Bilgisayar Bilimi Köşesi

Chris Stephenson ve Ali Nesin*
cs@cs.bilgi.edu.tr, anesin@bilgi.edu.tr

Özyinelemeli Fonksiyonlarla Sonsuz Listelerin Sıralanması

Geçen sayıda çabuk işleyen bir sıralama yöntemi görmüştük. Yöntemimiz son derece basitti. Küçük listeleri sıralamayı bildiğimizi varsayarak daha büyük listeleri sıralamıştık, yani sıralamak için matematikçilerin tümevarım yöntemini kullanmıştık. Sıralanacak listeyi önce ikiye bölerek daha kısa iki liste bulmuş, sonra bu iki kısa listeyi (tümevarımla) sıralamış, ardından bu iki sıralı kısa listeyi gene sıralı bir biçimde birleştirmiştik.

Eğer sıralanacak liste (ya da küme ya da dizi) sonsuzsa, yukardaki tümevarımsal yöntem hiçbir işe yaramaz. Bazılarının sandığı gibi, $\infty - 1$ ya da $\infty/2$, ∞'dan daha küçük bir şey değildir! Dolayısıyla sonsuz dizileri sıralamak için başka bir yöntem bulmalıyız.

1915-1998 arasında yaşamış, ünlü matematikçi ve bilgisayar bilimi uzmanı Richard Wesley Hamming şöyle bir problem ortaya atar: Asal çarpanları sadece 2, 3 ve 5 olan tüm sayıların sıralanmış listesini üreten bir program yazılabilir mi? Bu liste şöyle başlar: 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 27, 30, ... Program, bu sayıları küçükten büyüğe üretmeli ve bu özelliğe sahip olmayan sayıları üretmemeli.

Hamming

Elbette ki sonsuz bir küme bu. Yazacağımız program, girdi olarak n sayısını aldığında listenin ilk n elemanını küçükten büyüğe doğru sırayla vermeli.

İlk akla gelen yöntem var, sayıları 2'den başlayarak tek tek kontrol etmek ve özelliği sağlıyorsa listeye eklemek, sağlamıyorsa atmak. Ancak bu yöntem, bize epey vakit kaybettirecektir, çünkü sayılar büyüdükçe, listeye eklenecek sayıların aralıkları da artacaktır, dolayısıyla bir sonraki Hamming sayısını bulmak için gereksiz birçok sayıyı kontrol edip atlamamız gerekecek. Bu yöntem basit ancak pahalı bir yöntemdir. Ne demiş atalarımız: Vakit nakittir!

Her sayıyı kontrol etmeden, sıralamak istediğimiz sayıları küçükten büyüğe sıralı üreten bir yöntem bulmalıyız.

Gereksiz sayıları kontrol etmekten kurtulmak o kadar zor değil. Önce 2'nin, 3'ün ve 5'in katlarını yazalım:

1, 2, 4, 8, 16, ...
1, 3, 9, 27, 81, ...
1, 5, 25, 125, 625, ...

Sonra bu üç listedeki sayıları birbirleriyle çarparak elde ettiğimiz sayıları sıralayalım. Bu fikir de oldukça basit. Bu yazıda bu basit fikri nasıl programlayacağımızı göreceğiz.

Bulacağımız yöntemle çözülen birçok problem vardır. Örneğin,

(a) Kartezyen düzlemde, yani bildiğimiz $\mathbb{R}^2$'de, koordinatları doğal sayı olan noktaların (0, 0) noktasına olan uzaklıklarına göre dizilmesi, ya da,

(b) İki tamsayının küplerinin toplamı olarak iki değişik biçimde yazılan sayıların (yani Ramanujan sayılarının) listesinin bulunması (ki bu sayılardan sonsuz tane olduğu biliniyor) problemleri burada açıklayacağımız yöntemle çözülebilir.

Sonsuz Diziyi Bilgisayara Yüklemek. İlk önce sonsuz bir diziyi bilgisayara yüklemenin yolunu bulmalıyız. Her ne kadar (teorik bir programlama aygıtı olan) Turing makinasına sığacak bilginin uzunluğunun üstsınırı yoksa da, makina belli bir anda ancak sonlu sayıda bilgi barındırabilir. Gerçek, yani fiziksel bilgisayarlarsa gerçekten sınırlıdırlar, barındırabilecekleri bilginin uzunluğunun (evrendeki parçacık sayısı gibi bir) üstsınırı vardır. Bu durumda sonsuz listenin tamamını aynı anda bilgisayara yüklememiz olanaksızdır.

Bu yazıda da geçen yazımızın Scheme programlama dilini kullanacağız. Her ne kadar bu dili http://www.drscheme.org/ sitesinden indirebilirseniz de, buna illa da gerek yok, o programlama di-

* İstanbul Bilgi Üniversitesi öğretim üyesi.

li elinizin altında olmadan da bu yazıdaki programların ne dediğini anlayabilirsiniz. Nitekim, bu konulardan hiç çakmayan ve sadece birinci yazarın asistanlığı görevini üstlenen ikinci yazar, Scheme programlama dilini çalıştırmayı beceremeyen ama yazıyı ve programları çok zorlanmadan anlayabilmiştir.

