Hesapsal Yük Teorisi (computational complexity)

Hesapsal Yük Teorisi algoritmaların yer ve zaman gibi kaynakları ne kadar kullandığından hareketle bu algoritmaları kategorize etmeye uğraşır. Diğer bazı amaçlar, hangi algoritmaların çözümsüz olacağı, hangi algoritmaların çok zaman alacak olsa bile, eninde sonunda bir sonuca varabileceği gibi konulardır.

Algoritma türlerinin arasındaki benzerlikleri bulmak, yük teorisinde önemli bir yer tutar. Mühendisler için de bu kategorizasyonun direk bir etkisi olmaktadır. Mesela problem XYZ için bir algoritma yazmanız gerektiğini düşünün, ve aynı gün kuramsal bir bilgisayar bilim makâlesinde, sizin probleminizin diğer bir "ABC problemi" ile tıpatıp aynı olduğunu okudunuz. Bu makâleye göre, ABC probleminin kabakuvvet çözümünün "yavaş" olduğunu belirtilmiş olabilir, ve hızlı çözümün imkansız olduğu da ispat edilmiştir (mesela).

Bu bilgiden hareketle, siz üzerinde uğraştığınız algoritmanın hızlı cözümünün olamayacağını daha baştan anlamış oluyorsunuz. Bu sayede gereksiz zaman harcamayarak, ya problemi basitleştirmeye, ya da akıllı tahmin (heuristic) ekleyerek çözümü biraz olsun hızlandırmaya çalışabilirsiniz. Yazılımbilimde algoritmalar arasındaki benzerlik, çok sıkı bir ilişkidir, ve bu yüzden algoritma kategorilerinin hangi problemleri içerdiği çok büyük önem taşır.

Diğer Konular

Yazı dizimizin sonunda, hızlı ve ya yavaş algoritma sözünün teorik olarak ne ifade ettiğini, hesapsal yük teorisinin baş araçlarından olan indirgeme (reduction) tekniğinin ne olduğunu göreceğiz. Problemler arasındaki benzerliğin, birini ötekini indirgemek ile mümkün olduğunu göstermeye uğraşacağız.

Turing makinalarını, baştan planlı (deterministic), baştan plansız (nondeterministic) şekillerini de yazılarımızda görmeniz mümkün olacak.

Algoritma

Bilgisayarcılar için algoritma çok tanıdık bir kelimedir. Başı sonu belli, her muhtemel seçenek için önceden belirlenmiş bir kod parçasının devreye girdiği bir veri ve eylemler dizisidir algoritma.

Bu algoritmayı yazarken içinde bulunduğumuz evren, değişkenler, girdi, çıktı aletleri, eğer/eylem çiftleri, gibi kavramların olduğu bir evrendir. Bu dili işleten makinayı bir soyut makina olarak addedelim. Fakat göreceğiz ki, halâ

teorik iş yapmamız için bu makina yeteri kadar basit değildir. Kullandığımız 'esnek' dili destekleyen makinamız oldukça çetrefilli hâldedir. Ayrıca, dili değiştirirsek (Java yerine LISP gibi) makının da değişmesi gerekecektir, bütün dilleri temsil edebilen bir makina bulamaz mıyız? Teorik iş yapabilmemiz için böyle evrensel bir makinaya ihtiyacımız var.

Dili basitleştirelim. Direk erişimli (random access) belleği olan, komutları ve verisi aynı gözüken bir makina yapalım ve onun kullandığı dili tasarlayalım. (Bu makina günümüzde kullanılan bilgisayardır).

