belirtir. Problem uzayı boyutu olan n'nin yeterince büyük değerleri için algoritmanın çalışma zamanında bir üst sınır belirlendiğinde $O(f(n))$ karmaşıklık ölçümü çalışma zamanının asimptotik bir büyüme oranına sahip olmasına yol açar .

"Ω" : Ω sembolü, algoritmanın çalışma zamanı üzerinde bir alt sınırı belirtir. Eğer bazı c' ve n_0 sayıları ve her $n \geq n_0$ için bir algoritmanın çalışma zamanı bazı problem örneklerinde en az $c'.f(n)$ ise bu algoritma $\Omega(f(n))$ olarak isimlendirilir. *"O"* ve *"Ω"* sembollerini kısaca şu şekilde özetlenebilir :

Algoritmaların Performans Ölçümü

Eğer bir algoritmanın çalışma zamanı, $O(f(n))$ ise sabit bir c sayısı için problemin herbir örneği en çok $c.f(n)$ zamanında çalışır. Benzer şekilde, eğer bir algoritma $\Omega(f(n))$ zamanında çalışırsa, problemin bazı örnekleri c' bir sabit olmak üzere en az $c'.f(n)$ zamanında çalışır.

"Φ" : Φ sembolu bir algoritmanın performansı üzerinde hem alt hem de üst sınırı belirtir. Yani eğer bir algoritma hem $O(f(n))$ hem de $\Omega(f(n))$ ise bu algoritma $\Phi(f(n))$ olarak isimlendirilir.

Bir algoritmanın performansının alt veya üst sınırı algoritmanın çalışma zamanında bir üst sınır veya bir alt sınır belirleme anlamındadır .

Algoritmaların Performans Ölçümü

- Fonksiyonların Büyüme Oranları

- Bu bölümde algoritmaların performans ölçüsünde bize yardımcı olacak fonksiyonların büyüme oranları, modüler aritmetik ve buna ait temel kavramlar incelenmiştir.

- Önceki bölümde kısaca bahsettiğimiz “O” notasyonu fonksiyonlarının gelişiminde en yaygın kullanılan notasyon olduğundan burada bu notasyonu daha iyi anlamak için tanım ve örnekleri ele alınacaktır.

- “O” sembolü ilk olarak Alman 82 matematikçi Paul Gustav Heinrich Bachmann (1837-120) tarafından 1892 de sayı teorisi üzerine önemli bir kitapta kullanılmış olmasına rağmen çalışmasının her aşamasında bu sembolü kullanan Edmund Landau (1877-138) sayesinde “O”sembolü “Landau Sembolü” olarak da adlandırılır.

Algoritmaların Performans Ölçümü

Tanım 1.1: f ve g fonksiyonları reel sayılardan reel sayılara veya tam sayılardan reel sayılara tanımlı iki fonksiyon olsun. C ve k sabitleri için ;

$$|f(x)| \leq C\,|g(x)| \qquad (x > k)$$

sağlanıyorsa ; $\boldsymbol{f(x) = O\,(g\,(x))}$ denir.

Örnek 1.1 : $f(x) = x^2 + 2x + 1$ için $O\,(x^2)$ olduğunu gösterelim.

Çözüm 1.1 : $x > 1$ için ;

$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$ olduğundan $C = 4$ ve $k = 1$ için $f\,(x) = O\,(x^2)$ olur.

$x > 2$ için ;

$2x \leq x^2$ olduğundan

$0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2$ burada da $C = 3$ ve $k = 2$ için $f\,(x) = O\,(x^2)$ olduğu görülür.

Böylece , $f(x) = x^2 + 2x + 1$ ve $g(x) = x^2$ için $f(x) = O\,(g(x))$ dir.

Genelleyecek olursak geçişme özelliğinden aşağıdaki sonucu yazabiliriz.

$f(x) = O\,(g\,(x))$ ve x' in yeterince büyük değerleri için $h(x)$ fonksiyonu $g(x)$'den daha büyük mutlak değerlere sahip olsun.

Algoritmaların Performans Ölçümü

Bu durumda ;

$|f(x)| \leq C|g(x)|$ $\quad x > k$,

ve eğer bütün $x > k$ 'lar için $|h(x)| > |g(x)|$ oluyorsa;

$|f(x)| \leq C|h(x)|$ $\quad x > k$ olur.Bu nedenle ;

$f(x) \leq O(h(x))$ elde edilir.

Algoritmaların Performans Ölçümü

- Fonksiyonların Kombinasyonunun Büyüme Oranı

Algoritmaların çoğu iki veya daha fazla alt prosedürden oluşur.Böyle bir algoritmayı kullanarak belirli ölçüdeki bir problemi çözmek için bilgisayar tarafından kullanılan adımların sayısı bu alt prosedürler tarafından kullanılan adımların sayıları toplamına eşittir. Kullanılan adımların sayısının tahmini için 'O' notasyonu kullanılır.Bunun için iki fonksiyonun toplamı ve çarpımı için 'O' notasyonu nasıl bulunur onu inceleyelim.

$f_1(x) = O(g_1(x))$ ve $f_2(x) = O(g_2(x))$ olsun. C_1, C_2 , k_1 ve k_2 sabitleri için;

Algoritmaların Performans Ölçümü

$$|f_1(x)| \le C_1 |g_1(x)| \qquad x > k_1 ,$$
$$|f_2(x)| \le C_2 |g_2(x)| \qquad x > k_2 ,$$

olur.Bu iki fonksiyonun toplamı ise;

$$|(f_1 + f_2)(x)| = |f_1(x) + f_2(x)|$$
$$\le |f_1(x) + f_2(x)| \quad \text{(üçgen eşitsizliği } |a + b| \le |a| + |b|)$$

$x > k_1$, $x > k_2$ durumu için;

$$|f_1(x)| + |f_2(x)| < C_1 |g_1(x)| + C_2 |g_2(x)|$$
$$\le C_1 |g(x)| + C_2 |g(x)|$$
$$= (C_1 + C_2) |g(x)|$$

Algoritmaların Performans Ölçümü

$$= C\,|g(x)|\quad ,$$

burada, $C = C_1 + C_2$ ve $g(x) = \max(|g_1(x)|, |g_2(x)|)$ alındığından

$k=\max(k_1,k_2)$ iken

$$|(f_1+f_2)(x)| \le C\,|g(x)| \qquad x > k$$

olarak elde edilir.

Teorem.1.1.

$f_1(x) = O(g_1(x))$ ve $f_2(x) = O(g_2(x))$ olsun.Bu durumda;

$(f_1+f_2)(x) = O(\max(g_1(x)), g_2(x)))$ .Burada, $\max(g(x), g(x)) = g(x)$ olduğundan

$(f_1+f_2)(x)$ yine $O(g(x))$'dir.

Sonuç.1.1. $f_1(x) = O(g(x))$ ve $f_2(x) = O(g(x))$ olduğunu varsayalım.Böylece;

$(f_1+f_2)(x) = O(g(x))$'dir.