Akım (stream) ya da daha matematiksel olmak için ***sonsuz dizi*** ya da kısaca ***dizi*** adını vereceğimiz yeni bir liste yapısı yardımımıza koşacak.

Geçen sayımızda kullandığımız sonlu listeler, bir ilk terim ve onu izleyen (listenin kuyruğundaki) bir başka sonlu liste olarak tanımlanmıştı. Kuyruktaki bu sonlu liste duruma göre boş olabiliyordu (eğer tek elemanlı bir listeyle karşı karşıyaysak.) Yani, bir listeyi tanımlamak için,

liste = (listenin ilk terimi, listenin geri kalanı)

eşitliğini kullanmıştık.

Sonsuz dizileri de benzer şekilde iki parçalı elemanlarla göstereceğiz. İlk parça daha önce kullandığımız listelerdeki gibi bir eleman (data) olacak. İkinci kısmı gene kuyruktan oluşacak:

dizi = (dizinin ilk terimi, dizinin geri kalanı)

Ancak bu sefer, yukardakinden farklı olarak dizinin son terimi olmayacak, dolayısıyla program sadece dizinin başlangıcını bilecek, en sonu olmadığından en sonunu bilemeyecek. Öte yandan verilen her n için, program, dizinin n-inci terimin hesaplayabilecek, ama istediğimizde hesaplayacak yalnızca. Bir terim gerekmediği sürece o terimi hesaplamayacak. Böylece sonsuz dizimizin sadece hesaplar için gereken kısmı bilgisayarın aklında tutulacak.

Bu yöntemle sadece belli bir kurala bağlı olarak oluşmuş dizileri bilgisayara yükleyebiliriz. Sonsuza kadar yazı tura atarak elde ettiğiniz 0-1 dizisini bu yöntemle bilgisayara aktaramayız.

Scheme'e diziyi tanımlamak için, geçen sayımızda listelerde kullandığımız cons, car ve cdr fonksiyonlarının benzerleri olan ***scar***, ***scdr*** ve ***scons*** fonksiyonlarını kullanacağız. Anımsatalım: ***car***, listenin ilk terimini bulmaya; ***cdr***, listenin ikinci ve sonraki terimlerini, yani kuyruğunu bulmaya; ***cons*** da bir terimi, bir listenin başına getirerek yeni bir liste oluşturmamıza yarıyordu. ***scar***, ***scdr*** ve ***scons*** komutları sonsuz listeler için aynı görevi görürler. Scheme dilinin bu fonksiyonları anlayabilmesi için önce bunların tanımlanması gerekir elbet. Tanımlar aşağıda. (Bu programları anlamaya çalışmayın, konumuz bu değil. Yazımız için önemli olan, sadece bu tanımların çalışıyor olması. Scheme'de üstteki "Language" komutundan önce "PLT", sonra "Pretty Big" dilini seçin. Aşağıdakini ekranın üst bölümüne yazdıktan sonra sağ taraftaki "Execute" düğmesine basın.)

```
>(define-macro scons (lambda (x y)
  `(cons ,x (delay ,y))))
 (define scar car)
 (define scdr (lambda (s) (force (cdr s))))
```

Artık scons, scar, scdr fonksiyonları tanımlandı.

Yazılımda ve programların anlaşılmasında kolaylık olsun diye, bir a dizisinin terimlerini a_1, a_2, a_3, ... olarak yazacağız. Ayrıca programlarımızda "(scar a)" komutu yerine a1 ve "(scdr a)" komutu yerine a′ yazacağız, yani a′ simgesi a_2, a_3, a_4, ... dizisi anlamına gelecek. Okur, aşağıdaki programları scheme dilinde yazarken a1 yerine (scar a) ve a′ yerine (scdr a) yazmalıdır. Bunun gibi (scar (scdr s)) komutu yerine s2 yazacağız.

Birdizisi. Hemen bir dizi oluşturalım:

```
>(define birdizisi (scons 1 birdizisi))
```

Bu dizi neye benzedi? Birdizisi'nin birinci elemanı 1 ve geri kalan terimleri gene birdizisi'nin, yani gene kendisinin terimleri... Bir başka deyişle,

birdizisi = (1, birdizisi),

yani,

birdizisi = (1, (1, (1, (1, (1, birdizisi))))),

yani 1,1,1,1,1,1,... dizisi. Program

birdizisi = (1, birdizisi)

eşitliğini biliyor ama 1,1,1,1,1,1,... dizisi olduğunu bilmiyor! Belki de görmezden geliyor demek daha doğru olur. Dizinin terimlerini biz istemedikçe hesaplamıyor, yani diziyi bize gösteremiyor, yeterince yeri ve zamanı yok!

Bakalım birdizisi neymiş:

```
>birdizisi
 (1 . #<struct:promise>)
```

İkinci ve sonraki terimleri:

```
>(scdr birdizisi)
 (1 . #<struct:promise>)
```

Ya üçüncü ve sonraki terimleri?

```
>(scdr (scdr birdizisi))
 (1 . #<struct:promise>)
```

Ne kadar ileri gidersek gidelim, listenin başında hep 1 elemanı olacak. Yani birdizisi gerçekten 1'lerden oluşan bir dizi.