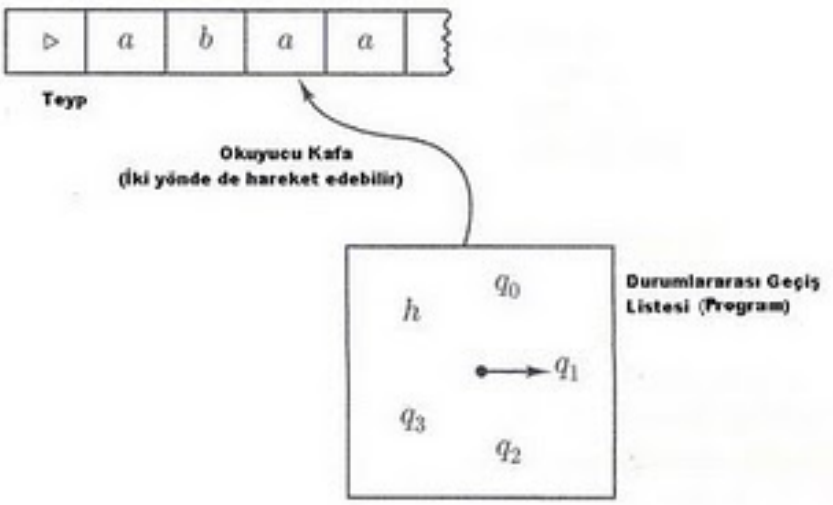
Peki bu makina daha da basit olamaz mı?

Olur. Tek bir teyp üzerinde girdisini tutan, her an, önceden belirli ve sayılı konum/durum içinde olabilen, komutlarını, durumdan/duruma geçiş listesi olarak tutan bir makina düşünelim.

Öyle gözüküyor ki, artık bilgisayar işleminin ruhuna indik. Bundan daha basit bir makina tasarlamamız mümkün gözükmüyor. Durum, geçiş, ve teyp kavramlarını kullanarak her türlü bilgisayar hesabını temsil edebileceğimizi düşünürsek (bunun ispatları yapılmıştır), en basit temsil şeklinde varmış olduğumuzu görüyoruz.

İşte bu makina, Turing makinası olarak bilinir.

Turing Makinası



Formel olarak, Turing makinası M şu dörtlüyü içerir:

$M = (K, \Sigma, \delta, s)$

K = makina durumu (state)

δ = gecis fonksiyonu

Σ = teyp alfabesi

s: teyp verisi

Dörtlü içindeki bütün terimler birer kümedir. K terimi, M makinasının içerdiği durumların kümesi, delta bütün geçişlerin listesi, sigma, alfabe olduğu için teypin kullandığı harflerin kümesi ve s, giriş için M makinasına verilen harflerin kümesidir.

Şimdi programa dönelim: Geçiş fonksiyonu olan delta, yani program, K x Sigma ile, (K U dur, "evet", "hayır") x Sigma x Sol, Sağ, Hareketsiz küme üyelerini birbirine eşler. Yani geçiş fonksiyonu, durum+teyp sembolu ikililerini, durum+teyp sembolu+teyp hareketi üçlülerine eşlemektedir.

Not: 'x' operasyonu, iki küme arasında kartezyen eşleme yapmaktadır. Yâni, A x B, A kümesinin her elemanı ile B kümesinin her elemanını eşleyerek, —A—*—B— sayıda yeni bir küme oluşturur. SQL dilini bilenler JOIN komutu ile bağlantı kurabilirler)

Turing makinasının işlevi, makinanın o anda gördüğü, kafasının okuduğu sembol, ve o an içinde olunan duruma göre başka bir duruma geçmek, ve (gerekiyorsa) teype yeni bir harf yazmak, sonra da teyp kafasını gene programa göre sağa ya da sola hareket ettirmekten ibarettir. Teyp kafasını hareketsiz bırakmakta mümkündür.

Bu kadar basit temel işlemlere dayanan bir modelin dünyadaki bütün algoritmaları temsil edebilmesi ilginç değil mi?