İlkn. Çok kullanışlı bir program (ya da fonksiyon) yazalım. Adına ***ilkn*** diyeceğimiz bu program verilen bir *s* dizisinin ilk terimlerini bir liste halinde yazacak. Önce programı yazalım, açıklamasını hemen sonra yapacağız.

```
>(define ilkn (lambda (s n) (cond ((< n 1) '())
 (else (cons s1 (ilkn s' (- n 1)))))))
```

"lambda (s n)" komutundan anlaşılacağı üzere, burada iki değişkenli bir program tanımlıyoruz. Birinci değişken olan *s* bir dizi, ikinci değişken olan *n* ise bir doğal sayı olacak. Bulduğumuz bu ilkn programı bize sonsuz bir *s* dizisinin ilk *n* terimini yazar.

Eğer $n = 0$ ise, program '() ile simgelenen boş-diziyi veriyor. Eğer $n > 0$ ise, program *s* dizisinin ilk *n* terimini (*s* dizisinin ilk terimiyle, ilk terimi attıktan sonra geri kalanının ilk $n - 1$ terimini cons komutuyla birleştirerek) veriyor. Birdizisi'nin ilk 15 terimini isteyelim:

```
>(ilkn birdizisi 15)
 (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
```

Görüldüğü gibi birdizisi dizisi istediğimiz sayıda (yani sonsuz sayıda) 1 içerecek. Böylece 1'lerden oluşan sonsuz bir liste elde ettik.

Stopla. Adına ***stopla*** diyeceğimiz verilen iki diziyi terim terim toplayan bir program yazalım:

```
>(define stopla (lambda (a b) (scons (+ a1 b1) (stopla a' b'))))
```

Yukardaki program, toplanacak olan *a* ve *b* dizilerinin ilk terimlerini toplar ve scons ile bu toplam, $a + b$ dizisinin ilk terimi olarak atanır. $a + b$ dizisinin geri kalanı (kuyruğu yani), *a* ve *b* dizilerinin kuyruklarının, tanımı verilen stopla programıyla toplanmasından oluşur.

Deneyip görelim; birdizisi'ni kendisiyle toplayıp ilk 5 terimi soralım:

```
>(ilkn (stopla birdizisi birdizisi) 5)
 (2 2 2 2 2)
```

Sçarp. Aynen yukardaki yöntemle, stopla yerine sçarp ve + yerine * yazarak iki diziyi çarpabiliriz:

```
>(define sçarp (lambda (a b) (scons (* a1 b1) (sçarp a' b'))))
```

Sfonksiyon2. Yukardaki iki örneği genelleştirebiliriz. Toplama ve çarpma gibi iki değişkenli bir *f* fonksiyonumuz ve *f* fonksiyonunu terimlerine uygulayabileceğimiz iki *a* ve *b* dizimiz olsun.

$$f(a_1, b_1),\ f(a_2, b_2),\ f(a_3, b_3),\ \ldots$$

dizisini oluşturmak istiyoruz. Aynen yukardaki gibi yapabiliriz:

```
>(define sfonksiyon2 (lambda (f a b)
 (scons (f a1 b1) (sfonksiyon2 f a' b'))))
```

Görüldüğü gibi sfonksiyon2'nin üç değişkeni var: iki değişkenli bir fonksiyon ve iki dizi. Programın verdiği sonuç aynen yukardaki dizi oluyor. Örneğin, yukardaki stopla fonksiyonunu şimdi (sfonksiyon2 + a b) olarak yazabiliriz.

```
>(ilkn (sfonksiyon2 + birdizisi birdizisi) 5)
 (2 2 2 2 2)
```

Sfonksiyon1. Yukarda yaptığımızı iki değişkenli bir *f* fonksiyonu yerine bir değişkenli bir *f* fonksiyonuyla yapabiliriz elbet. Eğer $a = (a_1, a_2, a_3, \ldots)$ bir diziyse ve *f* bir değişkenli bir fonksiyonsa, *f* fonksiyonunu *a* dizisinin her terimine uygulayarak yeni bir

$$f(a) = (f(a_1), f(a_2), f(a_3), \ldots)$$

dizisi elde edebiliriz. Şimdi bu yeni dizi elde etme yöntemini bilgisayara öğretelim.

```
> (define sfonksiyon1 (lambda (f s)
  (scons (f s1) (sfonksiyon1 f s'))))
```

"lambda (f s)" komutundan anlaşıldığı üzere, sfonksiyon1 fonksiyonu, bir *f* fonksiyonuna ve bir *s* dizisine uygulanıyor ve yeni bir dizi elde ediliyor. Yeni dizi, *f* fonksiyonu *s* dizisinin her terimine uygulanarak elde ediliyor. Örneğin, *s*, sayılardan oluşan bir listeyse ve *f*, 2'yle çarpmak fonksiyonuysa, yani (lambda (x) (* x 2)) fonksiyonuysa, yukardaki yöntemle listenin her terimini 2'yle çarpabiliriz. sfonksiyon1 yöntemini hemen uygulayalım:

```
>(ilkn (sfonksiyon1 (lambda (x) (* x 2)) birdizisi) 5)
 (2 2 2 2 2)
```

Tamsayılar. Birdizisi çok ilginç bir örnek olmadı. Şuna ne dersiniz?

```
>(define budane (scons 1 (stopla birdizisi budane)))
```

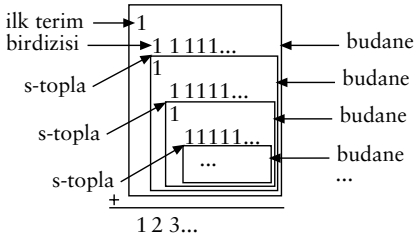
Bu dizinin ne olduğunu görebildiniz mi?

$$\text{budane} = (1, \text{birdizisi} + \text{budane})$$

yani,

$$\text{budane} = (1, 1111\ldots + \text{budane})$$

Altalta yazacak olursak budane'yi daha kolay buluruz:



Budane'nin birinci terimi 1. Bu, programdan belli. (n+1)-inci terimiyse, birdizisiyle budane dizilerinin n-inci terimlerinin toplamı... Demek ki ikinci terimi 1 + 1 = 2, üçüncü terimi 1 + 2 = 3, ...

Bakalım:

```
>(ilkn budane 10)
 (1 2 3 4 5 6 7 8 9 10)
```

Pozitif tamsayılar... Hem de hepsi!.. Dizimize budane yerine daha uygun bir ad verelim:

```
>(define tamsayılar (scons 1 (stopla birdizisi tamsayılar)))
```

Katları. Tamsayılar dizisinin tüm terimlerini bir n sabitiyle çarpıp (n, $2n$, $3n$, $4n$, ...) dizisini elde edelim:

```
>(define katları (lambda (n)
 (sfonksiyon1 (lambda (x) (* x n)) tamsayılar)))
```

Böylece, tamsayılar dizisinin tüm terimlerini n sabitiyle çarpmış oluruz. n = 7 için deneyelim:

```
>(ilkn (katları 7) 13)
 (7 14 21 28 35 42 49 56 63 70 77 84 91)
```

Kareler ve küpler. Doğal sayıların üçüncü güçlerinden oluşan 1, 8, 27, 81, 243, ... dizisini oluşturalım.

```
(define kare (lambda (x) (* x x)))
(define küp (lambda (x) (* x x x)))
(define kareler (sfonksiyon1 kare tamsayılar))
(define küpler (sfonksiyon1 küp tamsayılar))
>(ilkn küpler 6)
(1 8 27 64 125 216)
```

***n*'nin Güçleri.** Yazının başında da söylediğimiz gibi, Hamming'in problemini çözmek için,

1, 2, 4, 8, 16, ...

1, 3, 9, 27, 81, ...

1, 5, 25, 125, 625, ...

gibi belli bir n sayısının güçlerinden oluşan

$$1, n, n^2, n^3, n^4, n^5, \ldots$$

dizisine ihtiyacımız olacak. Hemen yazalım.

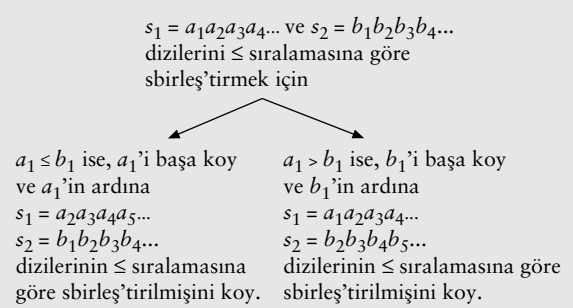
```
>(define güçler (lambda (n) (scons 1
(sfonksiyon1 (lambda (x) (* x n)) (güçler n)))))
>(ilkn (güçler 3) 8)
(1 3 9 27 81 243 729 2187)
```

Sbirleş. Küçükten büyüğe sıralanmış iki dizimiz olsun. Bunları birleştirebilir miyiz? Evet! Geçen sayıda gördüğümüz, sonlu listelerin birleştirilmesi için kullanılan yöntem bize yetecek. Ama dizinin sonunu kontrol edemeyiz ve etmeyeceğiz de, dizinin sonu yok ki kontrol edelim! Bir önceki sayıda sözünü ettiğimiz "birleştir" yöntemini kullanacağız. Bu yöntemin adını artık ***sbirleş*** olarak değiştiriyoruz.

Eğer "sıra" **herhangi** bir sıralamaysa (buradaki "herhangi" sıfatı önemli olacak, örneğin "sıra" sayılar arasındaki aşina olduğumuz < sıralaması olabilir), a ve b dizilerini şöyle birleştirebiliriz:

```
>(define sbirleş (lambda (sıra a b)
 (cond ((sıra a1 b1) (scons a1 (sbirleş sıra a′ b)))
 (else (scons b1 (sbirleş sıra a b′))))))
```

Bu programın ne yaptığını aşağıdaki gri karede de görebiliriz. lambda (sıra a b) komutundan da anlaşılacağı üzere, sbirleş fonksiyonunun, bir "sıra" sıralamasına ve iki diziye ihtiyacı var. Eğer a ve b dizileri "sıra" sıralamasıyla dizilmişlerse, bu program sayesinde gerçekten de "sıra" sıralamasıyla birleştirilerek dizilmiş bir dizi elde ederiz[1].