Örnek Turing makinası olarak, aşağıda teyp üzerinden verilen bir metnin palindrom olup olmadığını anlayabilen bir Turing Makinayı görebilirsiniz. Bu program (makina), eğer metin palindrom ise "evet" cevabı verecek, değil ise "hayır" cevabı verecektir. Palindrom metni, "arabaabara" gibi, içinde "araba" kelimesinin ters yüz edilerek yanyana konulduğu metne verilen isimdir. Palindrom tanımak çok kolay olmayıp çok zor da olmayan bir örnek olduğu için hesapsal yük teorisi kitaplarında oldukça kullanılmaktadır.

$p \in K,\ \sigma \in \Sigma$		$\delta(p,\sigma)$
s	0	$(q_0, \triangleright, \rightarrow)$
s	1	$(q_1, \triangleright, \rightarrow)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	$\sqcup$	('evet', $\sqcup$, $-$)
q_0	0	$(q_0, 0, \rightarrow)$
q_0	1	$(q_0, 1, \rightarrow)$
q_0	$\sqcup$	$(q_0', \sqcup, \leftarrow)$
q_1	0	$(q_1, 0, \rightarrow)$
q_1	1	$(q_1, 1, \rightarrow)$
q_1	$\sqcup$	$(q_1', \sqcup, \leftarrow)$

$p \in K,\ \sigma \in \Sigma$		$\delta(p,\sigma)$
q_0'	0	$(q, \sqcup, \leftarrow)$
q_0'	1	('hayır', 1, $-$)
q_0'	$\triangleright$	('evet', $\sqcup$, $\rightarrow$)
q_1'	0	('hayır', 1, $-$)
q_1'	1	$(q, \sqcup, \leftarrow)$
q_1'	$\triangleright$	('evet', $\triangleright$, $\rightarrow$)
q	0	$(q, 0, \leftarrow)$
q	1	$(q, 1, \leftarrow)$
q	$\triangleright$	$(s, \triangleright, \rightarrow)$

Church-Turing Tezi

Araştırmacılar uzun süre Turing makinasından daha basit bir model bulmaya uğraştılar, ve bu uğraşıda başarısız oldular.

Daha sonra araştırmacılar, kaç değişik makina modelinin mevcut olabileceğini anlamak için, en basit Turing makinasının çözemeyeceği problemleri çözecek makinalar tasarlamaya da uğraştılar. Mesela RAM, birden fazla teyp, vs. gibi ekler koyarak, basit modeli güçlendirmeye çabaladılar. Eğer en basit modelin çözemeyeceği bir problemi çözen bir model bulsalardı, bu model yeni ve alternatif bir model olabilirdi. Yeni modelin değişik olup olmadığını nasıl anlamak için, kuramcılar indirgeme denen bir tekniği kullandılar. İndirgeme, yeni modelle kurulmuş olan makinayı, eski modelle kurulmuş makina aracılığı ile, yani onun dili ile, simule etmektir.

Bu simulasyonun 'dönüşüm' denen aşamasında, simule edilen makinanın girdisi, ötekine çok hızlı bir şekilde dönüştürülür. Hemen ardından simule eden makinaya girdi olarak verilir. Çıktı da aynı şekilde dönüştürülür.

Eğer çok hızlı (polinom zamanlı) olarak dönüşümü yapabildiysek, ve simule de çalışır ise, indirgeme başarılı olmuş demektir.

Fakat, görülmüştür ki, envai türden ekler ile güçlenen her 'sözde yeni' model en basit Turing makinasına indirgenebilmiştir. Demek ki bu 'yeni' modeller gerçekten yeni model değillerdi, ve işte bu bu sayede bilgisayarlar için en

basit Turing makinasından alternatif bir model olamacağı kanıtlanmış oldu.

Bütün bu bulgulara dayanarak Church ve Turing şu tezi kabul etmeye karar verdiler.

"Bir algoritma ile, (bütün girdilerine "evet" ve "hayır" cevabı verebilen) bir Turing makinası tamamen aynıdır. Birbirleri arasında direk ilişki vardır. "

Yani, algoritma denen soyut kavram, en basit Turing makinası üzerinde yazılan bir program demektir, bütün teorik hesaplar ve kuramlar bu en basit makina üzerinden yapılabilir.