Şimdi 3'ün katlarından oluşan (katları 3) sıralı dizisiyle 5'in katlarından oluşan (katları 5) sıralı dizisini sbirleş kullanarak birleştirelim. (Yukardaki "sıra" sıralaması yerine bildiğimiz < sıralamasını kullanacağız elbet.)

```
>(ilkn (sbirleş < (katları 3) (katları 5)) 16)
 (3 5 6 9 10 12 15 15 18 20 21 24 25 27 30 30)
```

3'ün ve 5'in katlarından oluşan iki sonsuz diziyi sıraladık. Bu programda 3 × 5 = 15'in katları iki kez yazılıyor, ama şimdilik umursamayalım bunu.

İkiden fazla, diyelim üç diziyi birleştirmek de çok kolay. Önce ikisini birleştirip, sonucu diğeriyle birleştiririz.

```
>(ilkn (sbirleş < (katları 7) (sbirleş < (katları 3) (katları 5))) 21)
 (3 5 6 7 9 10 12 14 15 15 18 20 21 21 24 25 27 28 30 30 33)
```

Böylece kendi içinde sıralı dizileri, yine sıralı biçimde birleştirebiliriz.

1 Burada, aşina olduğumuz < sıralaması yerine herhangi bir başka sıralama, hatta bir yarısıralama bile kullanabilirdik. Tanımlanan sbirleş fonksiyonu her yarısıralama için iki listeyi o yarısıralamaya göre birleştirir.

Haming Sayıları. Bu bölümde Hamming'in sorusunu çözeceğiz. Önce daha basit bir problemi ele alalım: (3, 5), (2, 2), (7, 4) gibi pozitif doğal sayı çiftlerini bir biçimde sıralayalım. Bu soruyu çözdüğümüzde, buna benzer soruların çözümü için adım atmış olacağız ve özellikle Hamming'in sorusunun çözümüne yaklaşmış olacağız.

Sıralamak istediğimiz sayı çiftleri şunlar:

	1	2	3	4	5	...
1	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	...
2	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	...
3	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	...
4	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	...
5	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	...
	...	...	...	...	...	...

Bu sayı çiftlerini hangi sıralamaya göre sıralayacağımızı bilmemiz gerekiyor elbet. Ne de olsa, sayı çiftleri doğal bir biçimde sıralanmış olarak karşımıza çıkmazlar. Sıralamayı aşağıdaki tablo-

	1	2	3	4	5	...
1	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	...
2	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	...
3	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	...
4	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	...
5	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	...
	...	...	...	...	...	...

daki gibi çapraz yapabiliriz. Yani sıralamamız şöyle olabilir: (1, 1) ∠ (2, 1) ∠ (1, 2) ∠ (3, 1) ∠ (2, 2) ∠ (1, 3) ∠ (4, 1) ∠ (3, 2) ∠ (2, 3) ∠ (1, 4) ∠ (5, 1) ∠ (4, 2) ∠ (3, 3) ∠ (2, 4) ∠ (1, 5) ∠ (6, 1) ∠ ...

Bu sıralamaya ***çapraz sıralama*** diyelim.

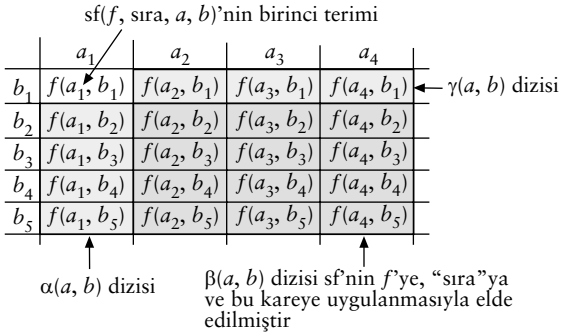
	a_1	a_2	a_3	a_4
1	$f(a_1, b_1)$	$f(a_2, b_1)$	$f(a_3, b_1)$	$f(a_4, b_1)$
2	$f(a_1, b_2)$	$f(a_2, b_2)$	$f(a_3, b_2)$	$f(a_4, b_2)$
3	$f(a_1, b_3)$	$f(a_2, b_3)$	$f(a_3, b_3)$	$f(a_4, b_3)$
4	$f(a_1, b_4)$	$f(a_2, b_4)$	$f(a_3, b_4)$	$f(a_4, b_4)$
5	$f(a_1, b_5)$	$f(a_2, b_5)$	$f(a_3, b_5)$	$f(a_4, b_5)$

Şimdi yukardaki fikri programlayabileceğimizi göreceğiz. Elimiz değmişken, sadece (i, j) sayı çiftlerini değil, iki değişkenli bir f fonksiyonu ve iki a ve b dizisi için, $(f(a_i, b_j))_{i,j}$ familyasını yukarıdaki tabloda görüldüğü gibi çapraz ya da herhangi bir başka sıralanmayla dizilmiş bir dizi halinde yazacağız. Buradaki f fonksiyonu, $f(x, y) = xy$ ya da $f(x, y) = (x, y)$ gibi herhangi bir fonksiyon olabilir.

Adına sf diyeceğimiz ve değerleri diziler olan bir fonksiyonun programını yazacağız. sf'nin,

- bir f fonksiyonu,
- bir "sıra" sıralaması ve
- iki a ve b dizisi

olmak üzere dört değişkeni olacak. Buradaki f, tekrar ediyoruz, toplama, çarpma ya da cons gibi a ve b'nin terimlerine uygulanabilen iki değişkenli herhangi bir fonksiyon olabilir. $sf(f$, sıra, a, $b)$ dizisi $(f(a_i, b_j))_{i,j}$ familyasını "sıra" sıralamasıyla sıraya dizecek, yani bir dizi haline dönüştürecek, sa-



dece dizinin birinci terimi $f(a_1, b_1)$ olacak. $sf(f$, sıra, a, $b)$ dizisinin kuyruğunu bulmak için yukarıdaki şekildeki α, β ve γ dizilerini α * (β * γ) sırasıyla sbirleştireceğiz, yani,

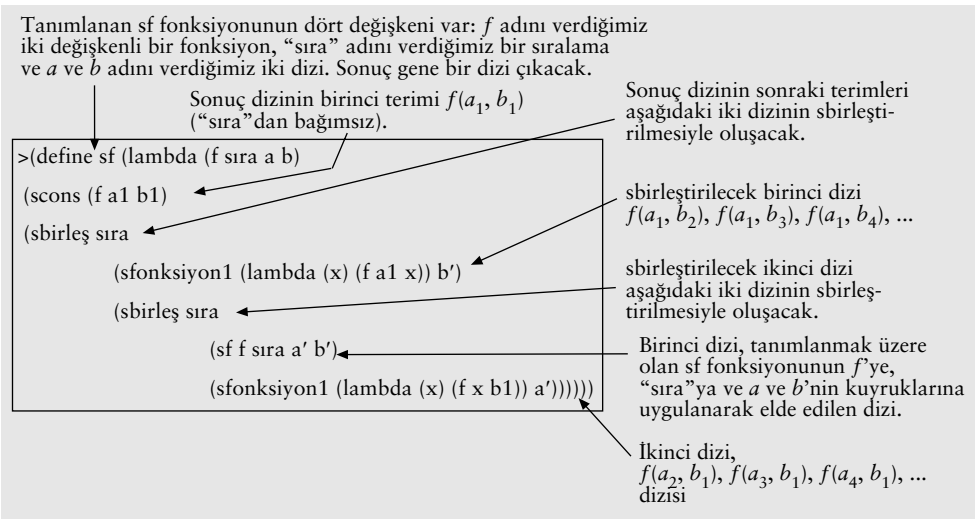
sf(f, sıra a, b) = ($f(a_1, b_1)$, sbirleş(α, sbirleş(β, γ)))

olacak.

İşte program:

```
>(define sf (lambda (f sıra a b) (scons (f a1 b1)
  (sbirleş sıra (sfonksiyon1 (lambda (x) (f a1 x)) b′)
  (sbirleş sıra (sf f sıra a′ b′)
  (sfonksiyon1 (lambda (x) (f x b1)) a′))))))
```

ve işte açıklaması:



İlk 19 Hamming sayısını görelim:

```
>(ilkn (sf * < (güçler 7) (sf * < (güçler 5) (güçler 3))) 19)
(1 3 5 7 9 15 21 25 27 35 45 49 63 75 81 105 125 135 147)
```

Bu yöntemin başka problemlere de uygulanabilirliği üzerine birkaç örnek görelim. Önce kolay bir problemi ele alacağız, sonra problemimiz daha ilginç olacak.

Sayı Çiftlerini Sıralamak. Sayı çiftlerini çapraz sıralamayla sıralayalım/dizelim. Çapraz sıralamaya bir daha bakalım: (1, 1) ∠ (1, 2) ∠ (2, 1) ∠ (1, 3) ∠ (2, 2) ∠ (3, 1) ∠ (1, 4) ∠ (2, 3) ∠ (3, 2) ∠ (4, 1) ∠ (1, 5) ∠ (2, 4) ∠ (3, 3) ∠ ...

(a, b) sayı çiftinin ***derece***si $a + b$ olsun. Örneğin:

Derecesi 2 olan: (1, 1).

Derecesi 3 olan: (1, 2) ve (2, 1).

Derecesi 4 olan: (1, 3), (2, 2) ve (3, 1).

Görüldüğü gibi çapraz sıralamada sayı çiftleri önce derecelerine göre sıralanmışlar; aynı derecedeki sayı çiftleri daha sonra ikinci koordinatlarına göre sıralanmışlar. Çözmek istediğimiz probleme göre bir başka derecelendirme de alabilirdik. Yukardaki derece kavramına ***çapraz-derece*** diyelim. İşte çapraz-derece fonksiyonunun programı:

```
>(define çapraz-derece (lambda (p) (+ (car p) (cdr p))))
```

Anlaşılacağı üzere, $p = (p_1, p_2)$ çiftiyse,

çapraz-derece(p) = $p_1 + p_2$ oluyor.

Derecelendirme fonksiyonunu bir (yarı-)sıralamaya dönüştürelim:

```
>(define derece->sıra (lambda (derece) (lambda (a b)
 (< (derece a) (derece b)))))
```

Yukardaki derece-sıra aslında "evet" ya da "hayır" yanıtlarından birini veren bir fonksiyon. Girdi olarak bir derece fonksiyonu veriliyor. Eğer derece(a) < derece(b) ise yanıt "evet" oluyor, yoksa "hayır" oluyor.