Bu ortak bilgisayar kavramında fikirbirliğine varılmasının ne kadar önemli olduğunu vurgulamak istiyorum. Teorik dünyada, 'bilgisayar' denince, formel bir kavram akla gelmelidir. En basit makinalar arasında en güçlüsü seçilerek, bu makinayı baz alan kuramların da aynı şekilde basit olması sağlanmıştır. Basitlik, bilim dünyasında önemli yer tutar.

"Her girdiye evet ya da hayır cevabı veren" makinaların özellikle belirtilmesi ilginçtir. Bunun sebebi şudur; Her Turing makınasının (yani programın) işleyişini biterek durması garanti değildir. Sonsuz döngüye giren programları hepimiz biliyoruz. Eğer evrendeki her Turing makinasını 11001010... gibi bir ikili düzen kodu ile belirtiliyorsak, bu makinalardan her biri durup, "evet" ya da "hayır" cevabı veriyor olamaz (bu söylemin ispatını Algoritma Yuku konulu yazida görebilirsiniz).

Church-Turing tezi, duran ve "evet" ya da "hayır" cevabı veren makinaların bir algoritma ile eşgörülmesini belirtmiştir.

Ek olarak belirtmek gerekir ki, Church-Turing tezi sadece bir tezdir, yani bir önkabuldür. Aynen matematikteki bir aksiyom gibidir, yani ispatlanmış teori değildir. Bu sebeple doğruluğu veya yanlışlığı ispat edilemez. Geometride nokta, çizgi gibi kavramların en baştan ispatsız olarak kabul edildiği gibi, Church-Turing tezi bir başlangıç önkabuludur. Bu önkabul olmadan geri kalan teorileri bir temele oturtmamız mümkün olmazdı.

Tabii, Church-Turing tezi bir tez olduğuna göre, başka bir tez gelecek olsa değişik bir bilgisayar bilim teorisi kurulabilirdi. Fakat araştırmacılar bunun mümkün olduğunu düşünmüyor.

Sonsuza Giden İkili Sayıların Kümesi

Aşağıda gösterilen küme, sayılamayan sonsuz bir kümedir.

B =
{1101 . . . . . }
{1011 . . . . . }
{1110 . . . . . }
{.100 . . . . . }
{.011 . . . . . }
. . .

Teori: B sayılamayan sonsuzluktadır.

İspat:

B'nin sayılabilir olduğunu farzedelim.

Kullanılan matematiksel teknik: Bir teorinin "karşıtının" doğ ru olduğunu, yani B'nin sayılabilir bir sonsuzluk olduğunu farzedip yola devam eder, ve anlamsız/saçma/absurd bir sonuca varırsak, tersini farzettiğimizi teori doğru demektir. Bu, yanlışın yanlışlığının doğruyu vermesidir bir anlamda.

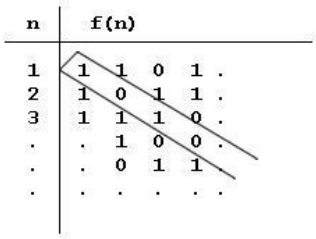
Bu teknik, matematikte "karşıtlık ile ispat etmek" diye bilinir. Teorinin tersini kabul edip yanlış bir sonuca vardıysak, demek ki teori doğrudur .

Devam edelim. B'nin sayılabilir olduğunu farzettiğimize göre, aşağıdaki gibi bir eşleme mümkün olabilir.

n	f(n)
1	1 1 0 1 .
2	1 0 1 1 .
3	1 1 1 0 .
.	. 1 0 0 .
.	. 0 1 1 .
.	. . . . .

Şimdi, doğal sayılar ile olan eşlemeyi yanlış çıkartmak için öyle bir sayı bulacağız ki, hiçbir n ile eşlenemeyecek.

Bu sayı, köşegen üzerindeki sayının ikili aritmetiğe göre tam tersi olsun (köşegen aşağıda gösterilmiştir)



Yani köşegendeki 1010.. yerine, 0101... kullanacağız. Bu sayı, bir n ile eşlenebilir mi?

Hayır! Neden olduğunu görelim. Bu eşlemenin imkansız olmasının sebebi, sol tarafta 1,2,..n diye giderken, n'in karşısındaki f(n)'in (terslik kuralımız yüzünden) n'inci değerinin her zaman gerekenden ters bir değer olacağıdır.

Halbuki, elimizde sonsuz tane 0 ve 1 var, ve elimizdeki 0101.. değerini bir yerlere koyabilmeliydik. Fakat elimizdeki gayet masum ve basit kurala göre bile bunu yapamıyoruz. Demek ki, başta yapılan faraziye, yanlış idi, bu da teorinin doğruluğunu ispatlar. B sayılamayan büyüklükte bir sonsuz kümedir.

Sonsuzluklar Arasındaki Farklar

İki sonsuzluk arasındaki en bariz fark, bir sonsuzluğun sayılabilir ötekinin de sayılamayan türden olduğu zaman ortaya çıkar. Sayılabilen sonsuzlukları tanımlamak için, ünlü matematikçi Kurt Gödel, incelediği kümeyi doğal sayılar ile eşleme tekniğini denedi. Doğal sayılar bildiğimiz gibi 1'den başlayarak sonsuza kadar birer birer artan tam sayıların kümesidir.

Sayılabilir Sonsuzluklar

Zaten herhangi bir şeyi sayarken de yaptığımız bu değil midir? Parmakla gösterip, söyleriz "bir..iki..üç...vs.", ve kullandığımız bütün bu sayılar birer doğal sayıdır. Yani sayarken biz de gösterdiğimiz şeyi, bir doğal sayı ile eşleriz.

Bu eşlemenin geçerli olabilmesi için, en güçlü matematiksel hâlinde olması gerekiyor, yani bize lazım olan birebir ve örten türden bir eşlemedir... A ve B kümesi düşünürsek; Birebir eşleme, iki değişik A elemanının hiçbir zaman aynı B elemanına eşlenmediği zaman ortaya çıkar, örten eşleme ise, B'nin

bütün elemanlarının A'nın bir elemanı ile muhakkak eşlendiği zaman ortaya çıkar.

Bu iki tür eşlemenin olduğu zaman, elimizde tekabül etme (correspondence) ilişkisi çıkar.

Şimdi tekabül tekniği kullanarak örnek kümeleri inceleyelim: Mesela, 2,4,6,... olarak ikişer ikişer artan sayılar kümesi sayılabilir bir sonsuzluk mudur?

Bu soruyu, yeni bilgilerimiz ışığında değiştirerek tekrar soruyoruz; Doğal sayılar ile 2,4,... kümesi arasında tekabül ilişkisi varmı dır?

Ek not: Lise matematiğinden hatırlayacağımız fonksiyon kavramı, aslında bir tekabül ilişkisidir.

Demek ki, doğal sayılar ile 2,4,...N arasında bir fonksiyon bulabilirsek, tekabül ilişkisini kurmuş olacağız, ve 2,4,...N'in sayılabilir bir küme olduğunu ispatlamış olacağız.

Bu fonksiyonu bulmak oldukça basit: f(x) = 2x. Demek ki 2,4,6.. kümesi sayılabilir bir sonsuzluktur.

Sayılamayan Sonsuzluklar

Gerçek sayılar, noktadan sonra kesire devam eden sayılardır, mesela pi sayısı 3.1415926.. ya da 2'nin karekökü 1.4142135... sayıları gerçek sayılardır. Cantor, R kümesinin sayılamaz olduğunu köşegenleştirme (diagonalization) tekniğini kullanarak ispat etmiştir.