Artık çapraz sıralanmış sayı çiftleri dizisini oluşturabiliriz:

```
>(ilkn (sf cons (derece->sıra çapraz-derece) tamsayılar tamsayılar) 9)
 ((1 . 1) (2 . 1) (1 . 2) (3 . 1) (2 . 2) (1 . 3) (4 . 1) (3 . 2) (2 . 3))
```

İşin güzel tarafı, bu diziyi değişik biçimde sıralamak istediğimizde tek yapmamız gereken sıralamayı değiştirmek olacaktır. Mesela, tamsayı ikililerimizi bir düzlemdeki koordinatlar olarak yorumlayıp, orijine uzaklıklarına göre sıralamak isteyebiliriz.

```
(define mesafe-derece (lambda (p) (+ (kare (car p)) (kare (cdr p)))))
>(ilkn (sf cons (derece->sıra mesafe-derece) tamsayılar tamsayılar) 9)
 ((1 . 1) (2 . 1) (1 . 2) (2 . 2) (3 . 1) (1 . 3) (3 . 2) (2 . 3) (4 . 1))
```

Ramanujan Sayıları. Ramanujan ve Hardy'nin meşhur hikâyesini bilirsiniz, bilmiyorsanız yazının sonundaki gri karede bu hikâyeyi bulabilirsiniz. İki küpün toplamı olarak iki ayrı şekilde yazılabilen sayılara ***Ramanujan sayıları*** diyoruz. Örneğin,

$$1729 = 12^3 + 1^3 = 10^3 + 9^3$$

eşitliğinden dolayı, 1729 bir Ramanujan sayısıdır.

Amacımız Ramanujan sayılarını bir dizi halinde yazmak. Yazacağımız program, tüm olası küp toplamlarından bir dizi oluşturacak ve bir süzgeç ile çift olanları eleyecek.

Doğal sayıların küplerinden oluşan (a^3, b^3) çiftlerini alacağız. Bunları sıraya dizmesini biliyoruz. Bu çiftlerin koordinatlarını toplayıp $a^3 + b^3$ sayısını elde edeceğiz ve tekrar eden değerleri bulacağız. Ancak (a^3, b^3) ve (b^3, a^3) çiftini ayrı ayrı sayarsak, o zaman $a \neq b$ için (a^3, b^3) ve (b^3, a^3) aynı toplamı verir. Bu yüzden ikilileri oluşturma yöntemimizi biraz değiştireceğiz.

```
>(define sf-uops (lambda (f sıra a b)
 (scons (f a1 b1) (sbirleş sıra (sfonksiyon1
            (lambda (x) (f a1 x)) b′) (sf-uops f sıra a′ b′)))))
>(ilkn (sf-uops cons (derece->sıra çapraz-derece)
                              tamsayılar tamsayılar) 9)
((1 . 1) (1 . 2) (2 . 2) (1 . 3) (2 . 3) (1 . 4) (3 . 3) (2 . 4) (1 . 5))
```

Artık sadece (1,2) üretiliyor ve (2,1) atlanıyor. Tüm ikililer için özyinelemeli olarak bu yapıldığında, tam istediğimiz gibi bir dizimiz olacaktır.

Bir dizinin içinde birden fazla geçen terimleri yakalayan bir fonksiyon yazalım:

```
>(define tekrar-eden-değerleri-bul (lambda (s)
 (cond ((= s1 s2) (scons s1 (tekrar-eden-değerleri-bul s′)))
 (else (tekrar-eden-değerleri-bul s′)))))
```

(Not: Yukardaki programdaki s2, (scar (scdr s)) anlamına geldiğini anımsatırız.)

Artık Ramanujan sayılarından bir dizi oluşturabiliriz:

```
>(ilkn (tekrar-eden-değerleri-bul
 (sf-uops + < küpler küpler)) 16)
 (1729 4104 13832 20683 32832 39312 40033 46683 64232
65728 110656 110808 134379 149389 165464 171288)
```

Dizimizin ilk terimi, Hardy'nin hiç ilginç bulmadığı taksi plakası olan 1729.

Ramanujan sayılarını oluşturan tamsayı ikililerinden bir dizi oluşturma sorusunu okura bırakıyo-

ruz. Kendinden önce gelen Ramanujan sayılarıyla bağlantılı olanları elemek de güzel problem olabilir. Örneğin 13832, $24^3 + 2^3$ ve $20^3 + 18^3$ olarak yazılır. İlk Ramanujan sayısı olan 1729'u oluşturan sayıları 2'yle çarpınca bu değerleri elde ediyoruz. Daha birçok şey deneyebiliriz. Üç sayının küpleri toplamı olarak iki farklı şekilde yazılabilen sayıları bulmak gibi. Veya iki sayının küpleri toplamı olarak üç ayrı şekilde ifade edilen sayılar araştırılabilir.

Şöyle bir diyalog geçtiğini düşünsenize:

Hardy: Bindiğim taksinin plakası çok saçmaydı, 87539319.