Teori: Gerçek sayılar kümesi R (real numbers), sayılamaz bir kümedir.

İspat: R'ın sayılamaz olduğunu ispat etmek için, R ile N (doğal sayılar) arasında tekabül ilişkisi olmadığını ispat etmek zorundayız. İspat, karşıtlık ile ispat etme tekniğini kullanacak. Düşünelim ki, N ile R arasında f denen bir tekabül ilişkisi mümkün. Bizim yapmamız gereken, f'in gerektiği gibi çalışamacağını ispat etmekten ibaret.

F'in doğru bir tekabül ilişkisi olabilmesi için, f bütün N'in elemanlarını, tüm R elemanları ile eşlemelidir. Ama biz öyle bir x bulacağız ki, bu x hiçbir N elemanı ile eşlenemeyecek. Aradığımız karşıtlıkta işte bu x olacak.

Bu x'i arayıp bulamayız tabii, ama inşa edebiliriz.

Şimdi, tekabül ilişkisinin olduğu farzından yola çıkarak, aşağıdaki türden bir

ilişkinin mevcut olduğunu varsayalım.

n	f(n)
1	3.14159....
2	55.555555...
3	0.12345...
4	0.5000000...
..	...

Bu tekabül ilişkisi, f(1) = 3.14159...., f(2) = 55.55555..., f(3) = .. .olarak devam ediyor. Yani, f işlevi 1 sayısını 3.14159 ile eşliyor, 2 sayısını 55.55555 ile eşliyor, vs.

Baştaki farzla ilerleyip geri kalan sonuçları patlatmak için, amacımız f(n)'in üyesi olamayacak bir x bulmak idi. Bunun için şöyle bir x kurgulayabiliriz.

X'in inşa kuralını şöyle saptayalım: X'in 1. basamağındaki sayı, f(1)'in noktadan sonraki 1. basamağındaki sayıdan farklı olsun. Ne olursa olsun (önemli değil) ama farklı olsun. Yukarıdaki f(1) örneğinde bu sayı 1 (3.14159..), o zaman x'in noktadan sonraki 1. sayısı, 1'den farklı olması gerekiyor; mesela, rasgele seçiyoruz, 4.

Aynı şekilde, x'in f(2)'de olamamasını zorlamak için, x'in 2. basmağındaki sayının f(2)'nin 2. basamağındaki sayıdan farklı seçiyoruz. Yani, 5 yerine (55.55555..) diyelim 6.

Gene aynı şekilde, x'in f(3) için, 3 yerine 4 seçebiliriz, vs..

Bu şekilde f(n)'in köşegeni üzerinde devam ederek bir x oluşturmuş oluruz.

n	f(n)
1	3.**1**4159....
2	55.5**5**5555...
3	0.12**3**45...
4	0.500**0**000...
..	...

x = 0.464...

X'in f(n)'in üyesi olamayacağını bu şekilde ispatlamış oluyoruz, çünkü x'in n'inci basamağı, f(n)'in noktadan sonraki n'inci basamağından *her zaman* değişik olacaktır.

Not: Biraz daha görsel olan ispatlar, şunu da ekleyebiliyor: X'i f(n) içine sokuşturmuş olduğumuzu düşünün;

n	f(n)
1	3.**1**4159....
2	55.5**5**5555...
3	0.12**3**45...
4	0.500**0**000...
..	...
..	0.464 ???

Soru işareti yerine hangi sayı gelmelidir? :)

Soru işareti yerine istediğiniz sayıyı koyun, bir taraf o sayının öyle kabul etmekte, x sırası ise ne olursa olsun, o sayı olmasın (!) demektedir. Bu da bir çakışma, uyuşmazlı, absürdlük ve saçmalıktır. Demek ki baştaki faraziyemiş yanlıştır. Demek ki, R kümesi olan f(n), doğal sayılar (n) ile eşlenemiyor; O zaman R sayılamayan büyüklükte sonsuz bir küme olmaktadır.