Ramanujan: Hayır Hardy! Hayır Hardy! Bu çok ilginç bir sayıdır. İki sayının küplerinin toplamı olarak üç ayrı şekilde yazılabilen en küçük sayı 87539319'dur: $87539319 = 255^3 + 414^3 = 228^3 + 423^3 = 167^3 + 436^3$.

Biraz önce yazdığımız program üzerinde birkaç değişiklik yeni problemimizi de çözecektir.

Sadece özyineleme kullanarak yaptıklarımız dikkat çekicidir; programcılığın özü olarak kabul edilen döngü ve değer aktarma (assignment) araçları hiç kullanmadan zor bir problemi çözdük. Kullanabileceğimiz araçları kısıtlayarak daha zor problemler çözebildik. Bu tanımları ve testleri yüklemek için: http://cs.bilgi.edu.tr/pages/academic_staff/lecturers/chris_stephenson/hamming2004.html♦

Ramanujan (1887-1920)

Bir Hint dahisidir. Analitik sayılar kuramına (örneğin sonsuz toplamlara ve çarpımlara) çok önemli katkıları olmuştur. 15 yaşında üçüncü ve dördüncü dereceden denklemleri çözmeyi başardı. Üniversiteye gitmedi. 23 yaşında, bulduğu sonuçları ünlü İngiliz matematikçisi Hardy'ye yolladı. Hardy'yle yakın arkadaşı Littlewood mektubu uzun süre dikkatle incelediler ve Ramanujan'a sıcak bir mektup yazdılar. 27 yaşında Londra'ya Hardy'nin yanına geldi. Ancak İngiliz mutfağına alışamadı. Birinci Dünya Savaşı da özel besinlerin bulunmasını zorlaştırıyordu. Sağlık problemleri başgösterdi. 1917'de ciddi olarak rahatsızlandı ve 1920'de öldü.

Ramanujan hasta yatağında yatarken, Hardy ziyaretine gelir. Hastasının dikkatini sağlıktan başka konulara çekmek isteyen Hardy, gelirken bindiği taksinin numarasının 1729 olduğunu, bu sayının da hiç ilginç olmadığını söyler. Ramanujan,

– Hayır Hardy, der, çok ilginç bir sayıdır bu! İki kübün toplamı olarak iki değişik biçimde yazılan en küçük sayıdır...

O günden beri bu tür sayılara ***taksi sayıları*** da denir!

Yazıda Kullanılan Tanımlar

```
(define-macro scons (lambda (x y) `(cons ,x (delay ,y))))
(define scar car)
(define scdr (lambda (s) (force (cdr s))))
(define birdizisi (scons 1 birdizisi))
(define ilkn (lambda (s n) (cond ((< n 1) '()) (else (cons (scar s) (ilkn (scdr s) (- n 1)))))))
(define stopla (lambda (a b) (scons (+ (scar a) (scar b)) (stopla (scdr a) (scdr b)))))
(define sçarp (lambda (a b) (scons (* (scar a) (scar b)) (sçarp (scdr a) (scdr b)))))
(define sfonksiyon2 (lambda (f a b) (scons (f (scar a) (scar b)) (sfonksiyon2 f (scdr a) (scdr b)))))
(define sfonksiyon1 (lambda (f a) (scons (f (scar a)) (sfonksiyon1 f (scdr a)))))
(define tamsayılar (scons 1 (stopla birdizisi tamsayılar)))
(define katları (lambda (n) (sfonksiyon1 (lambda (x) (* x n)) tamsayılar)))
(define kare (lambda (x) (* x x)))
(define küp (lambda (x) (* x x x)))
(define kareler (sfonksiyon1 kare tamsayılar))
(define küpler (sfonksiyon1 küp tamsayılar))
(define güçler (lambda (n) (scons 1 (sfonksiyon1 (lambda (x) (* x n)) (güçler n)))))
(define sbirleş (lambda (sıra a b) (cond ((sıra (scar a) (scar b)) (scons (scar a) (sbirleş sıra (scdr a) b)))
        (else (scons (scar b) (sbirleş sıra a (scdr b)))))))
(define sf (lambda (f sıra a b) (scons (f (scar a) (scar b)) (sbirleş sıra (sfonksiyon1 (lambda (x) ( f (scar a) x)) (scdr b))
        (sbirleş sıra (sf f sıra (scdr a) (scdr b)) (sfonksiyon1 (lambda (x) (f x (scar b))) (scdr a)))))))
(define çapraz-derece (lambda (p) (+ (car p) (cdr p))))
(define derece->sıra (lambda (derece) (lambda (a b) (< (derece a) (derece b)))))
(define mesafe-derece (lambda (p) (+ (kare (car p)) (kare (cdr p)))))
(define sf-uops (lambda (f sıra a b) (scons (f (scar a) (scar b)) (sbirleş sıra (sfonksiyon1 (lambda (x) (f (scar a) x)) (scdr b))
        (sf-uops f sıra (scdr a)(scdr b))))))
(define tekrar-eden-değerleri-bul (lambda (s) (cond ((= (scar s) (scar (scdr s)))
        (scons (scar s) (tekrar-eden-değerleri-bul (scdr s)))) (else (tekrar-eden-değerleri-bul (scdr s))))))
```